# FileIO

IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers

# Table of Contents

RTAC LIBRARY

# FileIO

## Introduction

The FileIO library includes the internal RTAC sel_file and sel_ftp_client libraries. This library provides function blocks that simplify asynchronous file management for basic file read and write operations. It also provides access to the underlying firmware interface libraries.

Because the classes this library provides manage asynchronous file operations, the user must call the `Run()` method of instantiated classes on every scan to ensure that all queued work is correctly performed and monitored. Each class provides various methods to initiate new read or write operations, collect data, or cause other changes in state.

**NOTE:** See the ACSELERATOR RTAC Library Extensions Instruction Manual (LibraryExtensionsIM) for explanation of the concepts used by the object-oriented extensions to the IEC 61131-3 standard.

## Special Considerations

➤ Copying classes from this library causes unwanted behavior. This means the following:

1. The assignment operator ":=" must not be used on any class from this library; consider assigning pointers to the objects instead.

```
// This is bad and in most cases will provide a compiler error
    such as:
// "C0328: Assignment not allowed for type class_FileIOObject"
myFileIOObject := otherFileIOObject;
```

```
// This is fine
someVariable := myFileIOObject.value;
// As is this
pt_myFileIOObject := ADR(myFileIOObject);
```

2. Classes from this library must never be VAR_INPUT or VAR_OUTPUT members in function blocks, functions, or methods. Place them in the VAR_-IN_OUT section or use pointers instead.

- ➤ Classes in this library have memory allocated inside them. As such, they should only be created in environments of permanent scope (e.g., Programs, Global Variable Lists, or VAR_STAT sections).
- ➤ All file read operations are done completely into RAM. This means that the reading of large files that exceed the available RAM will not work as expected.

## Supported Firmware Versions

Versions 3.5.4.0 and later of this library must be used with an RTAC device that is running firmware version R144-V1 firmware or later.

Versions 3.5.3.0 and later of this library must be used with an RTAC device that is running firmware version R143-V0 firmware or later.

Versions 3.5.2.0 and later of this library must be used on an RTAC device that is running firmware version R136-V2 or later.

Version 3.5.1.0 of this library must be used on an RTAC device that is running firmware version R135.

Previous versions of this library can be used on firmware versions R132–R135.

To enable FileIO library support, the device number of your RTAC must include the correct feature in its model option table (MOT). You cannot download projects that include this library to RTACs that do not support the library. Use the SEL website MOT configuration (https://selinc.com/products/) to ensure that a particular part number has FileIO support enabled.

## Global Parameters

The library applies the following values as maximums; they can be modified when the library is included in a project.

| Name | IEC 61131 Type | Value | Description |
|---|---|---|---|
| g_p_FileIo_MaxBufferSize | UDINT | 1048576 | The maximum internal buffer size allowed for class_FileReader2 or class_FileWriter. This value is the maximum file size you can read in, and the maximum amount you can append to a file at one time. The default value is $1024^2 \cdot 10$. |

## Enumerations

Enumerations make code more readable by allowing a specific number to have a readable textual equivalent.

## enum_FtpSendSchedule

| Enumeration | Description |
|---|---|
| ON_CLOSE | When the previous log file is closed, as part of starting a new log file with the `StartNewLog()` method, synchronize the closed log to the FTP server. |
| ON_UPDATE | Send active log file each time an additional log is added. |

## sel_file.Enum_protocol_id

This enumeration is defined in the underlying sel_file library. It lists the protocols that can create an event.

| Enumeration | Value |
|---|---|
| NO_PROTOCOL_SPECIFIED | 0 |
| SEL_CLIENT | 1 |
| SEL_SERVER | 2 |
| MODBUS_CLIENT | 3 |
| MODBUS_SERVER | 4 |
| DNP_CLIENT | 5 |
| DNP_SERVER | 6 |
| SEL_MIRRORED_BITS | 7 |
| C37118_CLIENT | 8 |
| GOOSE_RX | 9 |
| ACCESS_POINT_CLIENT | 10 |
| ACCESS_POINT_SERVER | 11 |
| GOOSE_TX | 12 |
| ETHERCAT | 13 |
| DNP_SERVER_FAILOVER | 14 |
| IEC61850_CLIENT | 15 |
| IEC61850_SERVER | 16 |
| NGVL | 17 |
| LG8979_CLIENT | 18 |
| LG8979_SERVER | 19 |
| IEC60870_CLIENT | 20 |
| IEC60870_SERVER | 21 |
| C37118_SERVER | 22 |
| SES92_SERVER | 23 |
| PGE2179_CLIENT | 24 |
| FLEX_PARSE_CLIENT | 25 |

## sel_file.Enum_event_type

This enumeration is defined in the underlying sel_file library. This enumeration defines the types of events the database can store. It is part of the event handle and should be used to help decide which sel_file.Enum_event_data to use when opening an event.

| Enumeration | Description |
|---|---|
| NO_EVENT_TYPE | There is no event type associated with this file. |
| CEV_FILE | A plaintext file containing event data. |
| COMTRADE | A zipped folder containing the event data as COMTRADE files. |

## sel_file.Enum_event_data

This enumeration is defined in the underlying sel_file library. It defines how that library attempts to open events.

| Enumeration | Description |
|---|---|
| RAW_DATA | Return the data exactly as it is stored in the database. |
| CFG_FILE | Treat the data as an archive and extract the first file with a .cfg extension. |
| DAT_FILE | Treat the data as an archive and extract the first file with a .dat extension. |
| HDR_FILE | Treat the data as an archive and extract the first file with a .hdr extension. |
| INF_FILE | Treat the data as an archive and extract the first file with a .inf extension. |

## sel_file.Enum_sel_file_errors

This enumeration is defined in the underlying sel_file library. It defines the status of file and SOE requests. After a call to a function, variables of this type will display IN_PROGRESS for a time, after which they will change to some other value. NO_ERROR implies that the task completed successfully, and SYSTEM_BUSY means that the driver already has too many jobs queued that it must complete before accepting any more jobs. In this case, a subsequent request might succeed if one of the queued jobs has been completed. Any other result should be descriptive of the error encountered.

| Enumeration | Description |
|---|---|
| NO_ERROR | The request completed successfully. |
| FILE_NOT_FOUND | The requested file was not found in the file system. |
| INVALID_FILE_NAME | The file name provided was invalid. |
| INVALID_FH | The file handle provided was not for an open file. |
| INVALID_FILTER | The filter provided was invalid. |
| INVALID_TIMESTAMP | The time stamp provided was invalid. |
| FS_OUT_OF_SPACE | The file system did not have enough space to perform the action. |
| DIR_LIST_NOT_INIT | The directory iterator has not be initialized. |
| SYSTEM_BUSY | The system is too busy to process the request. |
| TOO_MANY_TASKS | The system has received requests from more than two tasks. |
| OPERATION_FAILED | There was a system call failure while processing the request. |
| IN_PROGRESS | The system is processing the request. |

### sel_ftp_client.Enum_sel_ftp_client_errors

This enumeration is defined in the underlying sel_ftp_client library. It defines the status of FTP requests. After a call to a function, variables of this type will display IN_PROGRESS for a time, after which they will change to some other value. NO_ERROR implies that the task completed successfully, and SYSTEM_BUSY means that the driver already has too many jobs queued that it must complete before accepting any more jobs. In this case, a subsequent request might succeed if one of the queued jobs has been completed. Any other result should be descriptive of the error encountered.

| **Enumeration** | **Description** |
|---|---|
| NO_ERROR | The request successfully triggered an FTP attempt. |
| INIT_FAILED | The FTP process is not responding. |
| INVALID_OPERATION | There was a system call failure while processing the request. |
| INVALID_IP | The IP address provided was not in the form *XXX.XXX.XXX.XXX*, where *XXX* is an integer that is $\leq$ 255. |
| INVALID_USR_NAME | The username provided contained invalid characters. |
| INVALID_PASSWORD | The password provided contained invalid characters. |
| INVALID_FILE_NAME | The file name provided contained invalid characters. |
| INVALID_MAX_TIME | The time provided was less than or equal to 0. |
| FS_OUT_OF_SPACE | The file system did not have enough space to perform the action. |
| SYSTEM_BUSY | The system is too busy to process the request. |
| OPERATION_TIMEOUT | The FTP attempt took longer than the provided time-out. |
| IN_PROGRESS | The system is processing the request. |

## Structures

Structures provide a means to group together several memory locations (variables), making them easier to manage.

### struct_EventDetails

| **Name** | **IEC 61131 Type** | **Description** |
|---|---|---|
| Handle | Struct_event_handle | The details required to access this event via the database. |
| Device | STRING(32) | The name of the device the event was collected from. |
| TimeStamp | DT | The time stamp of the event as seconds since epoch. |
| TimeMilliseconds | UINT | The millisecond at which the event occurred. |
| FileSize | DINT | The size of the event in the database in bytes. |

### sel_file.Struct_event_handle

This struct is defined in the underlying sel_file library. This struct contains all information required to uniquely identify an event in the database.

| Name | IEC 61131 Type | Description |
|---|---|---|
| EventID | LINT | An identifier of the event. |
| ProtocolID | Enum_protocol_id | An identifier of the protocol that gathered the event. |
| EventType | Enum_event_type | The type of the event file. |

## sel_file.Struct_soe_content

This struct is defined in the underlying sel_file library. This struct contains all value fields returned by an SOE query.

| Name | IEC 61131 Type | Description |
|---|---|---|
| DeviceName | STRING(255) | The name of the device that logged this event on the RTAC. |
| TagName | STRING(255) | The tag that changed prompting this SOE to be logged. |
| Message | STRING(255) | The message of this SOE. |
| Category | STRING(255) | The category string provided for this SOE. |
| Priority | STRING(255) | The priority string provided for this SOE. |
| TimeStamp | DT | The time stamp of the event as seconds since epoch. |
| Millisecond | UINT | The time at which the event occurred with resolution to the ms. |
| DSTOffset | INT | The daylight-saving time offset that should be applied to the time stamp. |
| UTCOffset | INT | The timezone offset that should be applied to the time stamp. |
| AlarmEnabled | BOOL | This SOE can trigger an alarm. |

## sel_file.Struct_soe_content_id

This struct is defined in the underlying sel_file library. This struct contains all value fields returned by an SOE query.

| Name | IEC 61131 Type | Description |
|---|---|---|
| DeviceName | STRING(255) | The name of the device that logged this event on the RTAC. |
| TagName | STRING(255) | The tag that changed prompting this SOE to be logged. |
| Message | STRING(255) | The message of this SOE. |
| Category | STRING(255) | The category string provided for this SOE. |
| Priority | STRING(255) | The priority string provided for this SOE. |
| TimeStamp | DT | The time stamp of the event as seconds since epoch. |
| Millisecond | UINT | The time at which the event occurred with resolution to the ms. |
| DSTOffset | INT | The daylight-saving time offset that should be applied to the time stamp. |
| UTCOffset | INT | The timezone offset that should be applied to the time stamp. |
| AlarmEnabled | BOOL | This SOE can trigger an alarm. |
| RemoteSoe | BOOL | The SOE was generated by a device other than the local RTAC. |
| ID | STRING(80) | A unique identifier for this SOE. |

## sel_file.Struct_soe_filter

This struct is defined in the underlying sel_file library. This struct contains all filters possible to apply to an SOE query. If left empty, no filter will be applied on that field.

The filter string must contain only letters (case-sensitive); numbers; the symbols _, -, and " " (space); and the wildcard characters * and ?. The character * acts as a multicharacter wildcard and the character ? acts as a single character wildcard when inside any string.

| Name | IEC 61131 Type | Description |
|---|---|---|
| DeviceNameFilter | STRING(255) | The filter to place on the device name. |
| TagNameFilter | STRING(255) | The filter to use on the tag name. |
| MessageFilter | STRING(255) | The filter to use on the SOE message. |
| CategoryFilter | STRING(255) | The filter to use on the SOE category. |
| PriorityFilter | STRING(255) | The filter to use on the SOE priority. |
| ReturnAlarmSoeOnly | BOOL | Only return SOEs that can trigger an alarm. |

# Functions

This library provides the following functions, which allow single-operation asynchronous actions. Each call has a status argument that must persist across multiple scans. Calling any function does not complete the requested work, but rather queues the work to be performed over multiple scans. This means you should only call a given function once per desired operation. The system updates the status variable automatically when the operation completes in either success or failure. Avoid reusing pointers or variables passed as VAR_-IN_OUT until the status variable used has changed from the value of IN_PROGRESS.

