SoMachine
FileFormatUtility Library Guide

06/2017

EIO0000002530.00

www.schneider-electric.com

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

You agree not to reproduce, other than for your own personal, noncommercial use, all or part of this document on any medium whatsoever without permission of Schneider Electric, given in writing. You also agree not to establish any hypertext links to this document or its content. Schneider Electric does not grant any right or license for the personal and noncommercial use of the document or its content, except for a non-exclusive license to consult it on an "as is" basis, at your own risk. All other rights are reserved.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

© 2017 Schneider Electric. All Rights Reserved.

Table of Contents

Safety Information 5
About the Book 9
Part I General Information 15
Chapter 1 Presentation of the Library 17
General Information 17
Chapter 2 Common Inputs and Outputs 21
Behavior of Function Blocks with the Input `i_xExecute` 21
Part II Data Unit Types 23
Chapter 3 Enumerations 25
`ET_XmlItemType` 26
`ET_CsvReadMode` 27
`ET_ModeFileOpen` 28
`ET_Result` 29
Chapter 4 Structures 33
`ST_XmlItem` 34
`ST_XmlUserDefinedHeader` 36
`ST_CsvTable` 37
`ST_CsvFileInformation` 39
`ST_CsvWarnValueTruncated` 40
`ST_CsvReadParameter` 41
`ST_CsvWriteParameter` 42
Chapter 5 Aliases 43
Aliases 43
Part III Global Variables 45
Chapter 6 Global Constants List 47
Global Constants List (GCL) 47
Chapter 7 Global Parameter List 49
GPL 49
Part IV Global Functions 51
Chapter 8 Global Functions 53
`FC_EtResultToString` 53

Part V XML Program Organization Units (POU) 55
Chapter 9 XML Function Blocks 57
9.1 FB_XmlRead 58
FB_XmlRead Functional Description 59
FB_XmlRead Considerations 62
FB_XmlRead Troubleshooting 63
FB_XmlRead Example 64
9.2 FB_XmlWrite 66
FB_XmlWrite Functional Description 67
FB_XmlWrite Considerations 69
FB_XmlWrite Troubleshooting 70
FB_XmlWrite Example 71
Chapter 10 XML Functions 73
10.1 FC_XmlGetElementValue 74
FC_XmlGetElementValue Functional Description 75
FC_XmlGetElementValue Considerations 77
10.2 FC_XmlSetElementValue 78
FC_XmlSetElementValue Functional Description 79
FC_XmlSetElementValue Considerations 80
Part VI CSV Program Organization Units (POU) 81
Chapter 11 CSV Function Blocks 83
11.1 FB_CsvRead 84
FB_CsvRead Functional Description 85
FB_CsvRead Considerations 88
FB_CsvRead Troubleshooting 89
FB_CsvRead Example 90
11.2 FB_CsvWrite 92
FB_CsvWrite Functional Description 93
FB_CsvWrite Considerations 95
FB_CsvWrite Troubleshooting 96
FB_CsvWrite Example 97
Glossary 99
Index 101

Safety Information

Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, service, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result** in minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

BEFORE YOU BEGIN

Do not use this product on machinery lacking effective point-of-operation guarding. Lack of effective point-of-operation guarding on a machine can result in serious injury to the operator of that machine.

⚠ WARNING

UNGUARDED EQUIPMENT

- Do not use this software and related automation equipment on equipment which does not have point-of-operation protection.
- Do not reach into machinery during operation.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

This automation equipment and related software is used to control a variety of industrial processes. The type or model of automation equipment suitable for each application will vary depending on factors such as the control function required, degree of protection required, production methods, unusual conditions, government regulations, etc. In some applications, more than one processor may be required, as when backup redundancy is needed.

Only you, the user, machine builder or system integrator can be aware of all the conditions and factors present during setup, operation, and maintenance of the machine and, therefore, can determine the automation equipment and the related safeties and interlocks which can be properly used. When selecting automation and control equipment and related software for a particular application, you should refer to the applicable local and national standards and regulations. The National Safety Council's Accident Prevention Manual (nationally recognized in the United States of America) also provides much useful information.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the operator's hands and other parts of the body are free to enter the pinch points or other hazardous areas and serious injury can occur. Software products alone cannot protect an operator from injury. For this reason the software cannot be substituted for or take the place of point-of-operation protection.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

NOTE: Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the Function Block Library, System User Guide, or other implementation referenced in this documentation.

START-UP AND TEST

Before using electrical control and automation equipment for regular operation after installation, the system should be given a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such a check be made and that enough time is allowed to perform complete and satisfactory testing.

⚠ WARNING

EQUIPMENT OPERATION HAZARD

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices.
- Remove tools, meters, and debris from equipment.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Follow all start-up tests recommended in the equipment documentation. Store all equipment documentation for future references.

Software testing must be done in both simulated and real environments.

Verify that the completed system is free from all short circuits and temporary grounds that are not installed according to local regulations (according to the National Electrical Code in the U.S.A, for instance). If high-potential voltage testing is necessary, follow recommendations in equipment documentation to prevent accidental equipment damage.

Before energizing equipment:

- Remove tools, meters, and debris from equipment.
- Close the equipment enclosure door.
- Remove all temporary grounds from incoming power lines.
- Perform all start-up tests recommended by the manufacturer.

OPERATION AND ADJUSTMENTS

The following precautions are from the NEMA Standards Publication ICS 7.1-1995 (English version prevails):

- Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly operated.
- It is sometimes possible to misadjust the equipment and thus produce unsatisfactory or unsafe operation. Always use the manufacturer's instructions as a guide for functional adjustments. Personnel who have access to these adjustments should be familiar with the equipment manufacturer's instructions and the machinery used with the electrical equipment.
- Only those operational adjustments actually required by the operator should be accessible to the operator. Access to other controls should be restricted to prevent unauthorized changes in operating characteristics.

About the Book

At a Glance

Document Scope

This document describes the library FileFormatUtility.

The FileFormatUtility library implements functions which provide simplified access to files of certain formats.

The following file formats are supported:
- XML (eXtensible Markup Language)
- CSV (Comma Separated Values)

The FileFormatUtility library is supported by the following controllers:

- Modicon M241 Logic Controller
- Modicon M251 Logic Controller
- Modicon M258 Logic Controller
- Modicon LMC078 Motion Controller
- Modicon LMC058 Motion Controller

Validity Note

This document has been updated for the release of SoMachine V4.3.

The technical characteristics of the devices described in this document also appear online. To access this information online:

Step	Action
1	Go to the Schneider Electric home page *www.schneider-electric.com*.
2	In the **Search** box type the reference of a product or the name of a product range. ● Do not include blank spaces in the reference or product range. ● To get information on grouping similar modules, use asterisks (*).
3	If you entered a reference, go to the **Product Datasheets** search results and click on the reference that interests you. If you entered the name of a product range, go to the **Product Ranges** search results and click on the product range that interests you.
4	If more than one reference appears in the **Products** search results, click on the reference that interests you.
5	Depending on the size of your screen, you may need to scroll down to see the data sheet.
6	To save or print a data sheet as a .pdf file, click **Download XXX product datasheet**.

The characteristics that are presented in this manual should be the same as those characteristics that appear online. In line with our policy of constant improvement, we may revise content over time to improve clarity and accuracy. If you see a difference between the manual and online information, use the online information as your reference.

Product Related Information

⚠ WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.

Before you attempt to provide a solution (machine or process) for a specific application using the POUs found in the library, you must consider, conduct and complete best practices. These practices include, but are not limited to, risk analysis, functional safety, component compatibility, testing and system validation as they relate to this library.

⚠ WARNING

IMPROPER USE OF POUS

- Perform a safety-related analysis for the application and the devices installed.
- Ensure that the POUs are compatible with the devices in the system and have no unintended effects on the proper functioning of the system.
- Use appropriate parameters, especially limit values, and observe machine wear and stop behavior.
- Verify that the sensors and actuators are compatible with the selected POUs.
- Thoroughly test all functions during verification and commissioning in all operation modes.
- Provide independent methods for critical control functions (emergency stop, conditions for limit values being exceeded, etc.) according to a safety-related analysis, respective rules, and regulations.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Incomplete file transfers, such as data files, application files and/or firmware files, may have serious consequences for your machine or controller. If you remove power, or if there is a power outage or communication interruption during a file transfer, your machine may become inoperative, or your application may attempt to operate on a corrupted data file. If a an interruption occurs, reattempt the transfer. Be sure to include in your risk analysis the impact of corrupted data files.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION, DATA LOSS, OR FILE CORRUPTION

- Do not interrupt an ongoing data transfer.
- If the transfer is interrupted for any reason, re-initiate the transfer.
- Do not place your machine into service until the file transfer has completed successfully, unless you have accounted for corrupted files in your risk analysis and have taken appropriate steps to prevent any potentially serious consequences due to unsuccessful file transfers.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

The POUs provided with this library use variables of type POINTER TO internally. These pointers are assigned only on start of execution of the respective POU. This is, the pointers are not reassigned while the function block indicates `Busy`.

⚠ CAUTION
INVALID POINTER Do not use the “Online Change” command or the “Log in with online change” option as long as one of the function blocks of this library indicates `Busy` in your running application. **Failure to follow these instructions can result in injury or equipment damage.**

Related Documents

Document title	Reference
SoMachine Functions and Libraries User Guide	*EIO0000000735 (ENG)*; *EIO0000000792 (FRE)*; *EIO0000000793 (GER)*; *EIO0000000795 (SPA)*; *EIO0000000794 (ITA)*; *EIO0000000796 (CHS)*
SoMachine Programming Guide	*EIO0000000067 (ENG)*; *EIO0000000069 (FRE)*; *EIO0000000068 (GER)*; *EIO0000000071 (SPA)*; *EIO0000000070 (ITA)*; *EIO0000000072 (CHS)*

You can download these technical publications and other technical information from our website at *http://www.schneider-electric.com/en/download*.