## fun_FtpDownload

Use this function to download files to the local, sequestered file system from an FTP server.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| ftpServer | STRING(15) | The IP address of the FTP server being contacted. |
| localPath | STRING(255) | The local path and file name to which you are writing. It must begin with "/" and contain the full file path. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). |
| remotePath | STRING(255) | The complete path and file name of the file to download from the FTP server. It must contain only printable characters (ASCII values 0x32 to 0x7E inclusive), excluding single and double quotation characters. |
| username | STRING(32) | The username to be used. This must only contain alphanumeric characters and "_". |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| password | STRING(32) | The password for use on the FTP server. This may contain any printable ASCII characters excluding the quote characters. |
| timeout | UDINT | The number of seconds for the FTP attempt to run before it is canceled. This value must be greater than 0. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| status | Enum_sel_ftp_client_errors | The active status of this FTP event. |

### Processing

- Sets status to IN_PROGRESS.
- Queues the FTP download request for processing.
- Validates that the time-out exceeds zero seconds.
- Validates that all string characters meet the prescribed requirements.
- Validates that at least 250 MB of space are available in the file system for the download contents.
- Attempts to FTP the file onto the RTAC as an asynchronous event.
- A status of NO_ERROR implies that the FTP command was successfully issued, the FTP service completed, and the service returned no error code.

## fun_FtpEventUpload

Use this function to upload event files from the database to an FTP server.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| ftpServer | STRING(15) | The IP address of the FTP server being contacted. |
| localEvent | Struct_event_handle | The event file to upload. |
| remotePath | STRING(255) | The complete path and file name of the file to save on the FTP server. It must contain only printable characters (ASCII values 0x32 to 0x7E inclusive), excluding single and double quotation characters. |
| username | STRING(32) | The username to be used. This must only contain alphanumeric characters and "_". |
| password | STRING(32) | The password for use on the FTP server. This may contain any printable ASCII characters excluding the quote characters. |
| timeout | UDINT | The number of seconds for the FTP attempt to run before it is canceled. This value must be greater than 0. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| status | Enum_sel_ftp_client_errors | The active status of this FTP event. |

### Processing

- ➤ Sets status to IN_PROGRESS.
- ➤ Queues the FTP upload request for processing.
- ➤ Attempts to FTP the file from the RTAC as an asynchronous event.
- ➤ A status of NO_ERROR implies that the FTP command was successfully issued, the FTP service completed, and the service returned no error code.

## fun_FtpUpload

Use this function to upload individual files from the local, sequestered file system to an FTP server.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| ftpServer | STRING(15) | The IP address of the FTP server being contacted. |
| localPath | STRING(255) | The complete local path and file name to upload. It must begin with “/” and contain the full file path. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). |
| remotePath | STRING(255) | The complete path and file name of the file to write on the FTP server. It must contain only printable characters (ASCII values 0x32 to 0x7E inclusive), excluding single and double quotation characters. |
| username | STRING(32) | The username to be used. This must only contain alphanumeric characters and “_”. |
| password | STRING(32) | The password for use on the FTP server. This may contain any printable ASCII characters excluding the quote characters. |
| timeout | UDINT | The number of seconds for the FTP attempt to run before it is canceled. This value must be greater than 0. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| status | Enum_sel_ftp_client_errors | The active status of this FTP event. |

### Processing

- ➤ Sets status to IN_PROGRESS.
- ➤ Queues the FTP upload request for processing.
- ➤ Attempts to FTP the file from the RTAC as an asynchronous event.
- ➤ A status of NO_ERROR implies that the FTP command was successfully issued, the FTP service completed, and the service returned no error code.

## fun_DeleteFile

Use this function to delete any file or empty folder from the sequestered file system.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filename | STRING(255) | The full path and file name of the file to delete. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| status | Enum_sel_file_errors | The variable where the state of the asynchronous task that is deleting the file will be reported. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the file deletion request for processing.
- ➤ If *status* later changes to NO_ERROR, the system successfully found and deleted the file.
- ➤ If *status* later changes to OPERATION_FAILED, the system failed to either find or delete the file.
- ➤ If *status* later changes to anything else, the system stopped the request before the deletion command was issued.

## fun_FileSize

Use this function to request the size of any file in the sequestered file system. If the size of the file provided exceeds UDINT max (4,294,967,295) bytes, then the value that is returned will roll over and equal the file size modulo 4,294,967,296.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filename | STRING(255) | The full path and file name for the size calculations. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| status | Enum_sel_file_errors | The variable where the state of the asynchronous task that is obtaining the file size will be reported. |
| sizeInBytes | UDINT | After completion, this variable contains the size of the file in bytes. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the file size request for processing.
- ➤ If *status* later changes to NO_ERROR, *sizeInBytes* contains the file size.
- ➤ If *status* later changes to OPERATION_FAILED, then the system failed to find the file.
- ➤ If *status* later changes to anything else, *sizeInBytes* is undefined.

## fun_FilesystemFreeSpace

Use this function to validate how much usable space remains in the file system. FileIO will not use all the space on the file system, and the value returned by this function reflects that. When the value this function returns reaches zero, additional writes to the file system through FileIO will fail.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| status | Enum_sel_file_errors | The variable where the state of the asynchronous task that is obtaining the file system free space will be reported. |
| spaceAvailable | ULINT | After completion, this variable contains the space left in the file system that the FileIO library can use, in bytes. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the file system size request for processing.

- ➤ If *status* later changes to NO_ERROR, *spaceAvailable* contains the file system free space.
- ➤ If *status* later changes to anything else, *spaceAvailable* is undefined.

## fun_SoeAscending

Use this function to request a limited number of SOEs from the database, beginning with a specified time and moving toward the future.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_soeBuffer | POINTER TO Struct_soe_content | Pointer to the array to populate with the SOE data. The array provided must have at least *maxSoeCount* members, or memory corruption will occur. |
| startTime | DT | The earliest time to include in the results. This value must be between the years 2000 and 2037, or the call will result in an error. |
| maxSoeCount | UINT | The maximum number of SOEs to place in *pt_-soeBuffer*. This number must exceed zero to obtain a non-error result and it must not exceed the number of Struct_soe_content objects that can fit in the memory space *pt_soeBuffer* points to, or memory corruption will occur. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable that reports the state of the asynchronous task that is obtaining the SOE list. |
| soeCount | UINT | The location to report the number of SOEs placed in *pt_-soeBuffer*. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the SOE request for processing.
- ➤ If *status* later changes to NO_ERROR, *soeCount* SOEs were placed at location *pt_soeBuffer*.
- ➤ If *status* later changes to anything else, values at *pt_soeBuffer* remain unchanged.

## fun_SoeDescending

Use this function to request a limited number of SOEs from the database, beginning with a specified time and moving toward the past.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_soeBuffer | POINTER TO Struct_soe_content | Pointer to the array to populate with the SOE data. The array provided must have at least *maxSoeCount* members, or memory corruption will occur. |
| startTime | DT | The latest time to include in the results. This value must be between the years 2000 and 2037, or the call will result in an error. |
| maxSoeCount | UINT | The maximum number of SOEs to place in *pt_-soeBuffer*. This number must exceed zero to obtain a non-error result and it must not exceed the number of Struct_soe_content objects that can fit in the memory space *pt_soeBuffer* points to, or memory corruption will occur. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable that reports the state of the asynchronous task that is obtaining the SOE list. |
| soeCount | UINT | The location to report the number of SOEs placed in *pt_-soeBuffer*. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the SOE request for processing.
- ➤ If *status* later changes to NO_ERROR, *soeCount* SOEs were placed at location *pt_soeBuffer*.
- ➤ If *status* later changes to anything else, values at *pt_soeBuffer* remain unchanged.

## fun_SoeWindow

Use this function to request a limited number of SOEs from the database, beginning with a specified time and moving toward the specified, future end time.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_soeBuffer | POINTER TO Struct_soe_content | Pointer to the array to populate with the SOE data. The array provided must have at least *maxSoeCount* members, or memory corruption will occur. |
| startTime | DT | The earliest time to include in the results. This value must be between the years 2000 and 2037, or the call will result in an error. |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| endTime | DT | The latest time to include in the results. This value must be between the years 2000 and 2037, or the call will result in an error. |
| maxSoeCount | UINT | The maximum number of SOEs to place in *pt_-soeBuffer*. This number must exceed zero to obtain a non-error result and it must not exceed the number of Struct_soe_content objects that can fit in the memory space *pt_soeBuffer* points to, or memory corruption will occur. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable that reports the state of the asynchronous task that is obtaining the SOE list. |
| soeCount | UINT | The location to report the number of SOEs placed in *pt_-soeBuffer*. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the SOE request for processing.
- ➤ If *status* later changes to NO_ERROR, *soeCount* SOEs were placed at location *pt_soeBuffer*.
- ➤ If *status* later changes to anything else, values at *pt_soeBuffer* remain unchanged.

## fun_LocalSoeGetID

Use this function to request the data of a single SOE after a provided time stamp. If the system finds no SOE after the time stamp, it provides the nearest SOE before the time stamp. If the system finds no SOEs, then *status* reports an error. Any data returned are for an SOE the local RTAC generated.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| soeTime | DT | The time near which to search for an SOE. This value must be between the years 2000 and 2037, or the call will result in an error. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable where the state of the asynchronous task that is obtaining the SOE id will be reported. |
| soeData | Struct_soe_content_id | The location that will be populated with the information describing the returned SOE. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the SOE request for processing.
- ➤ If *status* later changes to NO_ERROR, *soeData* contains the pertinent information for one local SOE.
- ➤ If *status* later changes to anything else, the contents of *soeData* are undefined.

## fun_LocalSoeAscending

Use this function to request a limited number of SOEs from the database beginning with a specified SOE and moving toward the future. The order of the returned SOEs is the time of their creation on the RTAC, not the time of the actual event. The local RTAC generated all returned SOEs.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_soeBuffer | POINTER TO Struct_soe_content_id | Pointer to the array to populate with the SOE data. The array provided must have at least *max-SoeCount* members, or memory corruption will occur. |
| startID | STRING(80) | The unique identifier of a local SOE. |
| maxSoeCount | UINT | The maximum number of SOEs to place in *pt_-soeBuffer*. This number must exceed zero to obtain a non-error result and it must not exceed the number of Struct_soe_content_id objects that can fit in the memory space *pt_soeBuffer* points to, or memory corruption will occur. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable that reports the state of the asynchronous task that is obtaining the SOE list. |
| soeCount | UINT | The location to report the number of SOEs placed in *pt_-soeBuffer*. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the SOE request for processing.
- ➤ The selection of SOEs begins at the SOE after *startID*. If *startID* represent an invalid SOE, the system returns no SOEs.
- ➤ If *status* later changes to NO_ERROR, *soeCount* SOEs were placed at location *pt_soeBuffer*.
- ➤ If *status* later changes to anything else, values at *pt_soeBuffer* remain unchanged.

## fun_LocalSoeDescending

Use this function to request a limited number of SOEs from the database beginning with a specified SOE and moving toward the past. The order of the returned SOEs is the time of their creation on the RTAC, not the time of the actual event. The local RTAC generated all returned SOEs.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_soeBuffer | POINTER TO Struct_soe_content_id | Pointer to the array to populate with the SOE data. The array provided must have at least *max-SoeCount* members, or memory corruption will occur. |
| startID | STRING(80) | The unique identifier of a local SOE. |
| maxSoeCount | UINT | The maximum number of SOEs to place in *pt_-soeBuffer*. This number must exceed zero to ob-tain a non-error result and it must not exceed the number of Struct_soe_content_id objects that can fit in the memory space *pt_soeBuffer* points to, or memory corruption will occur. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable that reports the state of the asynchronous task that is obtaining the SOE list. |
| soeCount | UINT | The location to report the number of SOEs placed in *pt_-soeBuffer*. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the SOE request for processing.
- ➤ The selection of SOEs begins at the SOE before *startID*. If *startID* represent an invalid SOE, the system returns no SOEs.

- ➤ If *status* later changes to NO_ERROR, *soeCount* SOEs were placed at location *pt_soeBuffer*.
- ➤ If *status* later changes to anything else, values at *pt_soeBuffer* remain unchanged.