Terminology Derived from Standards

The technical terms, terminology, symbols and the corresponding descriptions in this manual, or that appear in or on the products themselves, are generally derived from the terms or definitions of international standards.

In the area of functional safety systems, drives and general automation, this may include, but is not limited to, terms such as *safety*, *safety function*, *safe state*, *fault*, *fault reset*, *malfunction*, *failure*, *error*, *error message*, *dangerous*, etc.

Among others, these standards include:

Standard	Description
EN 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2008	Safety of machinery: Safety related parts of control systems. General principles for design.
EN 61496-1:2013	Safety of machinery: Electro-sensitive protective equipment. Part 1: General requirements and tests.
ISO 12100:2010	Safety of machinery - General principles for design - Risk assessment and risk reduction
EN 60204-1:2006	Safety of machinery - Electrical equipment of machines - Part 1: General requirements
EN 1088:2008 ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2006	Safety of machinery - Emergency stop - Principles for design
EN/IEC 62061:2005	Safety of machinery - Functional safety of safety-related electrical, electronic, and electronic programmable control systems
IEC 61508-1:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: General requirements.
IEC 61508-2:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Requirements for electrical/electronic/programmable electronic safety-related systems.
IEC 61508-3:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems: Software requirements.
IEC 61784-3:2008	Digital data communication for measurement and control: Functional safety field buses.
2006/42/EC	Machinery Directive
2014/30/EU	Electromagnetic Compatibility Directive
2014/35/EU	Low Voltage Directive

In addition, terms used in the present document may tangentially be used as they are derived from other standards such as:

Standard	Description
IEC 60034 series	Rotating electrical machines
IEC 61800 series	Adjustable speed electrical power drive systems
IEC 61158 series	Digital data communications for measurement and control – Fieldbus for use in industrial control systems

Finally, the term *zone of operation* may be used in conjunction with the description of specific hazards, and is defined as it is for a *hazard zone* or *danger zone* in the *Machinery Directive* (*2006/42/EC*) and *ISO 12100:2010*.

NOTE: The aforementioned standards may or may not apply to the specific products cited in the present documentation. For more information concerning the individual standards applicable to the products described herein, see the characteristics tables for those product references.

Part I
General Information

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
1	Presentation of the Library	17
2	Common Inputs and Outputs	21

Chapter 1
Presentation of the Library

General Information

Library Overview

The FileFormatUtility library implements functions which provide simplified access to files of dedicated formats.

The following file formats are supported:

- XML (eXtensible Markup Language)
- CSV (Comma-Separated Values)

Characteristics of the Library

The table indicates the characteristics of the library:

Characteristic	Value
Library title	FileFormatUtility
Company	Schneider Electric
Category	• **Util** • **Application/Util**
Component	**FileFormatUtility**
Default namespace	**FFU**
Language model attribute	qualified-access-only *(see SoMachine, Functions and Libraries User Guide)*
Forward compatible library	Yes (FCL *(see SoMachine, Functions and Libraries User Guide)*)

NOTE: For this library, qualified-access-only is set. This means, that the POUs, data structures, enumerations, and constants have to be accessed using the namespace of the library. The default namespace of the library is **FFU**.

Overview of the POUs

Function block	Use
`FB_XmlRead` *(see page 58)*	Reads an XML file.
`FB_XmlWrite` *(see page 66)*	Creates an XML file.
`FC_XmlGetElementValue` *(see page 74)*	Reads the value of an element from the buffer (`XmlItems`).
`FC_XmlSetElementValue` *(see page 78)*	Modifies the value of an element in the buffer (`XmlItems`).
`FB_CsvRead` *(see page 84)*	Reads values from a CSV file.
`FB_CsvWrite` *(see page 92)*	Writes values to an existing or to a new CSV file.

Overview of the Structures in the Module-Specific Interface

Structure	Use
`ST_XmlItem` *(see page 34)*	Describes an element or an attribute that is read from or written to an XML file.
`ST_XmlUserDefinedHeader` *(see page 36)*	Allows you to define a header that is written at the beginning of the newly created XML file.
`ST_CsvTable` *(see page 37)*	Passes the buffer provided by the application to the corresponding function block.
`ST_CsvFileInformation` *(see page 39)*	Provides information about the CSV file that has been most recently processed by the function block `FB_CsvRead`.
`ST_CsvWarnValueTruncated` *(see page 40)*	In case a value has been truncated, it provides information about the first value that has been truncated during the execution of the function block `FB_CsvRead`.
`ST_CsvReadParameter` *(see page 41)*	Specifies the content to be read from the CSV file using the function block `FB_CsvRead`.
`ST_CsvWriteParameter` *(see page 42)*	Provides the parameter for the write operation that is executed by the function block `FB_CsvWrite`.

Overview of the Enumerations

Enumeration	Use
`ET_XmlItemType` *(see page 26)*	Specifies the type of an XML item.
`ET_CsvReadMode` *(see page 27)*	Parameter to specify the content to be read from the CSV file using the function block `FB_CsvRead`.
`ET_ModeFileOpen` *(see page 28)*	Parameter to specify the mode for opening a file.
`ET_Result` *(see page 29)*	Contains the possible values that indicate the result of operations executed by the POUs of this library.

Chapter 2
Common Inputs and Outputs

Behavior of Function Blocks with the Input `i_xExecute`

General Information

A rising edge of the input `i_xExecute` starts the execution of the function block. The function block continues execution and the output `q_xBusy` is set to TRUE. A rising edge at the input `i_xExecute` is ignored while the function block is being executed.

Once the execution is finished, the outputs `q_xDone` or `q_xError` remain TRUE until the input `i_xExecute` is set to FALSE. If the input is reset before the execution is finished the outputs `q_xDone` or `q_xError` are set to TRUE for one cycle.

Example

Part II
Data Unit Types

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
3	Enumerations	25
4	Structures	33
5	Aliases	43

Chapter 3
Enumerations

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
ET_XmlItemType	26
ET_CsvReadMode	27
ET_ModeFileOpen	28
ET_Result	29

ET_XmlItemType

Overview

Type:	Enumeration
Available as of:	V1.0.8.0

Description

The enumeration `ET_XmlItemType` specifies the type of an XML item.

Enumeration Elements

Name	Value (INT)	Description
`NotSet`	0	No type has been specified for the XML item.
`Element`	1	The item is of type element.
`Attribute`	2	The item is of type attribute.

Used By

- `ST_XmlItem`

ET_CsvReadMode

Overview

Type:	Enumeration
Available as of:	V1.0.8.0

Description

The enumeration ET_CsvReadMode defines the content to be read from the CSV file using the function block FB_CsvRead.

Enumeration Elements

Name	Value (INT)	Description
AllValues	0	All values are read from the CSV file.
OneRow	1	One row (record) is read from the CSV file.
OneColumn	2	One column is read from the CSV file.
OneValue	3	One single value is read from the CSV file.
GetFileInformation	4	Only the information about the content of the file is retrieved. No value is read.

Used By

- FB_CsvRead

ET_ModeFileOpen

Overview

Type:	Enumeration
Available as of:	V1.0.8.0

Description

The enumeration ET_ModeFileOpen specifies the mode for opening a file.

Enumeration Elements

Name	Value (INT)	Description
NotSet	0	No mode has been selected.
Append	1	An existing file is opened and the specified content is appended. If the file does not exist, the function block indicates an error.
AppendPlus	2	As for Append, an existing file is opened and the specified content is appended but with AppendPlus, if the file does not exist, a new file is created.
Create	3	A file is created and the specified content is written. If the file already exists, the function block indicates an error.
CreatePlus	4	As for Create, a file is created and the specified content is written but with CreatePlus, if the file already exists, the content is overwritten.

Used By

- FB_CsvWrite

ET_Result

Overview

Type:	Enumeration
Available as of:	V1.0.8.0

Description

The enumeration `ET_Result` contains the possible values that indicate the result of operations executed by the POUs of this library.

Enumeration Elements

Name	Value (UDINT)	Description
`Idle`	0	The function block is ready to be executed.
Status information, indicated by `q_etResult` if `q_xDone` = TRUE		
`OK`	1	The function block has been executed successfully.
Status information, indicated by `q_etResult` if `q_xError` = FALSE and `q_xBusy` = TRUE		
`CheckingInputs`	10	The inputs are being verified.
`Initializing`	15	An internal resource is being initialized.
`OpeningFile`	17	The file is being opened.
`AnalyzingFile`	20	The file is being analyzed.
`ReadingFile`	24	The file is being read.
`WritingFile`	25	The file is being written.
`ClosingFile`	30	The file is being closed.
`GetFileSize`	35	The size of the file is being retrieved.
Error information, indicated by `q_etResult` if `q_xError` = TRUE		
`FilePathInvalid`	100	The specified file path has an invalid syntax.
`XPathExpressionInvalid`	102	The specified XPath (XML Path language) expression has an invalid syntax or is not supported.
`FileInvalid`	103	The content of the specified file to be read is not supported.
`Timeout`	104	A timeout has expired during execution.
`FileOpenFailed`	110	An error has been detected while opening the file.
`FileWriteFailed`	111	An error has been detected while writing the file.
`FileCloseFailed`	112	An error has been detected while closing the file.

Name	Value (UDINT)	Description
FileAlreadyExists	113	The specified file for the write operation already exists. It is not allowed to be overwritten.
FileNotExists	114	The specified file for the write operation does not exist. It is not possible to append data.
GetFileSizeFailed	115	An error has been detected while retrieving the size of the file.
NumOfParentsExceeded	120	The nesting depth of the XML structure is greater than specified by the parameter Gc_udiXmlMaxNumOfParents in the global parameter list *(see page 49)*.
BufferFull	140	The size of the buffer for storing the read items is insufficient.
AdditionalContentInvalid	150	The pointer pbyAdditionalContent *(see page 36)* is 0 while the value of udiNumBytesToWrite is > 0.
ElementNotFound	160	No element has been found that matches the XPath expression.
XmlStructureInconsistent	165	The parent-child relations between the elements in the array XmlItems are not consistent.
XmlItemTypeInvalid	167	The type of an item in the array XmlItems is invalid.
ParsingFailed	169	While parsing the file, an internal error has been detected.
FileInconsistent	170	The structure of the parsed XML file is inconsistent. It contains at least one tag which is not properly closed.
TableReadValuesInvalid	171	The dimensions provided for the table where the read values shall be stored are not valid. Refer to ST_CsvTable *(see page 37)*.
TableWriteValuesInvalid	172	The dimensions provided for the table which contains the values that shall be written are not valid. Refer to ST_CsvTable *(see page 37)*.
TableInvalid	175	The pointer pbyAdditionalContent *(see page 36)* to the buffer provided by the application must not be 0.
ReadParameterInvalid	181	The parameters to control the read operation are not valid. Refer to ST_CsvReadParameter *(see page 41)*.
WriteParameterInvalid	182	The parameters to control write operation are not valid. Refer to ST_CsvWriteParameter *(see page 42)*.
FileReadFailed	190	While reading the file, an internal error has been detected.
TableTooSmall	200	The table does not provide enough cells to store the number of values which shall be read from the file.
ValueNotFound	210	The value specified by the row and column does not exist in the CSV file.

Name	Value (UDINT)	Description
`FilePathTooLong`	215	The specified file path without file extension characters is outside the valid range. It is not possible to add the default file extension. The file path without file extension is limited to 255 characters minus length of the default file extension.
`FilenameTooLong`	216	The specified file name including the file extension exceeds the allowed length of 126 characters.
`FilenameInvalid`	217	The specified file name is invalid.
`FirstItemNoElement`	220	The first item in the array `XmlItems` *(see page 34)* is not of type element.
`FirstItemInvalidParentIndex`	221	The value for `diParentIndex` of the first item in the array `XmlItems` *(see page 34)* is not -1.
`XpathRootElementDoesNotMatch`	222	The name of the first element in the array `XmlItems` *(see page 34)* does not match the root element specified in the XPath expression.
`XpathExpressionNotSupported`	230	The specified XPath expression is not supported by this function.
`UnexpectedProgramBehaviour`	999	An internal error has been detected. Contact your Schneider Electric support.

Used By

- `FB_XmlRead`
- `FB_XmlWrite`
- `FB_CsvRead`
- `FB_CsvWrite`

Chapter 4
Structures

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
ST_XmlItem	34
ST_XmlUserDefinedHeader	36
ST_CsvTable	37
ST_CsvFileInformation	39
ST_CsvWarnValueTruncated	40
ST_CsvReadParameter	41
ST_CsvWriteParameter	42

ST_XmlItem

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure `ST_XmlItem` is used to describe an element or an attribute that is read from or written to an XML file.

Structure Elements

Name	Data type	Description
`diParentIndex`	DINT	Indicates the index of the array where the parent element of the XML item is located (refer to *Example for Hierarchical Relations Indicated by uiParentIndex* *(see page 35)*). If the value is -1, the element is a root element.
`sName`	STRING[`GPL.Gc_uiXmlLengthOfString`]	Name of the element or attribute.
`sValue`	STRING[`GPL.Gc_uiXmlLengthOfString`]	Value of the element or attribute.
`etType`	`ET_XmlItemType` *(see page 26)*	Indicates the type of the XML item.
`uiNumOfAttributes`	ULINT	This value depends on the type of the XML item: ● If the item is of type element, then the value indicates the number of associated attributes. ● If the item is of type attribute, then the value indicates the sequential number.

Used By

- `FB_XmlRead`
- `FB_XmlWrite`

Example for Hierarchical Relations Indicated by `uiParentIndex`

The example illustrates the correlation between the parameter `uiParentIndex` in the buffer of type `XmlItems` provided by the application and the hierarchical structure in the XML document.

XML document

```
<?xml version="1.0" encoding="ASCII"?>
<AAA>
   <BBB />
   <CCC />
     <DDD>
       <EEE />
     </DDD>
</AAA>
```

Buffer of type `XmlItems` provided by the application

Array Index	uiParentIndex	sName	..
0	-1	AAA	
1	0	BBB	
2	0	CCC	
3	2	DDD	
4	3	EEE	

Element	Parent index	Explanation
AAA	–1	AAA is the root element. It has no parent.
BBB	0	AAA is the parent element. It is stored in array index 0.
CCC	0	
DDD	2	CCC is the parent element. It is stored in array index 2.
EEE	3	DDD is the parent element. It is stored in array index 3.

ST_XmlUserDefinedHeader

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure `ST_XmlUserDefinedHeader` allows you to define a header that is written to the XML file.

Structure Elements

Name	Data type	Description
`sUserComment`	STRING[255]	Enter a text that is converted to an XML comment. It is written at the beginning of the XML file.
`pbyAdditionalContent`	POINTER TO BYTE	Pointer to the buffer provided by the application. It contains the content that is to be written at the beginning of the XML file in addition to the comment.
`udiNumBytesToWrite`	UDINT	Specifies the size in bytes for the additional content.

Used By

- `FB_XmlWrite`

ST_CsvTable

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure `ST_CsvTable` is used to pass the buffer provided by the application to the corresponding function block.

Structure Elements

Name	Data type	Description
`pbyTable`	POINTER TO BYTE	Pointer to the buffer (two-dimensional ARRAY of type STRING) provided by the application.
`uiNumOfRows`	UINT	Specifies the number of rows (records) in the table.
`uiNumOfColumns`	UINT	Specifies the number of values per row (record) in the table.
`udiSizeOfTable`	UDINT	Specifies the total size in bytes of the table.

NOTE: To prevent access violation eventually caused by pointer access to the memory, use the arithmetic operator `SIZEOF` in conjunction with the targeted buffer to determine the value for `udiSizeOfTable`.

Example

The example shows how to assign the values to this structure:

```
PROGRAM POU
VAR
    g_asCsvTable:ARRAY[0..c_uiNumOfRows-1,0..c_uiNumOfColumns-1] OF
STRING(c_uiLengthOfValue);
        stCsvTable: FFU.ST_CsvTable;
END_VAR
VAR CONSTANT
        c_uiNumOfRows :UINT:= 100;
        c_uiNumOfColumns :UINT:= 10;
        c_uiLengthOfValue :UINT:= 40;
END_VAR
stCsvTable.pbyTable := ADR(g_asCsvTable);
stCsvTable.uiNumOfRows := c_uiNumOfRows;
stCsvTable.uiNumOfColumns := c_uiNumOfColumns;
stCsvTable.udiSizeOfTable := SIZEOF(g_asCsvTable);
```

Used By

- FB_XmlRead
- FB_XmlWrite

ST_CsvFileInformation

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure `ST_CsvFileInformation` provides information about the CSV file that has recently been processed by the function block `FB_CsvRead` *(see page 84)*.

Structure Elements

Name	Data type	Description
`udiFileSize`	UDINT	Indicates the size of the CSV file in bytes.
`udiNumOfValues`	UDINT	Indicates the number of values in the CSV file.
`udiNumOfRows`	UDINT	Indicates the number of rows (records) in the CSV file.
`udiNumOfColumns`	UDINT	Indicates the number of columns (values per record) in the CSV file.
`xTableInconsistent`	BOOL	Indicates TRUE if a different number of columns has been detected for at least for two rows.

Used By

- `FB_CsvRead`

ST_CsvWarnValueTruncated

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure `ST_CsvWarnValueTruncated` provides information about the first value that has been truncated during the execution of the function block `FB_CsvRead`.

Structure Elements

Name	Data type	Description
`xValueTruncated`	BOOL	Indicates TRUE if at least one value has been truncated.
`udiRow`	UDINT	Indicates the number of the row in which the first value has been truncated.
`udiColumn`	UDINT	Indicates the number of the column in which the first value has been truncated.

Used By

- `FB_CsvRead`

ST_CsvReadParameter

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure ST_CsvReadParameter is used to specify the content to be read from the CSV file using the function block FB_CsvRead.

Structure Elements

Name	Data type	Description
sDelimiter	STRING[5]	Specifies the character code for the separator that is used as delimiter between two values.
etReadMode	ET_ReadMode	Specifies the content to be read from the CSV file.
udiNumOfRow	UDINT	Specifies the number of the row to be read. This value is relevant for: ● ET_ReadMode.OneRow ● ET_ReadMode.OneValue Refer to ET_CSVReadMode *(see page 27)*.
udiNumOfColumn	UDINT	Specifies the number of the column to be read. This value is relevant for: ● ET_ReadMode.OneColumn ● ET_ReadMode.OneValue Refer to ET_CSVReadMode *(see page 27)*.

Used By

- FB_CsvRead

ST_CsvWriteParameter

Overview

Type:	Structure
Available as of:	V1.0.8.0
Inherits from:	-

Description

The structure `ST_CsvWriteParameter` is used to configure the write operation that is executed by the function block `FB_CsvWrite`.

Structure Elements

Name	Data type	Description
`sDelimiter`	STRING[5]	Specifies the character code for the separator that is inserted between two values.
`etModeFileOpen`	ET_ModeFileOpen *(see page 28)*	Specifies the write mode for opening or creating the CSV file.
`udiNumOfRow`	UDINT	Specifies the number of rows that shall be written. If this value is 0, the rows specified by the parameter `i_stBufferWriteValues.uiNumOfRows` will be written to the file.
`udiNumOfColumn`	UDINT	Specifies the number of columns per row that shall be written. If this value is 0, the columns specified by the parameter `i_stBufferWriteValues.uiNumOfColumns` will be written to the file.

Used By

- `FB_CsvWrite`

Chapter 5
Aliases

Aliases

Overview

Type:	ALIAS (DUT)
Available as of:	V1.0.8.0
Inherits from:	–

Description

An alias represents a complex data type which is used in this library.