## fun_RemoteSoeGetID

Use this function to request the data of a single SOE after a provided time stamp. If the system finds no SOE after the time stamp, it provides the nearest SOE before the time stamp. If the system finds no SOEs, then *status* reports an error. Any data returned are an SOE a device other than the local RTAC generated.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| soeTime | DT | The time near which to search for an SOE. This value must be between the years 2000 and 2037, or the call will result in an error. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable where the state of the asynchronous task that is obtaining the SOE id will be reported. |
| soeData | Struct_soe_content_id | The location that will be populated with the information describing the returned SOE. |

### Processing

- ➤ Sets *status* to IN_PROGRESS.
- ➤ Queues the SOE request for processing.
- ➤ If *status* later changes to NO_ERROR, *soeData* contains the pertinent information for one remote SOE.
- ➤ If *status* later changes to anything else, the contents of *soeData* are undefined.

## fun_RemoteSoeAscending

Use this function to request a limited number of SOEs from the database beginning with a specified SOE and moving toward the future. The order of the returned SOEs is the time of their creation on the RTAC, not the time of the actual event. A device other than the local RTAC generated all SOEs returned.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_soeBuffer | POINTER TO Struct_soe_content_id | Pointer to the array to populate with the SOE data. The array provided must have at least *maxSoeCount* members, or memory corruption will occur. |
| startID | STRING(80) | The unique identifier of a remote SOE. |
| maxSoeCount | UINT | The maximum number of SOEs to place in *pt_soe-Buffer*. This number must exceed zero to obtain a non-error result and it must not exceed the number of Struct_soe_content_id objects that can fit in the memory space *pt_soeBuffer* points to, or memory corruption will occur. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable that reports the state of the asynchronous task that is obtaining the SOE list. |
| soeCount | UINT | The location to report the number of SOEs placed in *pt_-soeBuffer*. |

### Processing

- Sets *status* to IN_PROGRESS.
- Queues the SOE request for processing.
- The selection of SOEs begins at the SOE after *startID*. If *startID* represent an invalid SOE, the system returns no SOEs.
- If *status* later changes to NO_ERROR, *soeCount* SOEs were placed at location *pt_soeBuffer*.
- If *status* later changes to anything else, values at *pt_soeBuffer* remain unchanged.

## fun_RemoteSoeDescending

Use this function to request a limited number of SOEs from the database beginning with a specified SOE and moving toward the past. The order of the returned SOEs is the time of their creation on the RTAC, not the time of the actual event. A device other than the local RTAC generated all SOEs returned.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_soeBuffer | POINTER TO Struct_soe_content_id | Pointer to the array to populate with the SOE data. The array provided must have at least *maxSoeCount* members, or memory corruption will occur. |
| startID | STRING(80) | The unique identifier of a remote SOE. |
| maxSoeCount | UINT | The maximum number of SOEs to place in *pt_soe-Buffer*. This number must exceed zero to obtain a non-error result and it must not exceed the number of Struct_soe_content_id objects that can fit in the memory space *pt_soeBuffer* points to, or memory corruption will occur. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filters | Struct_soe_filter | The values defining the filter criteria. |
| status | Enum_sel_file_errors | The variable that reports the state of the asynchronous task that is obtaining the SOE list. |
| soeCount | UINT | The location to report the number of SOEs placed in *pt_-soeBuffer*. |

### Processing

➤ Sets *status* to IN_PROGRESS.

➤ Queues the SOE request for processing.

➤ The selection of SOEs begins at the SOE after *startID*. If *startID* represent an invalid SOE, the system returns no SOEs.

➤ If *status* later changes to NO_ERROR, *soeCount* SOEs were placed at location *pt_soeBuffer*.

➤ If *status* later changes to anything else, values at *pt_soeBuffer* remain unchanged.

# Classes

This library provides the following classes as extensions of the IEC 61131 function block.

## class_DirectoryListing (Class)

This class calls the `sel_file.sel_open_dir()` function after a new listing is requested by the call to `CreateNewList()`. On the task where this class is instantiated, `Run()` must be called once on every scan to handle all of the asynchronous file system interactions. While there are still items to list, `Run()` will call `sel_file.sel_read_dir()` once each scan until the complete directory listing is built.

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| InProgress | BOOL | Stays TRUE while the `Run()` method constructs the listing. The class ignores any calls to `CreateNewList()` while this output is TRUE. |
| Error | BOOL | TRUE if the directory listing could not be created. |
| ErrorDesc | STRING(255) | The last error encountered is described here. |
| NewListReady | BOOL | Once a list has been built, this output is set to TRUE. |

## CreateNewList (Method)

This is one of the methods that can be called each time a new directory listing is required. If no filter is given, it will provide a complete listing of the directory.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| directoryName | STRING(255) | The directory path to get file list from. Path separators should be the "/" character. |
| filter | STRING(255) | If not blank, only those files that contain this substring will be appended to the list. |

### Processing

- ➤ Clears *Error*.
- ➤ Sets the NewListReady value to FALSE and destroys any internal lists.
- ➤ Initiates the enumeration of the directory, carried out by the `Run()` method.

## CreateNewerThanList (Method)

This is one of the methods that can be called each time a new directory listing is required. This method causes a list to be generated that contains all files with a date code equal to or newer than the value passed in.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| directoryName | STRING(255) | The directory path to get file list from. Path separators should be the "/" character. |
| filter | STRING(255) | If not blank, only those files that contain this substring will be appended to the list. |
| mtimeSec | DT | Last UTC modification time (DT#yyyy-mm-dd-00:00:00) |

### Processing

- ➤ Clears *Error*.
- ➤ Sets the NewListReady value to FALSE and deletes any internal lists.
- ➤ Initiates the enumeration of the directory, carried out by the `Run()` method.

## GetList (Method)

The call to this method must occur after the NewListReady output is TRUE to obtain the populated class_SELStringList. There can only be one call to this method per created list.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| list | SELString.class_SELStringList | The class_SelStringList to write the directory listing to. See the SELString library for more information about the type class_SELStringList. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the class_SELStringList provided was populated successfully. |

### Processing

- ➤ Returns FALSE if *NewListReady* is not TRUE.
- ➤ Populates *list* with the prepared list.
- ➤ Sets the NewListReady class output to FALSE.
- ➤ Destroys the internal list.

## Run (Method)

Call this method on every scan. It supervises the asynchronous directory listing.

### Processing

- ➤ If a directory listing has been initiated:
  - ➢ Ensure that the directory is opened, using the `sel_file.sel_open_dir()` function.
  - ➢ Repeatedly call the `sel_file.sel_read_dir()` and append a class_SELString to *list* for each file name containing the *filter* substring, until no more file names are returned.

- ➤ Once the directory listing is complete, it closes the operation by setting *NewListReady* to TRUE and calling `sel_file.sel_close_dir()`.
- ➤ If any error occurs, *Error* is set to TRUE and *ErrorDesc* is populated appropriately.

## class_EventReportListing (Class)

This class has been completely removed from the library. If it was included in projects, these projects will now provide compile-time errors. If you want event access, the class_-EventListing provides access to those files along with non-obfuscated file names, the ability to filter queries based on several criteria, and the ability to properly open COMTRADE file collections to view individual files.

Please note that the class_EventListing class does have one limitation that this class did not. All class_EventListing objects on a single RTAC task (e.g., Main or Automation) share an internal iterator. It is best practice to only have one class_EventListing per task.

## class_EventListing (Class)

This class calls `sel_file.sel_begin_event_iterator_filtered()` after a new listing request activated by a call to `CreateNewList()` or `CreateNewFilteredList()`. On the task in which this class is instantiated, `Run()` must be called once on every scan to handle all of the asynchronous file system interactions. While there are still items to list, `Run()` calls `sel_file.sel_next_event()` once each scan until the complete directory listing is built.

All class_EventListing objects on a single RTAC task (e.g., Main or Automation) share an internal iterator. It is best practice to only have one class_EventListing per task.

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| InProgress | BOOL | Stays TRUE while the `Run()` method constructs the listing. The class ignores any calls to `CreateNewList()` while this output is TRUE. |
| Error | BOOL | TRUE if the directory listing could not be created. |
| ErrorDesc | STRING(255) | The last error encountered is described here. |
| NewListReady | BOOL | Once a list has been built, this output is set to TRUE. |

## CreateNewList (Method)

This method may be called each time a new listing of event reports is required. It filters by device name only.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| deviceName | STRING(32) | If not blank, only events from this device will be listed. |

### Processing

- ➤ Clears *Error*.
- ➤ Sets the *NewListReady* value to FALSE and destroys any internal lists.
- ➤ Initiates the enumeration of the directory, carried out by the `Run()` method.

## CreateNewFilteredList (Method)

This method may be called each time a new listing of event reports is required. It filters by device name, time of creation, the reporting protocol, and the event type.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| deviceName | STRING(32) | If not blank, only events from this device will be listed. |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| startTime | DT | The earliest time stamp of a returned event as seconds since epoch. |
| endTime | DT | The latest time stamp of a returned event as seconds since epoch. |
| protocol | Enum_protocol_id | The protocol that collected the events. |
| eventType | Enum_event_type | The type of the events to be presented. |

### Processing

- ➤ Clears *Error*.
- ➤ Sets the *NewListReady* value to FALSE and destroys any internal lists.
- ➤ Initiates the enumeration of the events, carried out by the `Run()` method.
- ➤ Values of NO_PROTOCOL_SPECIFIED, NO_EVENT_TYPE, or zero for *startTime* and *endTime* result in the associated filter not being used.

## GetList (Method)

The call to this method must occur after the NewListReady output is TRUE to obtain the vector of event handles. There can only be one call to this method per created list.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| list | DynamicVectors.class_BaseVector | The vector to write the directory listing to. This vector must have been initialized with an element size SIZEOF(struct_EventDetails).<br><br>See the DynamicVectors library for more information about the type class_BaseVector. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the class_BaseVector provided was populated successfully. |

### Processing

- ➤ Returns FALSE if *NewListReady* is not TRUE.
- ➤ Populates *list* with the prepared list.
- ➤ Sets the *NewListReady* class output to FALSE.
- ➤ Destroys the internal list.

## Run (Method)

Call this method on every scan. It supervises the asynchronous event report listing.

### Processing

- ➤ If an event listing has been initiated:
  - ➢ Ensure that the listing is opened by using `sel_file.sel_begin_event_-iterator()`.
  - ➢ Repeatedly call `sel_file.sel_next_event()` and append a struct_EventDetails to *list* for each of the returned files that were issued by *deviceName*, the system returns no more files.
- ➤ Once the directory listing is complete, it closes the operation by setting *NewListReady* to TRUE.
- ➤ If any error occurs, *Error* is set to TRUE and *ErrorDesc* is populated appropriately.

## class_FileWriter (Class)

This class provides the ability to write files to the sequestered RTAC file system. This class is instantiated with a specific file name. The return value of each method is based on the success or failure of queuing the requested action. The final success or failure of each action is not determined until processing completes after multiple calls to the `Run()` method, which you must call every scan to perform whatever file-handling actions are buffered in its internal queue.

The g_p_FileIo_MaxBufferSize parameter, detailed in *Global Parameters on page 2*, dictates the maximum amount of data that can be buffered at one time before writing to the specified file.

### Initialization Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filename | STRING(255) | The full path of the file opened in append mode. The character "/" delimits the folder path. This path must end with the full file name, including extension. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). If the file does not exist, it will be created. |

### Properties

| Name | IEC 61131 Type | Access | Description |
|---|---|---|---|
| BytesLeft | UDINT | R | Number of unwritten bytes in the internal buffer. |
| Filename | STRING(255) | R/W | Write to this property to set the next file to which data are written. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../).<br><br>Writing to this property sets the *FileRename* output true. If data are appended to this class while *FileRename* is TRUE, subsequent attempts to set *Filename* are ignored until *FileRename* returns to FALSE.<br><br>After modifying *Filename*, any append method call queues data for the new file. |

Properties are internal values made visible through Get and Set accessors. Access is defined as R (read), W (write), or R/W (read/write).