XmlItems

Name	Data type	Description
XmlItems	ARRAY[0..GPL.Gc_udiXmlMaxNbOfElements] of (ST_XmlElement)	The device unit type XmlItems is used to store the elements that are read from or written to an XML file. Use it to declare the buffer provided by the application. It is linked to the function blocks FB_XmlRed and FB_XmlWrite.

Part III
Global Variables

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
6	Global Constants List	47
7	Global Parameter List	49

Chapter 6
Global Constants List

Global Constants List (GCL)

Overview

Type:	Global constants
Available as of:	V1.0.8.0

Description

The global constants list contains the global constants of the FileFormatUtility library.

Global Constants

Variable	Data type	Value	Description
`Gc_sLibraryVersion`	STRING[80]	Vx.x.x.0[1]	Library version
[1] This value varies to indicate the version of the library.			

Chapter 7
Global Parameter List

GPL

Overview

Type:	Global parameters
Available as of:	V1.0.8.0

Description

The global parameter list (GPL) contains global constants which are used by certain components of this library. The parameters can be edited individually for each application where the library is used. The modification must be done within the **Library Manager** of the project where the library is referenced.

Global Parameters

Variable	Data type	Default value	Range	Description
`Gc_udiXmlMaxNumOfItems`	UDINT	1000	1... 2147483647	Determines the size of the buffer which contains the elements with their attributes which were read from or will be written to an XML file. The value indicates the sum of the elements and attributes which can be stored in the buffer.
`Gc_uiXmlLengthOfString`	UINT	40	1...254	Determines the maximum length of the elements of type STRING in the `ST_XmlItem` structure *(see page 34)*.
`Gc_udiXmlMaxNumOfParents`	UDINT	20	1...10000	Determines the maximum nesting depth of the XML structure. The variable is used internally to determine the parent index.
`Gc_uiXmlWriteProcessingBlockSize`	UINT	5000	500...65535	Determines the size of the temporary buffer (in bytes) that is used for processing the content of the XML file during write operation.
`Gc_uiCsvReadProcessingBlockSize`	UINT	1000	100...65535	Determines the size of the temporary buffer (in bytes) that is used for processing the values of the CSV file during read operation.
`Gc_uiCsvWriteProcessingBlockSize`	UINT	1000	100...65535	Determines the size of the temporary buffer (in bytes) that is used for processing the values of the CSV file during write operation.

Part IV
Global Functions

Chapter 8
Global Functions

FC_EtResultToString

Overview

Type:	Function
Available as of:	V1.0.8.0
Inherits from:	–
Implements:	–

Task

Convert an enumeration element of type ET_Result to a variable of type STRING.

Functional Description

Using the function `FC_EtResultToString`, you can convert an enumeration element of type ET_Result to a variable of type STRING.

Interface

Input	Data type	Description
`i_etResult`	ET_Result	Enumeration with the result.

Return Value

Data type	Description
STRING(80)	The ET_Result converted to text.

Part V
XML Program Organization Units (POU)

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
9	XML Function Blocks	57
10	XML Functions	73

Chapter 9
XML Function Blocks

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
9.1	FB_XmlRead	58
9.2	FB_XmlWrite	66

Section 9.1
FB_XmlRead

What Is in This Section?

This section contains the following topics:

Topic	Page
FB_XmlRead Functional Description	59
FB_XmlRead Considerations	62
FB_XmlRead Troubleshooting	63
FB_XmlRead Example	64

FB_XmlRead Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-

Functional Description

The function block `FB_XmlRead` is used to read (parse) an XML file that is located on the file system of the controller, or on the extended memory (for example, an SD memory card). For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller.

The content of the XML file, XML elements together with their attributes and values, is stored in an array of type `XmlItems` in the application memory of the controller. You have to declare this array and assign it to the associated input `i_refXmlItems` on the function block. At the beginning of each read operation, the content of this array is erased.

The number of items (sum of elements and attributes) which can be stored in the array is specified by the parameter `Gc_udiXmlMaxNumOfItems` in the GPL *(see page 50)*.

The array contains the fields of type STRING to store the names and the values of the elements and attributes. You can specify the length of these STRINGs by the global parameter `Gc_uiXml-LengthOfString`. If a value to be read in the file exceeds the specified length, the original value is truncated. If at least one value has been truncated, this is indicated by the output `q_xWarnValueTruncated`.

NOTE: The output `q_xWarnValueTruncated` is valid only if the output `q_xDone` is TRUE.

The hierarchical structure of the elements from the XML file is indicated by the parameter `uiParentIndex` for each item in the array of type `XmlItems`. For further information, refer to `uiParentIndex` *Example for Hierarchical Relations Indicated by uiParentIndex (see page 35)*.

Interface

Input	Data type	Description
`i_xExecute`	BOOL	The function block executes the read operation with the specified XML file upon a rising edge of this input. Also refer to the chapter *Behavior of Function Blocks with the Input i_xExecute (see page 21)*.
`i_sFilePath`	STRING[255]	File path to the XML file that shall be read. If a file name is specified without file extension, the function block adds the extension .xml.
`i_sXPathItemToRead`	STRING[255]	XPath expression to address the elements which shall be read from the XML file. Default value: `'//*'`
`i_xReadElementsOnly`	BOOL	If this input is TRUE, the element names and their values are read and stored to the application buffer. If this input is FALSE, the attributes together with their values are also read and stored to the application buffer.
`i_refXmlItems`	REFERENCE TO `XmlItems`	Buffer provided by the application to store the elements read from the specified XML file. The buffer is erased with each execution of the function block.

Output	Data type	Description
`q_xDone`	BOOL	If this output is set to TRUE, the execution has been completed successfully.
`q_xBusy`	BOOL	If this output is set to TRUE, the function block execution is in progress.
`q_xError`	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to `q_etResult` and `q_etResultMsg`.
`q_etResult`	`ET_Result`	Provides diagnostic and status information as a numeric value. If `q_xBusy` = TRUE, the value indicates the status. If `q_xDone` or `q_xError` = TRUE, the value indicates the result.
`q_sResultMsg`	STRING[80]	Provides additional diagnostic and status information as a text message.
`q_udiNumOfItemsRead`	UDINT	Indicates the total number of elements and attributes read from the XML file.
`q_xWarnValueTruncated`	BOOL	If this output is set to TRUE, at least one value has been truncated. **NOTE:** The output is updated along with `q_xDone`.

For more information about the signal behavior of the basic inputs and outputs, refer to the chapter *Behavior of Function Blocks with the Input i_xExecute* *(see page 21)*.

XPath Expressions Defining the Content to Be Read

To be able to read a single element or a group of elements from the XML file, use the syntax of the XPath (XML Path) language. The content to be read is specified by the input `i_XpathItemToRead`

NOTE: The function block `FB_XmlRead` supports a subset of the features provided with XPath expressions.

The table lists the supported XPath expressions:

XPath expression	Description
`//*`	Selects all elements in the document.
`/`	Indicates an absolute path to an element.
`/…/child::*`	Selects all child elements of the node.
`/…/descendant::*`	Selects all descendant elements of the node.
`/…/<elementname>`	Selects all elements with the specified name of the node.
`/…/<elementname>[<n>]`	Selects the nth element with the specified name of the node.
`/…/<elementname>[@<attribute>]`	Selects all elements with the specified name and the specified attribute of the node.
`/…/<elementname>[@<attribute>=<value>]`	Selects all elements with the specified name and the specified attribute and value of the node.

NOTE: The predicates, that are the expressions within square brackets `[]`, can be followed by a slash `/` together with an element name to address the next child element.

Example: `/…/<elementname>[<n>]/<elementname>`

FB_XmlRead Considerations

Considerations

Consider the following constraints for reading an XML file:

- Only XML files with ASCII encoding are supported.
- Blank spaces are interpreted as values, tabs are not interpreted as values.
- Line breaks in values are not supported. If a value includes a line break, the characters before the line break are considered during parsing.
- Only the names of elements and their attributes together with their values are read and then stored in the buffer provided by the application. This has the effect that XML objects other than elements and attributes, such as comments, and DOCTYPE declarations are not detected by the XML parser.
- CDATA objects are not supported. The content of CDATA objects is interpreted as a value of the open element.
- The values read from the file are stored as STRING values in the application. This applies even to numeric values. Before they can be processed, the values need to be converted to the appropriate datatype. To achieve this, it is a good practice to use the `STRING_TO_` conversion functions. The conversion functions require a specific syntax of the STRING values depending on the target datatype. Consider these requirements when creating the files to simplify the processing of read values.
- Parsing the XML file is a time-consuming process. It is performed in parallel to the task that calls the function block. Thus, the time required for parsing is part of the task execution time. It results, for a single cycle, in an increased task execution time while the `FB_XmlRead` function block is executed. Depending on the size of the file and the controller, the increased task cycle can take up to a few seconds. To help prevent other processes from being blocked by parsing an XML file, create a separate task with low priority (>24) for this function. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.

FB_XmlRead Troubleshooting

Troubleshooting

This table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified file or the directory is not available. • The given file path has invalid syntax.	• Verify that the file exists under the specified directory. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator, while another controller supports '/' instead.
Execution ends with a detected error and result indicates `FileInvalid`.	• The file contains characters outside the ASCII range. • During parsing, an unexpected closing tag has been detected.	• Verify that the file contains only ASCII characters. • Verify that the XML file is valid: Make sure that each open tag is closed and that the elements are properly nested. **NOTE:** Element names are case-sensitive.
Execution ends with a detected error and result indicates `XPathExpressionInvalid`.	• The entered XPath expression is invalid or not supported.	• Verify the correct syntax of the XPath expression. • Verify that the expression is supported by the function block *(see page 61)*.
Execution ends with a detected error and result indicates `NumOfParentsExceeded`.	• The nesting depth of elements in the XML file to read is larger than specified.	• Increase the value for the parameter `Gc_udiXmlMaxNumOfParents` in the GPL *(see page 50)*. • Avoid the use of files with deeply nested XML structures.
Execution ends with a detected error and result indicates `BufferFull`.	• The number of items in the XML file to read is greater than the size of the buffer provided by the application.	• Increase the buffer size by the parameter `Gc_XmlMaxNumOfItems` in the GPL *(see page 50)*. • Split the read operation in several blocks. Use an appropriate XPath expression to limit the number of elements to read per block.

Issue	Cause	Solution
Execution ends with a detected error and result indicates `ElementNotFound`.	• The element specified in the XPath expression does not exist in the file. • The path declaration in the XPath expression does not match the structure in the XML file. • There is a typing error in the XPath expression.	• Verify that the specified element is typed correctly. • Verify that the path declaration meets the expected structure in the XML file. • Verify that the correct XML file is specified with the parameter `i_sFilePath` *(see page 60)*.
Execution ends with a detected error and result indicates `FileInconsistent`.	• The end of file has been reached but at least one tag has not been closed.	• Verify that the XML file is valid: Make sure that each open tag is closed and that the elements are properly nested.

FB_XmlRead Example

Overview

The following example shows how the elements read from an XML file are stored in the buffer provided by the application for further processing.

Example XML File

```
<?xml version="1.0" encoding="ASCII"?>
<!--This is the user comment.-->
<!DOCTYPE AAA SYSTEM "example.dtd">

<AAA>
  <BBB>1st bbb</BBB>
  <BBB>2nd bbb</BBB>
    <CCC id="1">
      <DDD id="1" activate="TRUE">ddd</DDD>
    </CCC>
</AAA>
```

NOTE: The comment and the doctype declaration (DTD) in lines 2 and 3 are not read. This information will not be available in the application.

Example Program

```
PROGRAM SR_Example
VAR
  fbRead :FFU.FB_XmlRead;
  astXmlItems :FFU.XmlItems;
  xCmdRead :BOOL;
END_VAR
fbRead
       i_xExecute := xCmdRead,
       i_sFilePath := '/sd0/Example.xml',
       i_sXPathItemsToRead := ,
       i_xReadElementsOnly := ,
       i_refXmlItems := astXmlItems,
       q_xDone => ,
       q_xBusy => ,
       q_xError => ,
       etResult => ,
       q_sResultMsg => ,
       q_udiNumOfItems => ,
       q_xWarnValueTruncated => ) ;
```

Buffer

The buffer provided by the application of type `XmlItems` contains the elements and attributes read from the XML file.

For this example, the value of the output `q_udiNumOfItems` is `8`.

Array Index	uiParentIndex	sName	sValue	etType	uiNumOfAttributes
0	-1	AAA	–	1	0
1	0	BBB	1st bbb	1	0
2	0	BBB	2nd bbb	1	0
3	2	CCC	–	1	1
4	3	id	1	2	1
5	3	DDD	ddd	1	2
6	5	id	1	2	1
7	5	activate	TRUE	2	2

Section 9.2
FB_XmlWrite

What Is in This Section?

This section contains the following topics:

Topic	Page
FB_XmlWrite Functional Description	67
FB_XmlWrite Considerations	69
FB_XmlWrite Troubleshooting	70
FB_XmlWrite Example	71

FB_XmlWrite Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-

Functional Description

The function block `FB_XmlWrite` is used to create or overwrite an XML file on the file system of the controller, or on the extended memory (for example, an SD memory card). For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller.

After the file has been created, the elements provided in the array of type `XmlItems` in the application memory of the controller are written into it.

The character code `LF (0A hex)` is used as line break in the created file.

The prolog `<?xml version="1.0" encoding="ASCII"?>` is inserted as first line of each file that is created by the function block.

After the prolog, the XML elements that are provided in the array of type `XmlItems`, including their attributes and their values, are written to the file. The structure or the nesting of the elements is specified by the parameter `uiParentIndex` which is part of the structure `ST_XmlItem`. For further information, refer to `uiParentIndex` *Example for Hierarchical Relations Indicated by uiParentIndex* *(see page 35)*.

The function block FB_XmlWrite provides the input i_sRootElement to specify a root element. This is useful in case the array providing the elements contains no root element or more than one root element:

- The use case without root element is allowed if the values of the uiParentIndex parameters in the array are all 0.
- The use case with more than one root element can occur if a group of child elements has been read from a file before the right operation and the modified data is to be written to a new file.

Apart from these two exceptional cases, the defined structure of the elements must be consistent. Otherwise the function block cancels writing the file and the file is discarded.

For items of type attribute, the parameter uiParentIndex does not have an effect. Attributes are assigned to the next upper item of type element in the array.

You can define additional content using the structure i_stUserDefinedHeader. The content is written between the prolog and the first element to the XML file. This additional content could be, for example, a comment (XML syntax) and/or a DOCTYPE declaration (DTD).

The input i_xOverwriteFile allows you to define whether an existing file will be overwritten. If the input is FALSE and the specified file already exists, the execution of the function block is canceled and an error is indicated.

Interface

Input	Data type	Description
i_xExecute	BOOL	The function block executes the read operation with the specified XML file upon a rising edge of this input. Also refer to the chapter *Behavior of Function Blocks with the Input i_xExecute* *(see page 21)*.
i_sFilePath	STRING[255]	File path to the XML file. If a file name is specified without file extension, the function block adds the extension .xml.
i_xOverwriteFile	BOOL	Specifies whether an existing file is to be overwritten. Set this input to TRUE to allow the replacement of an existing file.
i_sRootElement	WSTRING[GPL.Gc_uiXmlLengthOfWString]	Root element which is created if the array of structure XmlElements contains more than one root element.
i_stUserDefinedHeader	ST_XmlUserDefinedHeaderAscii	The structure contains user-defined content that is to be written at the beginning of the newly created XML file.
i_timTimeout	TIME	After this time has elapsed, the execution is canceled. If the value is T#0s, the default value T#2s is applied.
i_refXmlItems	REFERENCE TO XmlItems	Buffer provided by the application that contains the content to be written to the XML file.

Output	Data type	Description
q_xDone	BOOL	If this output is set to TRUE, the execution has been completed successfully.
q_xBusy	BOOL	If this output is set to TRUE, the function block execution is in progress.
q_xError	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to q_etResult and q_etResultMsg.
q_etResult	ET_Result	Provides diagnostic and status information as a numeric value. If q_xBusy = TRUE, the value indicates the status. If q_xDone or q_xError = TRUE, the value indicates the result.
q_sResultMsg	STRING[80]	Provides additional diagnostic and status information as a text message.

FB_XmlWrite Considerations

Considerations

Consider the following constraints for writing an XML file:

- File operations are time-consuming processes. To help prevent influencing time-critical control functions in your application, create a separate task with lower priority for these processes. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information about task management, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- The timeout parameter i_timTimeout is used to monitor the file operation. If the value specified for the timeout is exceeded during execution of the function block, the write operation is canceled and the function block indicates an error. To select an appropriate value for the timeout parameter, consider that the file operation takes several task cycles. Thus, calculate the product of the number of required task cycles and the task interval to specify the minimum value for the timeout parameter.
- The number of cycles depends, besides the amount of data to be written, on the size of the processing block which can be specified by the parameter Gc_uiXmlWriteProccessing-BlockSize in the GPL *(see page 49)*. The creation of the file is split into several write operations to reduce the load for the single task cycle. During each write operation, one data block is processed and written to the file. The larger the processing block size the fewer cycles are required for creating and writing the file. However, a larger block size increases the execution time of each write operation.

FB_XmlWrite Troubleshooting

Troubleshooting

This table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified directory is not available. • The given file path has an invalid syntax.	• Verify that the directory exists. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator, while another controller supports '/' instead.
Execution ends with a detected error and result indicates `FileAlreadyExists`.	• The specified file already exists and the input `i_xOverwriteFile` is FALSE.	• Specify another file name. • If the existing file can be overwritten, set the input `i_xOverwriteFile` to TRUE.
Execution ends with a detected error and result indicates `XmlStructureInconsistent`.	• At least one value for the parameter `uiParentIndex` in the buffer that contains the XML elements *(see page 35)* prevents the creation of a valid XML file. • The `uiParentIndex` of the first element is <> -1.	• Verify that the values defined for the parameter `uiParentIndex` *(see page 34)* are consistent. **NOTE:** Refer to the output `q_sResultMsg` for further information on the invalid parameter.
Execution ends with a detected error and result indicates `Timeout`.	• The specified value for the timeout is too low.	• Increase the timeout parameter according to the number of cycles needed to create the file. Consider also the task interval. • Increase the processing block size per write operation to reduce the number of function block calls which are needed to create the file.

FB_XmlWrite Example

Overview

The following example shows how the elements and attributes stored in the buffer provided by the application are written to the XML file. In addition to the elements and attributes, an example for the additional content is provided.

Buffer

The buffer provided by the application of type `XmlItems` contains elements to be written to the XML file.