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| Error | BOOL | TRUE if the function block cannot write the contents of its buffer to file. |
| ErrorDesc | STRING(255) | The last error encountered will be described here. |
| FileRename | BOOL | After the *Filename* property is set, this pin will remain TRUE until all pending writes to the previous file name have been completed. |

## AppendBytes (Method)

Call this method whenever bytes are to be appended to the write buffer. Every subsequent call of the `Run()` method will write as many bytes as possible until nothing remains in the write buffer.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the item to write to file, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to write, starting with *pt_data*. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE for successful addition of the data to the output buffer. |

### Processing

- ➤ Check that *numBytes* exceeds 0.
- ➤ Check that the memory region specified has read access.
- ➤ If both previous statements are true, copy the contents of the specified region into the output buffer.
- ➤ If the copy succeeded, return TRUE.
- ➤ If any error occurs, the method sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## AppendSELString (Method)

Call this method to append the content of a class_SELString to the write buffer.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to append. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *strSel* was successfully added to the output buffer. |

### Processing

- ➤ Copy the content of the supplied string to the output buffer.
- ➤ If the copy succeeded, return TRUE.
- ➤ If any error occurs, the method sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## AppendString (Method)

Call this method to append the content of a string to the write buffer.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to append. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *str* was successfully added to the output buffer. |

### Processing

- ➤ Copy the content of the supplied string to the output buffer.
- ➤ If copy succeeded, return TRUE.
- ➤ If any error occurs, the method sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## AppendVector (Method)

Call this method to append the content of a vector to the write buffer.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The vector to append to the file. See the DynamicVectors library documentation for information about the I_Vector interface. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the vector was successfully added to the output buffer. |

### Processing

- ➤ Check that vector passed in is valid and has contents to copy.
- ➤ Copy the content of the dynamic vector into the output buffer.
- ➤ If the copy succeeded, return TRUE.
- ➤ If any error occurs, the method sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## Run (Method)

Call this method on every scan to supervise the asynchronous writing of the internal buffer to the specified file.

### Processing

- ➤ If the file is not open, this method opens it in append mode and stores the file handle internally.
- ➤ If the file is open and there are data in the internal buffer, this method writes to the opened file.
- ➤ Monitors the asynchronous write process, clears the buffer of written data, and subtracts number of bytes written from *BytesLeft*.
- ➤ If any error occurs, this method sets *Error* to TRUE and fills *ErrorDesc* appropriately.

## class_FileReader2 (Class)

This class provides the ability to read files from the sequestered RTAC file system. Call the `Run()` method every scan to perform whatever file-handling actions are buffered in the internal queue.

The g_p_FileIo_MaxBufferSize parameter, detailed in *Global Parameters on page 2*, dictates the maximum file size that can be read using this class.

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| InProgress | BOOL | Stays TRUE while the `Run()` method reads a file. The class ignores any calls to a read method while this output is TRUE. |
| Error | BOOL | TRUE if the function block cannot read contents of the file into buffer. |
| ErrorDesc | STRING(255) | The last error encountered is described here. |
| BytesInBuffer | UDINT | The number of bytes that were read from file. Set to 0 when a read method is called, and populated when read is complete. |

## ReadFile (Method)

Call this method to read the content of a file into the internal buffer.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| filename | STRING(255) | The full path to the file of interest within the sequestered file system. The character "/" delimits the folder path. This path must end with the full file name, including extension. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). |

### Processing

- ➤ Checks that a read operation is not in progress.
- ➤ Ensures that the first character of *filename* is "/". This method prepends the character if it is missing from the *filename* provided.
- ➤ Initiates a read operation, which `Run()` performs.

## ReadEventFromDB (Method)

Call this method to read the content of an event from the database into the internal buffer.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| handle | Struct_event_handle | The details required to request this event from the database. |
| fileType | Enum_event_data | The file extension to attempt to extract from this event. |

### Processing

- ➤ Checks that a read operation is not in progress.
- ➤ Initiates a read operation, which `Run()` performs.

## CopyTo (Method)

Copies the contents of the buffer to a user-accessible location.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| startByte | UDINT | Indicates the first byte to copy as an offset from the beginning of the internal buffer. |
| pt_byte | POINTER TO BYTE | The destination address to where the bytes should be copied. |
| numBytes | UDINT | The maximum number of bytes to write out, starting with *startByte*. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| UDINT | Returns the number of bytes copied to the destination address. |

### Processing

- ➤ Checks that *startByte* is less than *BytesInBuffer* and that *pt_byte* is a valid pointer with write access. If the initial checks fail, return 0.
- ➤ Copies contents of internal buffer to destination until all remaining bytes in buffer have been copied or *numBytes* specified have been copied.
- ➤ Returns the number of bytes copied.

## AppendToSELString (Method)

Copies the contents of the internal buffer to a class_SELString.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| startByte | UDINT | Indicates the first byte to copy as an offset from the beginning of the internal buffer. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to which the contents of the internal buffer will be appended. See the SELString library documentation for information about the class_SELString type. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| UDINT | Returns the number of characters added to the SELString. |

### Processing

➤ Checks that *startByte* is less than *BytesInBuffer*. If the initial check fails, returns 0.

➤ Beginning with *startByte*, appends the bytes from the internal buffer to the *strSel* supplied, until one of the following occurs:

1. The class_SELString throws an internal error.
2. No bytes remain in the buffer.

➤ Returns the number of characters added to *strSel*.

## CopyToString (Method)

Copies the content of the internal buffer to a string.

**NOTE:** This method assumes that the string str is of type STRING(255). Smaller strings will cause undesired behavior.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| startByte | UDINT | Indicates the first byte to copy as an offset from the beginning of the internal buffer. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to which the content of the internal buffer will be written. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| UDINT | Returns the number of characters added to the string. |

### Processing

➤ Checks that *startByte* is less than *BytesInBuffer*. If the initial check fails, returns 0.

➤ Copies the bytes from the internal buffer to *str* until either of the following occurs:

1. Two hundred and fifty-five (255) characters have been copied.
2. No bytes remain in the buffer.

➤ Appends a null terminator onto the *str*.

## AppendToVector (Method)

Allows the content of the buffer to be copied to the end of a user-accessible vector.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| startByte | UDINT | Indicates the first byte to copy as an offset from the beginning of the internal buffer. |
| vector | I_Vector | The vector to which the internal buffer content is appended. See the DynamicVectors library documentation for information about the I_Vector interface. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| UDINT | Returns the number of elements added to the vector. |

### Processing

- ➤ Checks that *startByte* is less than *BytesInBuffer*. If the initial check fails, return 0.
- ➤ Pushes the contents of the buffer into the supplied *vector*.
- ➤ If the number of bytes specified for the copied output (*BytesInBuffer* minus *startByte*) is not evenly divisible by the vector *ElementSize*, then pads the last element appended to *vector* with trailing zeros.

## Run (Method)

Call this method on every scan to supervise the asynchronous reading of the specified file into the internal buffer.

### Processing

- ➤ Waits until the initiation of a file read operation by the `ReadFile()` or `ReadEventReport()` methods.
- ➤ If the file is not open, opens the file in read mode and stores the file handle internally.
- ➤ If the file is open and a read has been signaled by the `ReadFile()` or `ReadEventReport` methods, monitors the asynchronous task until complete.
- ➤ Populates *BytesInBuffer* upon completion of the read.
- ➤ If any error occurs, this method sets *Error* TRUE and fills *ErrorDesc* appropriately.

## class_FileReader (Class)

This is a deprecated class that is now an exact copy of class_FileReader2. The ability to view event files by name only has been removed from the file system. Projects that contain the ReadEventReport method will now generate compile-time errors. Reading files should be accomplished using full paths or Struct_event_handle objects.

The g_p_FileIo_MaxBufferSize parameter, detailed in *Global Parameters on page 2*, dictates the maximum file size that can be read using this class.

If you use this class, consider refactoring to use class_FileReader2.

## class_BasicDirectoryManager (Class)

This class manages files within a given directory by removing files based on the size of the directory, the number of files in the directory, or the maximum number of days to hold a file since modified.

This class does not do the following:

➤ Directly write any files.

➤ Modify any files.

➤ Monitor files within a subdirectory.

Before you can use class_BasicDirectoryManager to manage a directory, it must be provided the folder path to monitor, a maximum size for that directory, and either a maximum number of files to hold or a maximum number of days for which to hold files.

### File Blacklisting

File blacklisting allows for files to be ignored by the class_BasicDirectoryManager. A blacklisted file cannot be deleted, and it is not counted in the total directory size or number of files.

A file is blacklisted by having a period (.) as the first character in the file name.

For example, a file named ".somefile.txt" is ignored by class_BasicDirectoryManager, while a file named "MySpecialFileData.txt" is managed by class_BasicDirectoryManager.

### File Rotation

Periodically, the class compares the new directory size against the maximum permitted directory size set in the object declaration. If the directory exceeds the maximum folder size, the oldest file is deleted. If only one file exists, no files are deleted. This allows the single file to overfill the allotted maximum until the creation of a new file. This means that if the maximum number of files is set to one, the manager never deletes files based on the directory size or age of the file.

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| Directory | STRING(128) | The directory being managed. |
| Error | BOOL | TRUE if the class encounters any error. |
| ErrorDesc | STRING(255) | The last error encountered by this class. |
| SpaceUsed | ULINT | The size, in bytes, of all managed files in this directory. |
| MaxFolderSize | ULINT | The size, in bytes, at which the directory begins deleting files, starting at the oldest. |
| MaxFileCount | UDINT | The maximum number of files this directory stores. A value of zero indicates that MaxFileCount is ignored. |
| MaxDaysHeld | UDINT | The maximum number of days this directory stores files based on the time stamp. A value of zero indicates that MaxDaysHeld is ignored. |

## bootstrap_SetDirectory (Method)

This method is called once, before any other method, to configure the class_BasicDirectory-Manager. It provides the values the class uses to determine what directory to watch and when to delete files.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| folderName | STRING(127) | The folder to use and manage. The character “/” delimits the folder path. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). If the folder does not exist, the class will show an error until the directory is created by some other mechanism. |
| maximumFolderSize | ULINT | The size, in bytes, at which the directory begins deleting files, starting at the oldest. |
| maximumNumFiles | UDINT | The maximum number of files this directory stores. A value of zero indicates that maximumNumFiles is ignored. |
| maximumNumDays | UDINT | The maximum number of days this directory stores files based on the time stamp. A value of zero indicates that maximumNumDays is ignored. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if no errors occured during bootstrapping. |

## Run (Method)

Call this method on every scan. It supervises the asynchronous deletion of old files. Deletions occur only if the number of files in the directory, the size of the directory, or the number of days to hold a file exceeds user-set limits.

### Processing

This class maintains an internal state machine with a round-robin job scheduler, ensuring that the amount of processing overhead per scan remains relatively constant.

Subsequent calls to the `Run()` method perform the following sequence of operations:

1. If the directory listing is exhausted:
   a) Determine the cutoff file for deletions on the next scan by performing the following steps:
      i. Collect a running total of space moving backward in time.
      ii. Find the file that causes the space to be exceeded and store its time stamp.
      iii. Find the file that exceeds the file count and store its time stamp.
      iv. Set the cutoff time to the newest of the two saved time stamps.
   b) Restart the directory iterator.
2. If the directory listing is not exhausted, perform one of the following checks on the next file:
   a) If the file is blacklisted, ignore it.
   b) If the file is managed and newer than the cutoff time from the previous directory scan, leave it alone.
   c) If the file is managed and older than the cutoff time from the previous directory scan, delete it.

## class_DirectoryManager (Class)

This class allows for the creation of managed files, over time, in a controlled directory. It provides protection for the size of the directory, the number of files in the directory, and the size of those files.

Before you can use this class to manage a directory, you must provide it the folder path designating to where log files are written, a maximum size for that directory, a maximum number of files to allow in that directory, and a postfix to add to log files.

This class uses class_FileWriter objects to perform the writing of log files and event logs. See *class_FileWriter (Class) on page 25* for more detailed information about the limitations on the maximum size of log files or maximum number of buffered log entries.

### File Entries

File entries take exactly the data provided and append this information to the active file. No additional formatting is performed.

## Event Logs

In addition to log files, you may want to create a separate file that records information corresponding to some event, with custom formatting. These are referred to as “Event Logs,” and should not be confused with the “Event Records” relays generate, containing high-resolution waveforms. An event file is simply a custom log file written out to the managed directory, rotated with the  files (as described in *File Rotation on page 36*), and sent to the same FTP server (if set) for this manager object.

Event Logs are stored with the time stamp of when they were created. The format for these files is *YYYY-MM-DD-HH-MM_eventPostfix*, where *eventPostfix* is defined in the method call to write the file.

It is important to recognize that, because the file name does not include seconds, two events recorded within the same minute and defined with the same *eventPostfix* argument will cause the contents of the second event to be appended to the end of the first file.