Array Index	uiParentIndex	sName	sValue	etType	uiNumOfAttributes
0	-1	AAA	–	1	0
1	0	BBB	1st bbb	1	0
2	0	BBB	2nd bbb	1	0
3	2	CCC	–	1	1
4	–	id	1	2	1
5	3	DDD	ddd	1	2
6	–	id	1	2	1
7	–	activate	TRUE	2	2

Example Program

```
PROGRAM SR_Example
VAR
  fbWrite :FFU.FB_XmlWrite;
  astXmlItems :FFU.XmlItems;
  xCmdWrite :BOOL;
  sComment :STRING(255) := 'This is the user comment.';
  sExternalDTD :STRING := '<!DOCTYPE AAA SYSTEM "example.dtd">';
  stHeader :FFU.ST_XmlUserDefinedHeader;
END_VAR

stHeader.sUserComment := sComment;
stHeader.pbyAdditionalContent := ADR(sExternalDTD);
stHeader.sUserComment := Standard.LEN(sExternalDTD);

fbWrite(
        i_xExecute := xCmdWrite,
        i_sFilePath := '/sd0/Example.xml',
        i_xOverwriteFile := ,
        i_sRootElement := ,
        i_refXmlItems := astXmlItems,
        i_stUserDefinedHeader := stHeader,
        i_timTimeout := ,
        q_xDone => ,
        q_xBusy => ,
        q_xError => ,
        q_etResult => ,
        q_sResultMsg => );
```

Example XML File

```
<?xml version="1.0" encoding="ASCII"?>
<!--This is the user comment.-->
<!DOCTYPE AAA SYSTEM "example.dtd">

<AAA>
  <BBB>1st bbb</BBB>
  <BBB>2nd bbb</BBB>
    <CCC id="1">
      <DDD id="1" activate="TRUE">ddd</DDD>
    </CCC>
</AAA>
```

Chapter 10
XML Functions

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
10.1	FC_XmlGetElementValue	74
10.2	FC_XmlSetElementValue	78

Section 10.1
FC_XmlGetElementValue

What Is in This Section?

This section contains the following topics:

Topic	Page
FC_XmlGetElementValue Functional Description	75
FC_XmlGetElementValue Considerations	77

FC_XmlGetElementValue Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-

Functional Description

The function `FC_XmlGetElementValue` is used to read the value of the specified XML element from the buffer of type `XmlItems`.

The return value is TRUE if the function has been executed successfully. If the return value is FALSE, evaluate the output `q_etResult`.

Interface

Input	Data type	Description
`i_refXmlItems`	REFERENCE TO `XmlItems`	Buffer provided by the application which contains the elements and attributes read from or to be written to an XML file.
`i_sXpathToElement`	STRING[255]	XPath expression to specify the element that is to be read.

Output	Data type	Description
`q_etResult`	`ET_Result`	Provides diagnostic information as a numeric value.
`q_sElementValue`	STRING[`Gc_uiXmlLengthOfString`]	Provides the value of the specified element.

XPath Expressions Defining the Content to Be Read

Use the syntax of the XPath (XML Path) language to specify the element whose value is to be read. The parent-child relations between the elements in the buffer are determined by the parameter `diParentIndex` which indicates the array index where the parent element is stored.

NOTE: The function `FC_XmlGetElementValue` supports a subset of the features provided with XPath expressions.

The table lists the supported XPath expressions:

XPath expression	Description
`/…/<elementname>`	Selects the element with the specified name of the node.
`/…/<elementname>[<n>]`	Selects the nth element with the specified name of the node.
`/…/<elementname>[@<attribute>]`	Selects the element with the specified name of the node that has the specified attribute.
`/…/<elementname>[@<attribute>=<value>]`	Selects the element with the specified name of the node that has the specified attribute and value.

The predicates, that are the expressions within square brackets `[]`, can be followed by a slash `/` together with an element name to address the next child element.

Example: `/…/<elementname>[<n>]/<elementname>`

FC_XmlGetElementValue Considerations

Considerations for Reading XML Values

Consider the following constraints:

- The execution of the function or the search of the specified element in the buffer of type `XmlItems` is a time-consuming process and can take up to several milliseconds. To help prevent other processes from being blocked by this function, create a separate task with low priority (>24) for it. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- Depending on the size of the buffer `XmlItems` and the position where the element is stored, the execution time of this function can vary from one call to another. Consider this during the commissioning of your application.
- The function is designed to process exactly one element. If your buffer contains more than one element matching the XPath expression, it processes the element that is found first.

Section 10.2
FC_XmlSetElementValue

What Is in This Section?

This section contains the following topics:

Topic	Page
FC_XmlSetElementValue Functional Description	79
FC_XmlSetElementValue Considerations	80

FC_XmlSetElementValue Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-

Functional Description

The function `FC_XmlSetElementValue` is used to modify the value of the specified XML element from the buffer of type `XmlItems`.

The return value is TRUE if the function has been executed successfully. If the return value is FALSE, evaluate the output `q_etResult`.

Interface

Input	Data type	Description
`i_refXmlItems`	REFERENCE TO `XmlItems`	Buffer provided by the application which contains the elements and attributes read from or to be written to an XML file.
`i_sXpathToElement`	STRING[255]	XPath expression to specify the element that is to be read.
`i_sElementValue`	STRING`[Gc_uiXmlLengthOfString]`	The value to be set for the specified element.

Output	Data type	Description
`q_etResult`	`ET_Result`	Provides diagnostic information as a numeric value.

XPath Expressions Defining the Content to Be Set

Use the syntax of the XPath (XML Path) language to specify the element whose value is to be set. The parent-child relations between the elements in the buffer are determined by the parameter `diParentIndex` which indicates the array index where the parent element is stored.

NOTE: The function `FC_XmlSetElementValue` supports a subset of the features provided with XPath expressions.

The table lists the supported XPath expressions:

XPath expression	Description
`/…/<elementname>`	Selects the element with the specified name of the node.
`/…/<elementname>[<n>]`	Selects the nth element with the specified name of the node.
`/…/<elementname>[@<attribute>]`	Selects the element with the specified name of the node that has the specified attribute.
`/…/<elementname>[@<attribute>=<value>]`	Selects the element with the specified name of the node that has the specified attribute and value.

The predicates, that are the expressions within square brackets `[]`, can be followed by a slash `/` together with an element name to address the next child element.

Example: `/…/<elementname>[<n>]/<elementname>`

FC_XmlSetElementValue Considerations

Considerations for Setting XML Values

Consider the following constraints:

- The execution of the function or the search of the specified element in the buffer of type `XmlItems` is a time-consuming process and can take up to several milliseconds. To help prevent other processes from being blocked by this function, create a separate task with low priority (>24) for it. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- Depending on the size of the buffer `XmlItems` and the position where the element is stored, the execution time of this function can vary from one call to another. Consider this during the commissioning of your application.
- The function is designed to process exactly one element. If your buffer contains more than one element matching the XPath expression, it processes the element that is found first.

Part VI
CSV Program Organization Units (POU)

Chapter 11
CSV Function Blocks

What Is in This Chapter?

This chapter contains the following sections:

Section	Topic	Page
11.1	FB_CsvRead	84
11.2	FB_CsvWrite	92

Section 11.1
FB_CsvRead

What Is in This Section?

This section contains the following topics:

Topic	Page
FB_CsvRead Functional Description	85
FB_CsvRead Considerations	88
FB_CsvRead Troubleshooting	89
FB_CsvRead Example	90

FB_CsvRead Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-

Functional Description

The function block `FB_CsvRead` is used to read a CSV file located on the file system of the controller or on the extended memory (for example, an SD memory card). For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller.

The CSV file to read contains a number of values (columns) which are arranged in single records (rows). The values are separated by a specific delimiter. The records are separated by a line break.

Based on the specified character code for the delimiter, the function block identifies the individual values while reading the content of the file. The character code for the line break depends on the operating system where the file has been created. The function block supports the three most commonly used line break character codes (ASCII). It detects the line break character used while reading the content of the file.

The following line break characters (ASCII) are supported:

- `CRLF (0D0A hex)`: Used on operating systems such as Windows and MS-DOS.
- `LF (0A hex)`: Used on operating systems such as Unix, Linux, Mac OS X, and Android.
- `CR (0D hex)`: Used on operating systems such as Mac OS X version 9 or earlier.

The values read from the specified file are stored in the read buffer provided by the application in variables of type STRING. Declare the read buffer in the application as a two-dimensional ARRAY of type STRING. Use the input `i_stTableReadValues` to provide the dimensions of the array and the pointer to that array to the function block. For further information, refer to the structure `ST_CsvTable` *(see page 37)*.

The file to be read must contain only ASCII characters to help ensure the correct rendition of the content of the file in the application. Files can contain a byte order mark (BOM) at the beginning which indicates the encoding of the processed file. ASCII-encoded files do not contain a BOM. The function block verifies the specified file for specific BOMs.

In case the file contains one of the following BOMs, the execution of the function block is canceled and an error is indicated:

`FE FF hex`, `FF EF hex`, or `EF BB BF hex`

Use the input `i_stReadParameter` to determine the amount of data to be read. You can read the entire content of the file. You can also select a single record (row), a single column, or a single value to be read. Further, you can select to read only the file information which is provided by the output `q_stFileInformation`.

Interface

Input	Data type	Description
`i_xExecute`	BOOL	The function block executes the read operation with the specified CSV file upon a rising edge of this input. Also refer to the chapter *Behavior of Function Blocks with the Input i_xExecute* *(see page 21)*.
`i_sFilePath`	STRING[255]	File path to the CSV file. If a file name is specified without file extension, the function block adds the extension .csv.
`i_stReadParameter`	`ST_CsvReadParameter`	Specifies the content to be read from the file.
`i_timTimeout`	TIME	After this time has elapsed, the execution is canceled. If the value is T#0s, the default value T#2s is applied.
`i_stTableReadValues`	`ST_CsvTable`	Structure to pass the buffer provided by the application to the function block (refer to the `ST_CsvTable` structure *(see page 37)*).

Output	Data type	Description
`q_xDone`	BOOL	If this output is set to TRUE, the execution has been completed successfully.
`q_xBusy`	BOOL	If this output is set to TRUE, the function block execution is in progress.

Output	Data type	Description
q_xError	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to q_etResult and q_etResultMsg.
q_etResult	ET_Result	Provides diagnostic and status information as a numeric value. If q_xBusy = TRUE, the value indicates the status. If q_xDone or q_xError = TRUE, the value indicates the result.
q_sResultMsg	STRING[80]	Provides additional diagnostic and status information as a text message.
q_stFileInformation	ST_CsvFileInfo	The structure contains information about the last file that has been processed
q_stWarnValueTruncated	ST_CsvWarnValueTruncated	Provides information about the first value that has been truncated, if available. **NOTE:** The output is updated along with q_xDone.

For more information about the signal behavior of the basic inputs and outputs, refer to the chapter *Behavior of Function Blocks with the Input i_xExecute* *(see page 21)*.

FB_CsvRead Considerations

Considerations

Consider the following constraints for reading a CSV file:

- Only CSV files with ASCII encoding are supported.
  It is not verified whether the file encoding is actually ASCII. If the file contains characters outside the range of the ASCII character code set, the values rendered by the application are not valid.
- The values read from the file are stored as STRING values in the application. This applies even to numeric values. Before they can be processed, the values need to be converted to the appropriate datatype. To achieve this, it is a good practice to use the `STRING_TO_` conversion functions. The conversion functions require a specific syntax of the STRING values depending on the target datatype. Consider these requirements when creating the files to simplify the processing of read values.
- File operations are time-consuming processes. To help prevent influencing time-critical control functions in your application, create a separate task with lower priority for these processes. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information about task management, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- The timeout parameter `i_timTimeout` is used to monitor the read operation. If the value specified for the timeout is exceeded during execution of the `FB_CsvRead` function block, the read operation is canceled and the function block generates an error. To select an appropriate value for the timeout parameter, consider that the read operation takes several task cycles. Thus, calculate the product of the number of required task cycles and the task interval to specify the minimum value for the timeout parameter.
- The number of cycles depends, besides the size of the file to be read, on the size of the processing block which can be specified by the parameter `Gc_uiCsvReadProccessing-BlockSize` in the GPL *(see page 49)*. Analyzing the content of the file is split into several read operations to reduce the load for the single task cycle. During each read operation, one data block is processed and, if requested, stored to the buffer. The larger the processing block size, the fewer cycles are required for reading the file. However, a larger block size increases the execution time of this single read operation.
- The character code specified as delimiter is not supported as part of a value.

FB_CsvRead Troubleshooting

Troubleshooting

The table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified directory is not available. • The given file path has invalid syntax.	• Verify that the directory exists. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator, while another controller supports '/' instead.
Execution ends with a detected error and result indicates `TableInvalid`.	• The pointer to the read buffer is not assigned.	• Verify the correct assignment of the parameter `pbyTable` in the structure `i_stTableReadValues` *(see page 37)*.
Execution ends with a detected error and result indicates `TableReadValuesInvalid`.	• The dimensions of the specified buffer (table) are not consistent.	• Verify the correct assignment of the parameter in the structure `i_stTableReadValues` *(see page 37)*.
Execution ends with a detected error and result indicates `ReadParameterInvalid`.	• The parameters specified for the read mode are inconsistent. • No character has been specified for the delimiter.	• Verify that the value selected for `etReadMode` is supported by `ET_CsvReadMode` *(see page 27)*. • If `etReadMode` = `OneRow`, the value for `uiNumOfRow` must not be 0. • If `etReadMode` = `OneColumn`, the value for `uiNumOfColumn` must not be 0. • If `etReadMode` = `OneValue`, the values for `uiNumOfRow` and `uiNumOfColumn` must not be 0. • The value for `sDelimiter` must not be empty.
Execution ends with a detected error and result indicates `Timeout`.	• The specified value for the timeout is too low.	• Increase the timeout parameter according to the number of cycles needed to read the file. Also consider the task interval. • Increase the processing block size per read operation to reduce the number of function block calls which are needed to read the file.

FB_CsvRead Example

Overview

The following example shows how the values read from a CSV file are stored in the buffer provided by the application for further processing.

Example CSV File

Content of the file example.csv with delimiter specified as semicolon(;):

```
PNo;PName;PValue;PUnit;Description
1;Velo_Max;1500;rpm;maximum velocity of the motor
2;Velo_Min;100;rpm;minimum velocity of the motor
3;Velo_ManMode_Slow;150;rpm;velocity manual mode slow move
4;Velo_ManMode_Fast;600;rpm;velocity manual mode fast move
```

Example Program

```
PROGRAM SR_Example
VAR
  fbRead :FFU.FB_CsvRead;
  xCmdRead :BOOL;
  sCsvTable:ARRAY[0..c_uiNumOfRows-1,0..c_uiNumOfColumns-
1] OF STRING(c_uiLengthOfValue);
  stCsvTable: FFU.ST_CsvTable;
END_VAR
VAR CONSTANT
  c_uiNumOfRows :UINT:= 8;
  c_uiNumOfColumns :UINT:= 7;
  c_uiLengthOfValue :UINT:= 20;
END_VAR

fbRead.i_stBufferReadValues.pbyTable := ADR(g_asCsvTable);
fbRead.i_stBufferReadValues.uiNumOfRows := c_uiNumOfRows;
fbRead.i_stBufferReadValues.uiNumOfColumns := c_uiNumOfColumns;
fbRead.i_stBufferReadValues.udiSizeOfTable := SIZEOF(asCsvTable);

fbRead.i_stReadParameter.sDelimiter := ';';
fbRead.i_stReadParameter.etReadMode := FFU.ET_CsvReadMode.AllValues;
fbRead.i_stReadParameter.uiNumOfRow := 0;
fbRead.i_stReadParameter.uiNumOfColumn := 0;

fbRead(
       i_xExecute:= xCmdRead,
       i_sFilePath:= '/sd0/Example.csv',
       i_stBufferReadValues:= ,
```

```
        i_stReadParameter:= ,
        i_timTimeout:= ,
        q_xDone=> ,
        q_xBusy=> ,
        q_xError=> ,
        q_etResult=> ,
        q_sResultMsg=> ,
        q_stFileInformation=> ,
        q_stCsvWarnValueTruncated=> ,
        xValueTruncated=> );
```

Buffer

The buffer provided by the application contains the elements read from the CSV file:

Array Index	0	1	2	3	4	5	6
0	PNo	PName	PValue	PUnit	Description	–	–
1	0	Velo_Max	1500	rpm	maximum velocity of	–	–
2	1	Velo_Min	100	rpm	minimum velocity of	–	–
3	2	Velo_Man_Slow	150	rpm	velocity manual mode	–	–
4	3	Velo_Man_Fast	600	rpm	velocity manual mode	–	–
5	–	–	–	–	–	–	–
6	–	–	–	–	–	–	–
7	–	–	–	–	–	–	–

NOTE: The values in the column Description have been truncated while reading the file because the length of the STRINGs in the read buffer is limited to 20 characters. The output `q_stWarn-ValueTruncated` of the function block indicates that the value in row 2 and column 5 has been truncated.

Section 11.2
FB_CsvWrite

What Is in This Section?

This section contains the following topics:

Topic	Page
FB_CsvWrite Functional Description	93
FB_CsvWrite Considerations	95
FB_CsvWrite Troubleshooting	96
FB_CsvWrite Example	97

FB_CsvWrite Functional Description

Overview

Type:	Function block
Available as of:	V1.0.8.0
Inherits from:	-
Implements:	-

Functional Description

The function block `FB_CsvWrite` is used to write values to a CSV file located on the file system of the controller or on the extended memory (for example, an SD memory card). It can also create a new file. For information on the file system, refer to the chapter *Flash Memory Organization* in the Programming Guide of your controller.

The data to be written to the file is stored in the buffer provided by the application as variables of type STRING. Declare the buffer in the application as a two-dimensional ARRAY of type STRING. Use the input `i_stTableWriteValues` to provide the dimensions of the array and the pointer to that array to the function block. For further information, refer to the structure `ST_CsvTable` *(see page 37)*.

The two-dimensional array represents the table structure consisting of rows and columns. Each row represents a record. The number of columns represents the maximum number of values one record can have.

The input `i_stWriteParameter` provides the parameter to control the write operation. Use the parameter `sDelimiter` to specify the character code for the delimiter that is inserted to separate the individual values of the file. The value of the parameter `etModeFileOpen` allows you to specify whether the data is to be appended to an existing file or whether a new file is to be created. Use the parameters `uiNumOfRows` and `uiNumOfColumns` to specify the amount of data to be written.

The character code `LF (0A hex)` is inserted to provoke a line break between two records.

Interface

Input	Data type	Description
i_xExecute	BOOL	The function block opens or creates the specified CSV file and writes the specified content into it upon a rising edge of this input.
i_sFilePath	STRING[255]	File path to the CSV file. If a file name is specified without file extension, the function block adds the extension .csv.
i_stWriteParameter	ST_WriteParameter	Specifies the mode for opening the CSV file and the content to be written into the file.
i_timTimeout	TIME	After this time has elapsed, the execution is canceled. If the value is T#0s, the default value T#2s is applied.
i_stTableWriteValues	ST_CsvTable	Structure to pass the buffer provided by the application to the function block (refer to the ST_CsvTable structure *(see page 37)*).

Output	Data type	Description
q_xDone	BOOL	If this output is set to TRUE, the execution has been completed successfully.
q_xBusy	BOOL	If this output is set to TRUE, the function block execution is in progress.
q_xError	BOOL	If this output is set to TRUE, an error has been detected. For details, refer to q_etResult and q_etResultMsg.
q_etResult	ET_Result	Provides diagnostic and status information as a numeric value. If q_xBusy = TRUE, the value indicates the status. If q_xDone or q_xError = TRUE, the value indicates the result.
q_sResultMsg	STRING[80]	Provides additional diagnostic and status information as a text message.
q_udiFileSize	UDINT	Provides the file size in bytes of the file recently processed.