## File Rotation

Periodically, the class compares the new directory size against the maximum permitted directory size set in the object declaration. If the directory exceeds the maximum folder size, the oldest file is deleted. If no other files exist except the active  file, no files are deleted. This allows the single active  file to overfill the allotted maximum until the creation of a new  file.

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| Directory | STRING(127) | The directory being managed. |
| ActiveFile | STRING(128) | The rotating file presently waiting for write requests. |
| Error | BOOL | TRUE if the class cannot write the contents of its buffer to file. |
| ErrorDesc | STRING(255) | The last error encountered by this class. |
| SpaceUsed | ULINT | The size, in bytes, of all managed files in this directory. |
| MaxFolderSize | ULINT | The size, in bytes, at which the directory begins deleting files, starting at the oldest. |
| MaxFileSize | UDINT | The size, in bytes, at which this class rotates its automatically generated files. |
| MaxFileCount | UDINT | The maximum number of files this directory stores. A value of zero indicates that MaxFileCount is ignored. |

## bootstrap_SetDirectory (Method)

This method is called once, before any other method, to configure the class_DirectoryManager. It provides the values the class uses to determine where to store files, what to call them, and when to create and delete them.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| folderName | STRING(127) | The folder to use and manage. The character "/" delimits the folder path. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). If the folder does not exist, it will be created the first time that a file is written. Any files in this directory that are not managed files will be deleted. |
| filenamePostfix | STRING(16) | A string that is added to the end of the timestamped file name on every file. |
| maximumFolderSize | UDINT | The size, in bytes, at which the directory manager begins deleting files, starting at the oldest. |
| maximumFileSize | UDINT | The size, in bytes, at which this class rotates its automatically generated files. |
| maximumNumFiles | UDINT | The maximum number of files this directory stores. A value of zero indicates that maximumNumFiles is ignored. |
| rollFileAtDay | BOOL | Close the working file each day at midnight and start a new file. |

## SetFtpServerForArchiving (Method)

Call this method once to specify a remote FTP server to which generated files are sent and how often the files should be sent.

Every FTP attempt generates a log file to assist with debugging (overwriting the previous log file if it exists). The file includes success notifications as well as errors the ftp client encounters (such as a bad username or password). View the following file, found at the root of the sequestered file system, via a web browser after attempting an FTP file transfer:

```
ftplog.txt
```

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| ftpServer | STRING(15) | The IP address of the FTP server being contacted. |
| remotePath | STRING(127) | The folder on the FTP server to where the local files are sent. |
| username | STRING(32) | The FTP username used to log into the server. This must contain only alphanumeric or underscore characters. |
| password | STRING(32) | The password associated with the FTP username used to log into the server. This may contain any printable ASCII characters, excluding the quote characters. |
| timeout | UDINT | The number of seconds for the FTP attempt to be run before it is canceled. Must be greater than 0. |
| schedule | enum_FtpSendSchedule | Specify when local files should be sent to the remote FTP server. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the arguments provided are within range. |

### Processing

- ➤ Validates the input strings and confirms that a valid IP address is provided.
- ➤ If the inputs provided are valid, sets internal variables so that the `Run()` method attempts to send files, and returns TRUE.
- ➤ If the inputs provided are invalid, returns FALSE.

## SetFileHeaderBytes (Method)

This method sets a block of text the class will place at the beginning of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *numBytes* of zero clears any existing header; new files will be started with the first data entry instead.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to use as the header block, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to store, beginning with pt_data. |

## SetFileHeaderSELString (Method)

This method sets a block of text the class will place at the beginning of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *strSel* of `Size` zero clears any existing header; new files will be started with the first data entry instead.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to use as the header block. |

## SetFileHeaderString (Method)

This method sets a block of text the class will place at the beginning of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *str* of LEN zero clears any existing header; new files will be started with the first data entry instead.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to use as the header block. |

## SetFileHeaderVector (Method)

This method sets a block of text the class will place at the beginning of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *vector* of `Size` zero clears any existing header; new files will be started with the first data entry instead.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The vector to use as the header block. See the DynamicVectors library documentation for information about the I_Vector interface. |

## SetFileFooterBytes (Method)

This method sets a block of text the class will place at the end of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *numBytes* of zero clears any existing footer.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to use as the footer block, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to store, beginning with pt_data. |

## SetFileFooterSELString (Method)

This method sets a block of text the class will place at the end of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *strSel* of `Size` zero clears any existing footer.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to use as the footer block. |

## SetFileFooterString (Method)

This method sets a block of text the class will place at the end of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *str* of LEN zero clears any existing footer.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to use as the footer block. |

## SetFileFooterVector (Method)

This method sets a block of text the class will place at the end of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *vector* of `Size` clears any existing footer.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The vector to use as the new footer block. See the DynamicVectors library documentation for information about the I_Vector interface. |

## StartNewFile (Method)

Use this method to close the active file and begin a new one. Unless you call this method, a new file starts only if the conditions provided in `bootstrap_SetDirectory()` are met, (i.e., rollFileAtDay is TRUE and a new day has begun or the active file size exceeded maximumFileSize).

### Processing

- For an active log file and a non-empty footer string, this method places that string at the end of the active file.
- Closes the active file.
- Creates a new file with the present time stamp.
- For a non-empty header string, places that string at the top of the new file.

## WriteToFileBytes (Method)

Call this method to append a raw byte array to the active file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to write to file, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to write, beginning with pt_data. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *pt_data* was successfully added to the output buffer. |

### Processing

➤ Appends *numBytes* characters, starting at address *pt_data* to the output buffer.

➤ Does not append a newline to the output buffer.

➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteToFileSELString (Method)

Call this method to append an SELString to the active file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to append. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *strSel* was successfully added to the output buffer. |

### Processing

➤ Appends the content of *selStr* to the output buffer.

➤ Does not append a newline to the output buffer.

➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteToFileString (Method)

Call this method to append a string to the active file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to append to the file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *str* was successfully added to the output buffer. |

### Processing

- ➤ Appends the value of *str* to the output buffer.
- ➤ Does not append a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteToFileVector (Method)

Call this method to append a vector of data to the active file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The vector to append to the file. See the DynamicVectors library documentation for information about the I_Vector interface. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *vector* was successfully added to the output buffer. |

### Processing

- ➤ Appends the content of *vector* to the output buffer.
- ➤ Does not append a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## EventLogFromBytes (Method)

Call this method to write a log file with contents defined in a contiguous set of memory.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to write to file, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to write, beginning with pt_data. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

➤ Obtains the system time through the `SYS_TIME()` function call.

➤ Constructs the new file name from the time and *eventPostfix*.

➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.

➤ Passes the provided data to the internal class_FileWriter that handles event logs.

➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromSELString (Method)

Call this method to write a log file with contents defined in a class_SELString.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The content of the event file. |
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.
- ➤ Passes the provided data to the internal class_FileWriter that handles event logs.
- ➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromString (Method)

Call this method to write a log file with contents defined in a string.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The content of the event file. |
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.
- ➤ Passes the provided data to the internal class_FileWriter that handles event logs.
- ➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromVector (Method)

Call this method to write a log file with contents defined in an I_Vector.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The content of the event file. See the DynamicVectors library documentation for information about the I_Vector interface. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.
- ➤ Passes the provided data to the internal class_FileWriter that handles event logs.
- ➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## Run (Method)

Call this method on every scan. It supervises the asynchronous writing of queued data to active files and the asynchronous deletion of old files. Deletions occur only if the number of files in the directory or size of the directory exceed user-set limits.

This method is also responsible for sending local files to a remote FTP server if the user has configured FTP through a successful call to `SetFtpServerForArchiving()`.

### Processing

This class maintains an internal state machine with a round-robin job scheduler, ensuring that the amount of processing overhead per scan remains relatively constant.

Subsequent calls to the `Run()` method perform the following sequence of operations:

1. If the system day of year has changed since the last time `Run()` was called and the class is set to start a new file each day, the method starts a new log by calling the `StartNewFile()` method.
2. Is this object already in one of the states described in *Processing States on page 57*?
   - ➤ **Yes**: Continues execution of that state.
   - ➤ **No**: Evaluates the job priority list described in *Processing Jobs on page 57* and executes the next job.

3. Calls `Run()` on the internal class_FileWriter object that handles the writing of entries.
4. Calls `Run()` on the internal class_FileWriter object that handles the writing of event logs.

### Processing Jobs

Only one job is performed per call to this method. The jobs are listed below in priority order:

1. Enters the Send File state if a write operation has been completed since the last Send File state completed (determined by looking for the falling edge of class_-FileWriter.BytesLeft <> 0).
2. Enters the Directory Housekeeping state if there is no directory listing or the last listing was exhausted.
3. Enters the Resend File state if there are unsent files that have not been synchronized to the remote server .

### Processing States

Some of the jobs in *Processing Jobs on page 57* cause this object to enter a state. The following describes these states and their exit criteria:

➤ **Send File**: This state exits immediately if a valid FTP server was not provided using the method `SetFtpServerForArchiving()`.

If the FTP server was set appropriately, the behavior of this state varies depending on the value of the *schedule* argument passed in using the `SetFtpServerForArchiving()` method call. *Enumerations on page 2* defines the enumeration for this argument.

- ➢ **schedule = ON_CLOSE**: If this write was initiated by the `StartNewFile()` method, the closed file is sent to the FTP server using the `sel_ftp_client.ftp_-upload()` function call.

  The state is maintained until the file is sent and then successfully read back using the `sel_ftp.ftp_download()` function call.

  If any error occurs, this method sets *Error* to TRUE and fills *ErrorDesc* appropriately.
- ➢ **schedule = ON_UPDATE**: The active file is sent to the server using the method call `sel_ftp.ftp_upload()`.

➤ **Directory Housekeeping**: The following sub-states exist in this state.

- ➢ Obtain the size of the active file.
- ➢ If the active file size is greater than *maximumFileSize*, start a new file.

- If the file list is exhausted:
    1. Determine the cutoff file for deletions on the next scan by performing the following steps:
        a. Collect a running total of space moving backward in time.
        b. Find the file that causes space to be exceeded and store the time stamp of that file.
        c. Find the file that exceeds the file count moving backward in time and store its time stamp.
        d. Set the cutoff time to the newest of the two saved time stamps.
    2. Restart the directory iterator.
- If the directory listing is not exhausted, perform one of the following checks on the next file:
    - If unmanaged, delete it.
    - If the file is managed and newer than the cutoff time from the previous directory scan, leave it alone.
    - If the file is managed and older than the cutoff time from the previous directory scan, delete it.

## class_TimeBasedDirectoryManager (Class)

This class allows for the creation of managed files, over time, in a controlled directory. It provides protection for the size of the directory, the number of files in the directory, and the size of those files.

Before you can use this class to manage a directory, you must provide it the folder path designating to where log files are written, a maximum size for that directory, a maximum number of days for which to hold files, and a postfix to add to log files.

This class uses class_FileWriter objects to perform the writing of log files and event logs. See *class_FileWriter (Class) on page 25* for more detailed information about the limitations on the maximum size of log files or maximum number of buffered log entries.

### File Entries

File entries take exactly the data provided and append this information to the active file. No additional formatting is performed.

### Event Logs

In addition to log files, you may want to create a separate file that records information corresponding to some event, with custom formatting. These are referred to as “Event Logs,” and should not be confused with the “Event Records” relays generate, containing high-resolution waveforms. An event file is simply a custom log file written out to the managed directory, rotated with the  files (as described in *File Rotation on page 48*), and sent to the same FTP server (if set) for this manager object.

Event Logs are stored with the time stamp of when they were created. The format for these files is *YYYY-MM-DD-HH-MM_eventPostfix*, where *eventPostfix* is defined in the method call to write the file.

It is important to recognize that, because the file name does not include seconds, two events recorded within the same minute and defined with the same *eventPostfix* argument will cause the contents of the second event to be appended to the end of the first file.

## File Rotation

Periodically, the class compares the new directory size against the maximum permitted directory size set in the object declaration. If the directory exceeds the maximum folder size, the oldest file is deleted. If no other files exist except the active  file, no files are deleted. This allows the single active  file to overfill the allotted maximum until the creation of a new  file.

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| Directory | STRING(127) | The directory being managed. |
| ActiveFile | STRING(128) | The rotating file presently waiting for write requests. |
| Error | BOOL | TRUE if the class cannot write the contents of its buffer to file. |
| ErrorDesc | STRING(255) | The last error encountered by this class. |
| SpaceUsed | ULINT | The size, in bytes, of all managed files in this directory. |
| MaxFolderSize | ULINT | The size, in bytes, at which the directory begins deleting files, starting at the oldest. |
| MaxFileSize | UDINT | The size, in bytes, at which this class rotates its automatically generated files. |
| MaxDaysHeld | UDINT | The maximum number of days this directory stores files based on the time stamp. |

## bootstrap_SetDirectory (Method)

This method is called once, before any other method, to configure the class. It provides the values the class uses to determine where to store files, what to call them, and when to create and delete them.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| folderName | STRING(127) | The folder to use and manage. The character "/" delimits the folder path. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). If the folder does not exist, it will be created the first time that a file is written. Any files in this directory that are not managed files will be deleted. |
| filenamePostfix | STRING(16) | A string that is added to the end of the time-stamped file name on every file. |

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| maximumFolderSize | UDINT | The size, in bytes, at which the directory begins deleting files, starting at the oldest. |
| maximumFileSize | UDINT | The size, in bytes, at which this class rotates its automatically generated files. |
| maximumNumDays | UDINT | The maximum number of days from today this directory stores files based on the time stamp. |
| rollFileAtDay | BOOL | Close the working file each day at midnight and start a new file. |

## SetFileHeaderBytes (Method)

This method sets a block of text the class will place at the beginning of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *numBytes* of zero clears any existing header; new files will be started with the first data entry instead.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to use as the header block, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to store, beginning with *pt_data*. |

## SetFileHeaderSELString (Method)

This method sets a block of text the class will place at the beginning of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *strSel* of `Size` zero clears any existing header; new files will be started with the first data entry instead.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to use as the header block. |

## SetFileHeaderString (Method)

This method sets a block of text the class will place at the beginning of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *str* of LEN zero clears any existing header; new files will be started with the first data entry instead.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to use as the header block. |

## SetFileHeaderVector (Method)

This method sets a block of text the class will place at the beginning of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *vector* of `Size` zero clears any existing header; new files will be started with the first data entry instead.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The vector to use as the header block. See the DynamicVectors library documentation for information about the I_Vector interface. |

## SetFileFooterBytes (Method)

This method sets a block of text the class will place at the end of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *numBytes* of zero clears any existing footer.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to use as the footer block, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to store, beginning with *pt_data*. |

## SetFileFooterSELString (Method)

This method sets a block of text the class will place at the end of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *strSel* of `Size` zero clears any existing footer.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to use as the footer block. |

## SetFileFooterString (Method)

This method sets a block of text the class will place at the end of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *str* of LEN zero clears any existing footer.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to use as the footer block. |

## SetFileFooterVector (Method)

This method sets a block of text the class will place at the end of every non-event file it creates. If you desire a newline, you must include it in the provided data. A *vector* of `Size` clears any existing footer.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The vector to use as the new footer block. See the DynamicVectors library documentation for information about the I_Vector interface. |

## StartNewFile (Method)

Use this method to close the active file and begin a new one. Unless you call this method, a new file starts only if the conditions provided in `bootstrap_SetDirectory()` are met, (i.e., rollFileAtDay is TRUE and a new day has begun or the active file size exceeded maximumFileSize).

### Processing

- ➤ For an active log file and a non-empty footer string, this method places that string at the end of the active file.
- ➤ Closes the active file.
- ➤ Creates a new file with the present time stamp.
- ➤ For a non-empty header string, places that string at the top of the new file.

## WriteToFileBytes (Method)

Call this method to append a raw byte array to the active file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to write to file, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to write, beginning with pt_data. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *pt_data* was successfully added to the output buffer. |

### Processing

- ➤ Appends *numBytes* characters, starting at address *pt_data* to the output buffer.
- ➤ Does not append a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteToFileSELString (Method)

Call this method to append an SELString to the active file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to append. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *strSel* was successfully added to the output buffer. |

### Processing

- ➤ Appends the content of *selStr* to the output buffer.
- ➤ Does not append a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteToFileString (Method)

Call this method to append a string to the active file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to append to the file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *str* was successfully added to the output buffer. |

### Processing

- ➤ Appends the value of *str* to the output buffer.
- ➤ Does not append a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteToFileVector (Method)

Call this method to append a vector of data to the active file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The vector to append to the file. See the DynamicVectors library documentation for information about the I_Vector interface. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *vector* was successfully added to the output buffer. |

### Processing

- ➤ Appends the content of *vector* to the output buffer.
- ➤ Does not append a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## EventLogFromBytes (Method)

Call this method to write a log file with contents defined in a contiguous set of memory.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to write to file, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to write, starting with *pt_data*. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.
- ➤ Passes the provided data to the internal class_FileWriter that handles event logs.
- ➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromSELString (Method)

Call this method to write a log file with contents defined in a class_SELString.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The content of the event file. |
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.

➤ Passes the provided data to the internal class_FileWriter that handles event logs.

➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromString (Method)

Call this method to write a log file with contents defined in a string.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The content of the event file. |
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

➤ Obtains the system time through the `SYS_TIME()` function call.

➤ Constructs the new file name from the time and *eventPostfix*.

➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.

➤ Passes the provided data to the internal class_FileWriter that handles event logs.

➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromVector (Method)

Call this method to write a log file with contents defined in an I_Vector.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The content of the event file. See the DynamicVectors library documentation for information about the I_Vector interface. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

➤ Obtains the system time through the `SYS_TIME()` function call.

➤ Constructs the new file name from the time and *eventPostfix*.

➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.

➤ Passes the provided data to the internal class_FileWriter that handles event logs.

➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## Run (Method)

Call this method on every scan. It supervises the asynchronous writing of queued data to active files and the asynchronous deletion of old files. Deletions occur only if the number of files in the directory or size of the directory exceed user-set limits.

This method is also responsible for sending local files to a remote FTP server if the user has configured FTP through a successful call to `SetFtpServerForArchiving()`.

### Processing

This class maintains an internal state machine with a round-robin job scheduler, ensuring that the amount of processing overhead per scan remains relatively constant.

Subsequent calls to the `Run()` method perform the following sequence of operations:

1. If the system day of year has changed since the last time `Run()` was called and the class is set to start a new file each day, the method starts a new log by calling the `StartNewFile()` method.
2. Is this object already in one of the states described in *Processing States on page 57*?

   ➤ **Yes**: Continues execution of that state.

   ➤ **No**: Evaluates the job priority list described in *Processing Jobs on page 57* and executes the next job.

3. Calls `Run()` on the internal class_FileWriter object that handles the writing of entries.
4. Calls `Run()` on the internal class_FileWriter object that handles the writing of event logs.

## Processing Jobs

Only one job is performed per call to this method. The jobs are listed below in priority order:

1. Enters the Send File state if a write operation has been completed since the last Send File state completed (determined by looking for the falling edge of class_-FileWriter.BytesLeft <> 0).
2. Enters the Directory Housekeeping state if there is no directory listing or the last listing was exhausted.
3. Enters the Resend File state if there are unsent files that have not been synchronized to the remote server .

## Processing States

Some of the jobs in *Processing Jobs on page 57* cause this object to enter a state. The following describes these states and their exit criteria:

➤ **Send File**: This state exits immediately if a valid FTP server was not provided using the method `SetFtpServerForArchiving()`.

  If the FTP server was set appropriately, the behavior of this state varies depending on the value of the *schedule* argument passed in using the `SetFtpServerForArchiving()` method call. *Enumerations on page 2* defines the enumeration for this argument.

  ➢ **schedule = ON_CLOSE**: If this write was initiated by the `StartNewFile()` method, the closed file is sent to the FTP server using the `sel_ftp_client.ftp_-upload()` function call.

    The state is maintained until the file is sent and then successfully read back using the `sel_ftp.ftp_download()` function call.

    If any error occurs, this method sets *Error* to TRUE and fills *ErrorDesc* appropriately.

  ➢ **schedule = ON_UPDATE**: The active file is sent to the server using the method call `sel_ftp.ftp_upload()`.

➤ **Directory Housekeeping**: The following sub-states exist in this state.

  ➢ Obtain the size of the active file.

  ➢ If the active file size is greater than *maximumFileSize*, start a new file.

- ➢ If the file list is exhausted:
    1. Determine the cutoff file for deletions on the next scan by performing the following steps:
        a. Collect a running total of space moving backward in time.
        b. Find the file that causes space to be exceeded and store the time stamp of that file.
        c. Calculate and save the time stamp that exceeds the maxNumDays value.
        d. Set the cutoff time to the newest of the two saved time stamps.
    2. Restart the directory iterator.
- ➢ If the directory listing is not exhausted, perform one of the following checks on the next file:
    - ➢ If unmanaged, delete it.
    - ➢ If the file is managed and newer than the cutoff time from the previous directory scan, leave it alone.
    - ➢ If the file is managed and older than the cutoff time from the previous directory scan, delete it.

## class_LogDirectoryManager (Class)

This class allows for the creation of managed files, over time, in a controlled directory. It provides protection for the size of the directory, the number of files in the directory, and the size of those files.

Before you can use this class to manage a directory, you must provide it the folder path designating to where log files are written, a maximum size for that directory, a maximum number of files to allow in that directory, and a postfix to add to log files.

This class uses class_FileWriter objects to perform the writing of log files and event logs. See *class_FileWriter (Class) on page 25* for more detailed information about the limitations on the maximum size of log files or maximum number of buffered log entries.

### Log Entries

Log entries are prefixed with a time stamp and added as a single-row entry to the active log file.

The log file names are stored with the time stamp of when they were created. The format for these files is *YYYY-MM-DD-HH-MM_logPostfix*, where *logPostfix* is set in the declaration of the class.

## Event Logs

In addition to log files, you may want to create a separate file that records information corresponding to some event, with custom formatting. These are referred to as "Event Logs," and should not be confused with the "Event Records" relays generate, containing high-resolution waveforms. An event file is simply a custom log file written out to the managed directory, rotated with the log files (as described in *File Rotation on page 59*), and sent to the same FTP server (if set) for this manager object.

Event Logs are stored with the time stamp of when they were created. The format for these files is *YYYY-MM-DD-HH-MM_eventPostfix*, where *eventPostfix* is defined in the method call to write the file.

It is important to recognize that, because the file name does not include seconds, two events recorded within the same minute and defined with the same *eventPostfix* argument will cause the contents of the second event to be appended to the end of the first file.

## File Rotation

Periodically, the class compares the new directory size against the maximum permitted directory size set in the object declaration. If the directory exceeds the maximum folder size, the oldest file is deleted. If no other files exist except the active log file, no files are deleted. This allows the single active log file to overfill the allotted maximum until the creation of a new log file.

### Initialization Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| folderName | STRING(127) | The folder to write logs to and manage. The character "/" delimits the folder path. It may contain all printable ASCII characters between 16#20(Space) and 16#7E(~) except for ", ', :, <, %, >, ?, \, and \|. It cannot contain any file path manipulation variables (//, /./, /../). If the folder does not exist, it will be created the first time that a log is written. Any files in this directory that are not log files will be deleted. |
| logPostfix | STRING(16) | A string that is added to the end of the time-stamped file name on every log file. |
| maxFolderSize | UDINT | The size, in bytes, at which the directory begins deleting files, starting at the oldest. |
| maxNumFiles | UDINT | The maximum number of files this directory stores. A value of zero indicates that maxNumFiles is ignored. |
| autoStartNewLogDaily | BOOL | If this is TRUE, a new log file is automatically created on the first PLC scan of every day, regardless of whether an entry is written that day. If FALSE, a new log file will only be created at the first log entry method call. |

### Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| Directory | STRING(127) | The directory being managed. |
| ActiveFile | STRING(128) | The rotating file presently waiting for write requests. |
| Error | BOOL | TRUE if the class cannot write the contents of its buffer to file. |
| ErrorDesc | STRING(255) | The last error encountered by this class. |
| SpaceUsed | ULINT | The size, in bytes, of all managed files in this directory. |
| MaxFolderSize | ULINT | The size, in bytes, at which the directory begins deleting files, starting at the oldest. |
| MaxFileSize | UDINT | The size, in bytes, at which this class rotates its automatically generated files. |
| MaxFileCount | UDINT | The maximum number of files this directory stores. A value of zero indicates that MaxFileCount is ignored. |

## SetFtpServerForArchiving (Method)

Call this method once to specify a remote FTP server to which generated files are sent and how often the files should be sent.