FB_CsvWrite Considerations

Considerations

Consider the following constraints for writing a CSV file:

- File operations are time-consuming processes. To help prevent influencing time-critical control functions in your application, create a separate task with lower priority for these processes. In addition, consider whether the watchdog for this task may be disabled to avoid watchdog exceptions during the parsing process. For more information about task management, refer to the chapter *System and Task Watchdogs* in the Programming Guide of your controller.
- The timeout parameter `i_timTimeout` is used to monitor the file operation. If the value specified for the timeout is exceeded during execution of the function block, the write operation is canceled and the function block indicates an error. To select an appropriate value for the timeout parameter, consider that the file operation takes several task cycles. Thus, calculate the product of the number of required task cycles and the task interval to specify the minimum value for the timeout parameter.
- The number of cycles depends, besides the amount of data to be written, on the size of the processing block which can be specified by the parameter `Gc_uiCsvWriteProccessin-BlockSize` in the GPL *(see page 49)*. The writing data to the file is split into several write operations to reduce the load for the single task cycle. During each write operation, one data block is processed and written to the file. The larger the processing block size the fewer cycles are required for creating and writing the file. However, a larger block size increases the execution time of each write operation.

FB_CsvWrite Troubleshooting

Troubleshooting

This table describes some general issues and their solutions:

Issue	Cause	Solution
Execution ends with a detected error and result indicates `FilePathInvalid`.	• The specified directory is not available. • The given file path has invalid syntax. • The specified file does not exist on the specified file path.	• Verify that the directory exists. In case the file path targets extended memory (such as an SD memory card), verify that it is available. • Verify that the syntax used is supported by your particular controller. For example, your controller may support '\' as a separator, while another controller supports '/' instead. • For the parameter `etModeFileOpen` *(see page 28)*, select the value `AppendPlus`.
Execution ends with a detected error and result indicates `FileAlreadyExists`.	• The specified file already exists but the `etModeFileOpen` *(see page 28)* does not allow overwriting the file.	• Specify another file name. • If the existing file can be overwritten, select for the parameter `etModeFileOpen` *(see page 28)* the value `CreatePlus`.
Execution ends with a detected error and result indicates `TableInvalid`.	• The pointer to the read buffer is not assigned.	• Verify the correct assignment of the parameter `pbyTable` in the structure `i_stTableWriteValues` *(see page 37)*.
Execution ends with a detected error and result indicates `TableWriteValuesInvalid`.	• The dimensions of the specified CSV table are not consistent.	• Verify the correct assignment of the parameter in the structure `i_stTableWriteValues` *(see page 37)*.
Execution ends with a detected error and result indicates `WriteParameterInvalid`.	• The parameter specified for the write mode are inconsistent. • No character has been specified for the delimiter.	• Verify that the selected value for `etModeFileOpen` is supported by `etModeFileOpen` *(see page 28)* • The value for `sDelimiter` must not be empty.
Execution ends with a detected error and result indicates `Timeout`.	• The specified value for the timeout is too low.	• Increase the timeout parameter according to the number of cycles needed to create the file. Consider also the task interval. • Increase the processing block size per write operation to reduce the number of function block calls which are needed to create the file.

FB_CsvWrite Example

Overview

The following example shows how to implement the FB_CsvWrite function block to write data to an existing CSV file.

Buffer

The buffer provided by the application contains the elements to be written to the CSV file:

Array Index	0	1	2	3	4	5	6
0	2017-02-24	09:30:10	OpMode	Machine - AutoMode	–	–	–
1	2017-02-24	09:35:27	Error	Module1 - ErrorID 16#A123	–	–	–
2	2017-02-24	09:35:27	State	Machine - ErrorStop	–	–	–
3	–	–	–	–	–	–	–
4	–	–	–	–	–	–	–
5	–	–	–	–	–	–	–
6	–	–	–	–	–	–	–
7	–	–	–	—	–	–	–

Example Program

```
PROGRAM SR_Example
VAR
  fbWrite :FFU.FB_CsvWrite;
  xCmdWrite :BOOL;
  asCsvTable:ARRAY[0..c_uiNumOfRows-1,0..c_uiNumOfColumns-
1] OF STRING(c_uiLengthOfValue);
  stCsvTable: FFU.ST_CsvTable;
END_VAR
VAR CONSTANT
  c_uiNumOfRows :UINT:= 8;
  c_uiNumOfColumns :UINT:= 7;
  c_uiLengthOfValue :UINT:= 80;
END_VAR

fbWrite.i_stTableWriteValues.pbyTable := ADR(asCsvTable);
fbWrite.i_stTableWriteValues.uiNumOfRows := c_uiNumOfRows;
fbWrite.i_stTableWriteValues.uiNumOfColumns := c_uiNumOfColumns;
```

```
fbWrite.i_stTableWriteValues.udiSizeOfTable := SIZEOF(asCsvTable);

fbWrite.i_stWriteParameter.sDelimiter := ';';
fbWrite.i_stWriteParameter.etModeFileOpen := FFU.ET_ModeFileOpen.Append
;
fbWrite.i_stWriteParameter.uiNumOfRows := 3;
fbWrite.i_stWriteParameter.uiNumOfColumns := c_uiNumOfColumns;

fbWrite(
       i_xExecute:= xCmdWrite,
       i_sFilePath:= '/sd0/Example.csv',
       i_stWriteParameter:= ,
       i_timTimeout:= ,
       i_stTableWriteValues:= ,
       q_xDone=> ,
       q_xBusy=> ,
       q_xError=> ,
       q_etResult=> ,
       q_sResultMsg=> ,
       q_stFileInformation=> );
```

Example CSV File

Content of example.csv after appending new data:

```
Date;Time;Eventype;Event;;;;
2017-02-24;09:27:36;OpMode;Machine - Power on;;;;
2017-02-24;09:27:54;OpMode;Machine - Manual Mode;;;;
2017-02-24;09:28:32;OpMode;Machine - Homing;;;;
2017-02-24;09:29:44;State;Machine - All Modules Homed;;;;
2017-02-24;09:30:10;OpMode;Machine - AutoMode;;;;
2017-02-24;09:35:27;Error;Module1 - ErrorID 16#A123;;;;
2017-02-24;09:35:27;State;Machine - ErrorStop;;;;
```

Glossary

A

application

A program including configuration data, symbols, and documentation.

C

configuration

The arrangement and interconnection of hardware components within a system and the hardware and software parameters that determine the operating characteristics of the system.

controller

Automates industrial processes (also known as programmable logic controller or programmable controller).

E

expansion bus

An electronic communication bus between expansion I/O modules and a controller.

I

I/O

(*input/output*)

P

program

The component of an application that consists of compiled source code capable of being installed in the memory of a logic controller.

Index

A

aliases, *43*

C

common inputs and outputs
- behavior of function blocks with the input i_xExecute, *21*

E

ET_CsvReadMode, *27*
- AllValues, *27*
- GetFileInformation, *27*
- OneColumn, *27*
- OneRow, *27*
- OneValue, *27*

ET_ModeFileOpen, *28*
- Append, *28*
- AppendPlus, *28*
- Create, *28*
- CreatePlus, *28*
- NotSet, *28*

ET_Result, *29*
- AdditionalContentInvalid, *30*
- AnalyzingFile, *29*
- BufferFull, *30*
- CheckingInputs, *29*
- ClosingFile, *29*
- ElementNotFound, *30*
- FileAlreadyExists, *30*
- FileCloseFailed, *29*
- FileFormatUtility, *29*
- FileInconsistent, *30*
- FileInvalid, *29*
- FilenameInvalid, *31*
- FilenameTooLong, *31*
- FileNotExists, *30*
- FileOpenFailed, *29*
- FilePathInvalid, *29*
- FilePathTooLong, *31*
- FileReadFailed, *30*
- FileWriteFailed, *29*
- FirstItemInvalidParentIndex, *31*
- FirstItemNoElement, *31*
- GetFileSize, *29*
- GetFileSizeFailed, *30*
- Idle, *29*
- Initializing, *29*
- NumOfParentsExceeded, *30*
- OK, *29*
- OpeningFile, *29*
- ParsingFailed, *30*
- ReadingFile, *29*
- ReadParameterInvalid, *30*
- TableInvalid, *30*
- TableReadValuesInvalid, *30*
- TableTooSmall, *30*
- TableWriteValuesInvalid, *30*
- Timeout, *29*
- UnexpectedProgramBehaviour, *31*
- ValueNotFound, *30*
- WriteParameterInvalid, *30*
- WritingFile, *29*

XmlItemTypeInvalid, *30*
XmlStructureInconsistent, *30*
XPathExpressionInvalid, *29*
XpathExpressionNotSupported, *31*
XpathRootElementDoesNotMatch, *31*
ET_XmlItemType, *26*
Attribute, *26*
Element, *26*
NotSet, *26*

F

FB_CsvRead, *84*
FB_CsvRead considerations, *88*
FB_CsvRead example, *90*
FB_CsvRead functional description, *85*
FB_CsvRead troubleshooting, *89*
FB_CsvWrite, *92*
FB_CsvWrite considerations, *95*
FB_CsvWrite example, *97*
FB_CsvWrite functional description, *93*
FB_CsvWrite troubleshooting, *96*
FB_XmlRead, *58*
FB_XmlRead considerations, *62*
FB_XmlRead example, *64*
FB_XmlRead functional description, *59*
FB_XmlRead troubleshooting, *63*
FB_XmlWrite, *66*, *67*, *69*, *71*
FB_XmlWrite troubleshooting, *70*
FC_EtResultToString, *53*
FC_XmlGetElementValue, *74*
FC_XmlGetElementValue considerations, *77*
FC_XmlGetElementValue functional description, *75*
FC_XmlSetElementValue, *78*
FC_XmlSetElementValue considerations, *80*
FC_XmlSetElementValue functional description, *79*
FileFormatUtility, *17*
GCL (Global Constants List), *47*
GPL, *49*

G

GCL (Global Constants List)
FileFormatUtility, *47*
GPL
FileFormatUtility, *49*

S

ST_CsvFileInformation, *39*
ST_CsvReadParameter, *41*
ST_CsvTable, *37*
ST_CsvWarnValueTruncated, *40*
ST_CsvWriteParameter, *42*
ST_XmlItem, *34*
ST_XmlUserDefinedHeader, *36*

X

XmlItems, *43*