Every FTP attempt generates a log file to assist with debugging (overwriting the previous log file if it exists). The file includes success notifications as well as errors the ftp client encounters (such as a bad username or password). View the following file, found at the root of the sequestered file system, via a web browser after attempting an FTP file transfer:

```
ftplog.txt
```

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| ftpServer | STRING(15) | The IP address of the FTP server being contacted. |
| remotePath | STRING(127) | The folder on the FTP server to where the local files are sent. |
| username | STRING(32) | The FTP username used to log into the server. This must contain only alphanumeric or underscore characters. |
| password | STRING(32) | The password associated with the FTP username used to log into the server. This may contain any printable ASCII characters, excluding the quote characters. |
| timeout | UDINT | The number of seconds for the FTP attempt to be run before it is canceled. Must be greater than 0. |
| schedule | enum_FtpSendSchedule | Specify when local files should be sent to the remote FTP server. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the arguments provided are within range. |

### Processing

- ➤ Validates the input strings and confirms that a valid IP address is provided.
- ➤ If the inputs provided are valid, sets internal variables so that the `Run()` method attempts to send files, and returns TRUE.
- ➤ If the inputs provided are invalid, returns FALSE.

## StartNewLog (Method)

Call this method to create a new log file. All new log entries are added to this file until this method is called again or the system day of year changes. Do not call this method if you desire exactly one log file per day.

### Processing

- ➤ If there was an active log file, adds a log entry with the text: `--End log file--`
- ➤ Updates internal retained variable storing the active log time via the `SYS_TIME()` function call.
- ➤ Adds a log to this new file with the text: `--Begin log file--`

## WriteLogEntryBytes (Method)

Call this method to append a raw byte array to the active log file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to write to file, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to write, beginning with pt_data. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *pt_data* was successfully added to the output buffer. |

### Processing

- ➤ Gets the current date from `SYS_TIME()`.
- ➤ Writes the time stamp of the log entry to the output buffer in the form *YYYY-MM-DD-HH-MM-SS.MiS*, where *MiS* is milliseconds.
- ➤ Appends *numBytes* characters starting at address *pt_data* to the output buffer.
- ➤ Appends a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteLogEntrySELString (Method)

Call this method to append an SELString to the active log file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The class_SELString to append. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *strSel* was successfully added to the output buffer. |

### Processing

- ➤ Gets the current date from `SYS_TIME()`.
- ➤ Writes the time stamp of the log entry to the output buffer in the form *YYYY-MM-DD-HH-MM-SS.MiS*, where *MiS* is milliseconds.
- ➤ Appends the content of *selStr* to the output buffer.
- ➤ Appends a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteLogEntryString (Method)

Call this method to append a string to the active log file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The string to add to the log. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *str* was successfully added to the output buffer. |

### Processing

- ➤ Gets the current date from `SYS_TIME()`.
- ➤ Writes the time stamp of the log entry to the output buffer in the form *YYYY-MM-DD-HH-MM-SS.MiS*, where *MiS* is milliseconds.

- ➤ Appends the value of *str* to the output buffer.
- ➤ Appends a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## WriteLogEntryVector (Method)

Call this method to append a vector of data to the active log file.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The vector to append to the file. See the DynamicVectors library documentation for information about the I_Vector interface. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the content of *vector* was successfully added to the output buffer. |

### Processing

- ➤ Gets the current date from `SYS_TIME()`.
- ➤ Writes the time stamp of the log entry to the output buffer in the form *YYYY-MM-DD-HH-MM-SS.MiS*, where *MiS* is milliseconds.
- ➤ Appends the content of *vector* to the output buffer.
- ➤ Appends a newline to the output buffer.
- ➤ If any error occurs, sets *Error* TRUE and populates *ErrorDesc* appropriately.

## EventLogFromBytes (Method)

Call this method to write a log file with contents defined in a contiguous set of memory.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| pt_data | POINTER TO BYTE | The address of the bytes to write to file, as returned by the `ADR()` function. |
| numBytes | UDINT | The number of bytes to write, starting with *pt_data*. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.
- ➤ Passes the provided data to the internal class_FileWriter that handles event logs.
- ➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromSELString (Method)

Call this method to write a log file with contents defined in a class_SELString.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| strSel | class_SELString | The content of the event file. |
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.

- ➤ Passes the provided data to the internal class_FileWriter that handles event logs.
- ➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromString (Method)

Call this method to write a log file with contents defined in a string.

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| str | STRING(255) | The content of the event file. |
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.
- ➤ Passes the provided data to the internal class_FileWriter that handles event logs.
- ➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## EventLogFromVector (Method)

Call this method to write a log file with contents defined in an I_Vector.

### Inputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| vector | I_Vector | The content of the event file. See the DynamicVectors library documentation for information about the I_Vector interface. |

### Inputs/Outputs

| Name | IEC 61131 Type | Description |
|---|---|---|
| eventPostfix | STRING(16) | A string that is added to the end of the time-stamped file name of an event log file. |

### Return Value

| IEC 61131 Type | Description |
|---|---|
| BOOL | Returns TRUE if the data are successfully added to the output buffer. |

### Processing

- ➤ Obtains the system time through the `SYS_TIME()` function call.
- ➤ Constructs the new file name from the time and *eventPostfix*.
- ➤ Sets the internal class_FileWriter object that handles event logs to use the new file name.
- ➤ Passes the provided data to the internal class_FileWriter that handles event logs.
- ➤ If any error occurs, sets *Error* to TRUE, populates *ErrorDesc* appropriately, and returns FALSE.

## Run (Method)

Call this method on every scan. It supervises the asynchronous writing of queued data to active files and the asynchronous deletion of old files. Deletions occur only if the number of files in the directory or size of the directory exceed user-set limits.

This method is also responsible for sending local files to a remote FTP server if the user has configured FTP through a successful call to `SetFtpServerForArchiving()`.

### Processing

This class maintains an internal state machine with a round-robin job scheduler, ensuring that the amount of processing overhead per scan remains relatively constant.

Subsequent calls to the `Run()` method perform the following sequence of operations:

1. If the system day of year has changed since the last time `Run()` was called, the method starts a new log by calling the `StartNewLog()` method.
2. Is this object already in one of the states described in *Processing States on page 67*?
   - ➤ **Yes**: Continues execution of that state.
   - ➤ **No**: Evaluates the job priority list described in *Processing Jobs on page 67* and executes the next job.
3. Calls `Run()` on the internal class_FileWriter object that handles the writing of entries.
4. Calls `Run()` on the internal class_FileWriter object that handles the writing of event logs.

### Processing Jobs

Only one job is performed per call to this method. The jobs are listed below in priority order:

1. Enters the Send File state if a write operation has been completed since the last Send File state completed (determined by looking for the falling edge of class_-FileWriter.BytesLeft <> 0).
2. Enters the Directory Housekeeping state if there is no directory listing or the last listing was exhausted.
3. Enters the Resend File state if there are unsent files that have not been synchronized to the remote server .

### Processing States

Some of the jobs in *Processing Jobs on page 67* cause this object to enter a state. The following describes these states and their exit criteria:

➤ **Send File**: This state exits immediately if a valid FTP server was not provided using the method `SetFtpServerForArchiving()`.

If the FTP server was set appropriately, the behavior of this state varies depending on the value of the *schedule* argument passed in using the `SetFtpServerForArchiving()` method call. *Enumerations on page 2* defines the enumeration for this argument.

- ➢ **schedule = ON_CLOSE**: If this write was initiated by the `StartNewLog()` method, the closed file is sent to the FTP server using the `sel_ftp_client.ftp_-upload()` function call.

  The state is maintained until the file is sent and then successfully read back using the `sel_ftp.ftp_download()` function call.

  If any error occurs, this method sets *Error* to TRUE and fills *ErrorDesc* appropriately.

- ➢ **schedule = ON_UPDATE**: The active file is sent to the server using the method call `sel_ftp.ftp_upload()`.

➤ **Directory Housekeeping**: The following sub-states exist in this state.

- ➢ Obtain the size of the active file.
- ➢ If the active file size is greater than 1/3 of the *maxFolderSize*, start a new file.

- If the file list is exhausted:
    1. Determine the cutoff file for deletions on the next scan by performing the following steps:
        a. Collect a running total of space moving backward in time.
        b. Find the file that causes space to be exceeded and store the time stamp of that file.
        c. Find the file that exceeds the file count moving backward in time and store its time stamp.
        d. Set the cutoff time to the newest of the two saved time stamps.
    2. Restart the directory iterator.
- If the directory listing is not exhausted, perform one of the following checks on the next file:
    - If unmanaged, delete it.
    - If the file is managed and newer than the cutoff time from the previous directory scan, leave it alone.
    - If the file is managed and older than the cutoff time from the previous directory scan, delete it.

## Benchmarks

### Benchmark Platforms

The benchmarking tests recorded for this library are performed on the following platforms.

- SEL-3505
    - R136-V0 firmware
- SEL-3530
    - R136-V0 firmware
- SEL-3555
    - Dual-core Intel i7-3555LE processor
    - 4 GB ECC RAM
    - R136-V0 firmware

### Benchmark Test Descriptions

It is important to note that all computation in the sel_file and sel_ftp_client libraries is performed at a lower priority than any logic processing functionality. The time required to perform a given action is proportional to the RTAC processing burden. The only values this document records are the times for queuing that lower-priority work. The system performs the lower-priority work asynchronously, so the `Run()` method of each class must be called or the status variable passed to the functions must be monitored on every scan to supervise the asynchronous operations.

Calculation of each time is the average of 100 iterations of the described computation.

## class_DirectoryListing

### CreateNewList

The time necessary to request a new list when provided a 255-character path.

### GetList

The time necessary to copy a list containing 10 file names.

### Run

The time necessary to call `Run()` on each scan when there is directory work pending.

### Idle

The time necessary to call `Run()` on each scan when there is no directory work pending.

## class_EventListing

### CreateNewList

The time necessary to request a new list when provided a 255-character path.

### GetList

The time necessary to copy a list containing 10 file names.

### Run

The time necessary to call `Run()` on each scan when there is directory work pending.

### Run (Idle)

The time necessary to call `Run()` on each scan when there is no directory work pending.

## class_FileWriter

### AppendBytes

The time necessary to request 255 bytes be written via `AppendBytes()`.

### AppendVector

The time necessary to request 255 bytes be written via `AppendVector()`.

### AppendString

The time necessary to request 255 bytes be written via `AppendString()`.

### AppendSELString

The time necessary to request 255 bytes be written via `AppendSELString()`.

### Run

The time necessary to call `Run()` on the scan it switches from idle to work pending. This tests how long it takes to request a copy of 255 characters when switching from idle state.

### Run (Idle)

The time necessary to call `Run()` on each scan when there is no work pending.

## class_FileReader2

### ReadFile

The time necessary to request that a file with a 255-byte-long file name be opened via `ReadFile()`.

### CopyTo

The time necessary to copy 255 bytes from the internal buffer via `CopyTo()`.

### AppendToVector

The time necessary to append 255 bytes from the internal buffer by using `AppendToVector()`.

### CopyToString

The time necessary to copy 255 bytes from the internal buffer via `CopyToString()`.

### AppendToSELString

The time necessary to append 255 bytes from the internal buffer by using `AppendToSELString()`.

### Run

The time necessary to call `Run()` on a class_FileReader2 on the scan it switches from work pending to idle. This tests how long it takes to request a copy of 255 characters into the internal buffer when switching to idle state.

### Run (Idle)

The time necessary to call `Run()` on a class_FileReader each scan when there is no work pending.

## class_LogDirectoryManager

### StartNewLog

The time necessary to call `StartNewLog()`.

### WriteLogEntryBytes

The time necessary to call `WriteLogEntryBytes()` with 255 bytes of input.

### WriteLogEntryVector

The time necessary to call `WriteLogEntryVector()` with 255 bytes of content.

### WriteLogEntryString

The time necessary to call `WriteLogEntryString()` with 255 characters of content.

### WriteLogEntrySELString

The time necessary to call `WriteLogEntrySELString()` with 255 characters of content.

### EventLogFromBytes

The time necessary to call `EventLogFromBytes()` with 255 bytes of input.

### EventLogFromVector

The time necessary to call `EventLogFromVector()` with 255 bytes of input.

### EventLogFromString

The time necessary to call `EventLogFromString()` with 255 bytes of input.

### EventLogFromSELString

The time necessary to call `EventLogFromSELString()` with 255 bytes of input.

### Run

The time necessary to call `Run()` with FTP configured.

## class_DirectoryManager

### StartNewFile

The time necessary to call `StartNewLog()`.

#### WriteLogEntryBytes

The time necessary to call `WriteLogEntryBytes()` with 255 bytes of input.

#### WriteLogEntryVector

The time necessary to call `WriteLogEntryVector()` with 255 bytes of content.

#### WriteLogEntryString

The time necessary to call `WriteLogEntryString()` with 255 characters of content.

#### WriteLogEntrySELString

The time necessary to call `WriteLogEntrySELString()` with 255 characters of content.

#### EventLogFromBytes

The time necessary to call `EventLogFromBytes()` with 255 bytes of input.

#### EventLogFromVector

The time necessary to call `EventLogFromVector()` with 255 bytes of input.

#### EventLogFromString

The time necessary to call `EventLogFromString()` with 255 bytes of input.

#### EventLogFromSELString

The time necessary to call `EventLogFromSELString()` with 255 bytes of input.

#### Run

The time necessary to call `Run()` with FTP configured.

### class_TimeBasedDirectoryManager

#### StartNewFile

The time necessary to call `StartNewLog()`.

#### WriteLogEntryBytes

The time necessary to call `WriteLogEntryBytes()` with 255 bytes of input.

#### WriteLogEntryVector

The time necessary to call `WriteLogEntryVector()` with 255 bytes of content.

#### WriteLogEntryString

The time necessary to call `WriteLogEntryString()` with 255 characters of content.

#### WriteLogEntrySELString

The time necessary to call `WriteLogEntrySELString()` with 255 characters of content.

#### EventLogFromBytes

The time necessary to call `EventLogFromBytes()` with 255 bytes of input.

#### EventLogFromVector

The time necessary to call `EventLogFromVector()` with 255 bytes of input.

#### EventLogFromString

The time necessary to call `EventLogFromString()` with 255 bytes of input.

#### EventLogFromSELString

The time necessary to call `EventLogFromSELString()` with 255 bytes of input.

#### Run

The time necessary to call `Run()` with FTP configured.

### fun_FtpDownload

The time necessary to request a file download when provided a 255-character path.

### fun_FtpEventUpload

The time necessary to request an event upload when provided a 255-character path.

### fun_FtpUpload

The time necessary to request a file upload when provided a 255-character path.

### fun_DeleteFile

The time necessary to request a file delete when provided a 255-character path.

## fun_FileSize

The time necessary to request a file size when provided a 255-character path.

## fun_FilesystemFreeSpace

The time necessary to request the available free space on the system.

## fun_SoeAscending

The time necessary to request a list of 10 SOE reports without a filter.

## fun_SoeDescending

The time necessary to request a list of 10 SOE reports without a filter.

## fun_SoeWindow

The time necessary to request a list of 10 SOE reports without a filter.

## fun_LocalSoeGetID

The time necessary to request a list of 10 SOE reports without a filter.

## fun_LocalSoeAscending

The time necessary to request a list of 10 SOE reports without a filter.

## fun_LocalSoeDescending

The time necessary to request a list of 10 SOE reports without a filter.

## fun_RemoteSoeGetID

The time necessary to request a list of 10 SOE reports without a filter.

## fun_RemoteSoeAscending

The time necessary to request a list of 10 SOE reports without a filter.

### fun_RemoteSoeDescending

The time necessary to request a list of 10 SOE reports without a filter.

## Benchmark Results

| Operation Tested | Platform (time in $\mu s$) | | |
|---|---|---|---|
| | SEL-3505 | SEL-3530 | SEL-3555 |
| class_DirectoryListing.CreateNewList | 327 | 202 | 11 |
| class_DirectoryListing.GetList | 72 | 45 | 4 |
| class_DirectoryListing.Run | 137 | 42 | 3 |
| class_DirectoryListing.Run (Idle) | 5 | 4 | 1 |
| class_EventListing.CreateNewList | 6 | 4 | 1 |
| class_EventListing.GetNewList | 353 | 218 | 73 |
| class_EventListing.Run | 18 | 12 | 2 |
| class_EventListing.Run (Idle) | 2 | 1 | 1 |
| class_FileWriter.AppendBytes | 27 | 20 | 2 |
| class_FileWriter.AppendVector | 37 | 17 | 1 |
| class_FileWriter.AppendString | 43 | 26 | 2 |
| class_FileWriter.SELString | 1170 | 678 | 53 |
| class_FileWriter.Run | 62 | 50 | 5 |
| class_FileWriter.Run (Idle) | 5 | 4 | 1 |
| class_FileReader2.ReadFile | 25 | 14 | 1 |
| class_FileReader2.CopyTo | 38 | 19 | 1 |
| class_FileReader2.AppendToVector | 824 | 459 | 19 |
| class_FileReader2.AppendToString | 13 | 6 | 1 |
| class_FileReader2.AppendToSELString | 460 | 287 | 17 |
| class_FileReader2.Run | 4 | 2 | 1 |
| class_FileReader2.Run (Idle) | 3 | 1 | 1 |
| class_LogDirectoryManager.StartNewLog | 552 | 285 | 13 |
| class_LogDirectoryManager.WriteLogEntryBytes | 113 | 127 | 4 |
| class_LogDirectoryManager.WriteLogEntryVector | 159 | 100 | 4 |
| class_LogDirectoryManager.WriteLogEntryString | 143 | 71 | 5 |
| class_LogDirectoryManager.WriteLogEntrySELString | 1934 | 1030 | 45 |
| class_LogDirectoryManager.EventLogFromBytes | 145 | 91 | 4 |
| class_LogDirectoryManager.EventLogFromVector | 149 | 68 | 3 |
| class_LogDirectoryManager.EventLogFromString | 172 | 116 | 4 |
| class_LogDirectoryManager.EventLogFromSELString | 1927 | 1024 | 44 |
| class_LogDirectoryManager.Run | 624 | 368 | 14 |
| class_DirectoryManager.StartNewFile | 237 | 104 | 5 |
| class_DirectoryManager.WriteLogEntryBytes | 92 | 27 | 1 |
| class_DirectoryManager.WriteLogEntryVector | 41 | 15 | 1 |
| class_DirectoryManager.WriteLogEntryString | 46 | 30 | 2 |
| class_DirectoryManager.WriteLogEntrySELString | 1770 | 952 | 44 |
| class_DirectoryManager.EventLogFromBytes | 286 | 135 | 4 |
| class_DirectoryManager.EventLogFromVector | 220 | 85 | 4 |
| class_DirectoryManager.EventLogFromString | 158 | 80 | 4 |
| class_DirectoryManager.EventLogFromSELString | 1966 | 1020 | 43 |
| class_DirectoryManager.Run | 574 | 309 | 14 |

| **Operation Tested** | **Platform (time in $\mu s$)** | | |
|---|---|---|---|
| | **SEL-3505** | **SEL-3530** | **SEL-3555** |
| class_TimeBasedDirectoryManager.StartNewFile | 256 | 95 | 5 |
| class_TimeBasedDirectoryManager.WriteLogEntryBytes | 79 | 27 | 1 |
| class_TimeBasedDirectoryManager.WriteLogEntryVector | 46 | 17 | 1 |
| class_TimeBasedDirectoryManager.WriteLogEntryString | 47 | 21 | 2 |
| class_TimeBasedDirectoryManager.WriteLogEntrySELString | 1769 | 969 | 45 |
| class_TimeBasedDirectoryManager.EventLogFromBytes | 282 | 116 | 4 |
| class_TimeBasedDirectoryManager.EventLogFromVector | 236 | 84 | 3 |
| class_TimeBasedDirectoryManager.EventLogFromString | 165 | 82 | 4 |
| class_TimeBasedDirectoryManager.EventLogFromSELString | 198 | 1016 | 45 |
| class_TimeBasedDirectoryManager.Run | 592 | 348 | 16 |
| fun_FtpDownload | 112 | 76 | 4 |
| fun_FtpEventUpload | 75 | 45 | 5 |
| fun_FtpUpload | 88 | 88 | 5 |
| fun_DeleteFile | 73 | 50 | 4 |
| fun_FileSize | 93 | 53 | 5 |
| fun_FilesystemFreeSpace | 75 | 41 | 4 |
| fun_SoeAscending | 106 | 64 | 5 |
| fun_SoeDescending | 106 | 63 | 4 |
| fun_SoeWindow | 118 | 62 | 4 |
| fun_LocalSoeGetID | 122 | 60 | 5 |
| fun_LocalSoeAscending | 108 | 64 | 4 |
| fun_LocalSoeDescending | 118 | 65 | 4 |
| fun_RemoteSoeGetID | 113 | 61 | 4 |
| fun_RemoteSoeAscending | 116 | 65 | 5 |
| fun_RemoteSoeDescending | 106 | 63 | 4 |

# Examples

*These examples demonstrate the capabilities of this library. Do not mistake them as suggestions or recommendations from SEL.*

*Implement the best practices of your organization when using these libraries. As the user of this library, you are responsible for ensuring correct implementation and verifying that the project using these libraries performs as expected.*

## Writing From Changing String

### Objective

You want to append the contents of a string to a file every time the value of that string changes.

## Assumptions

This example assumes that a user-specified IEC 61131 function called fun_StringDifferent provides functionality similar to that in *Code Snippet 1*.

**Code Snippet 1 fun_StringsDifferent**

```
(* Compares str1 to str2. If they are identical until the null terminator
is encountered in both strings, then return FALSE. *)
FUNCTION fun_StringsDifferent : BOOL
VAR CONSTANT
    c_maxStringSize : UDINT := 255;
END_VAR
VAR_IN_OUT
    str1 : STRING(c_maxStringSize); // The first string to compare
    str2 : STRING(c_maxStringSize); // The second string to compare
END_VAR
VAR
    i : UDINT;
    differenceFound : BOOL := FALSE;
END_VAR
```

```
FOR i := 0 TO c_maxStringSize - 1 DO
    IF (0 = str1[i]) AND (0 = str2[i]) THEN
        // Found null terminator on both strings at the same time.
        EXIT;
    ELSIF str1[i] <> str2[i] THEN
        differenceFound := TRUE;
        EXIT;
    END_IF
END_FOR
(* Return True if difference found *)
fun_StringsDifferent := differenceFound;
```

## Solution

Because we assume that a simple string comparison function exists, we can write out the contents of a string to a file every time it changes by using just a few lines of code, as in *Code Snippet 2*.

**Code Snippet 2 prg_WriteStringOnChange**

```
PROGRAM prg_WriteStringOnChange
VAR
    TheStringToWrite : STRING(255) := '';
    TheStringLastWritten : STRING(255) := '';
    FileWriter : class_FileWriter('/OutputFolder/OutputFile.txt');
END_VAR
```

**Code Snippet 2 prg_WriteStringOnChange (Continued)**

```
IF fun_StringsDifferent(TheStringToWrite, TheStringLastWritten) THEN
    FileWriter.AppendString(TheStringToWrite);
    TheStringLastWritten := TheStringToWrite;
END_IF
FileWriter.Run(); // Run this every scan regardless.
```

## Reading File Contents Into Byte Array

### Objective

You want to read the contents of a file into a byte array.

### Assumptions

The file “/FileToRead.txt” exists in the root of the RTAC public file system.

### Solution

The file is read into an internal buffer and then copied into an empty user-supplied byte array using the program in *Code Snippet 3*.

**Code Snippet 3 prg_ReadToByteArray**

```
PROGRAM prg_ReadToByteArray
VAR CONSTANT
    c_ByteArraySize : UDINT := 10_000;
END_VAR
VAR
    TheByteArray : ARRAY[1..c_ByteArraySize] OF BYTE;
    Filename : STRING(255) := '/FileToRead.txt';
    FileReader : class_FileReader2;
    FirstScan : BOOL := TRUE;
    Copied : BOOL := FALSE;
END_VAR
```

```
IF FirstScan THEN
    //Initiate the File Read.
    FileReader.ReadFile(Filename);
    FirstScan := FALSE;
ELSIF 0 < FileReader.BytesInBuffer AND NOT Copied THEN
    FileReader.CopyTo(startByte := 0,
                      pt_byte := ADR(TheByteArray),
                      numBytes := c_ByteArraySize);
    Copied := TRUE;
END_IF
FileReader.Run(); // Run this every scan regardless.
```

## Reading File Contents Into Dynamic Byte Vector

### Objective

You want to read the contents of a file into a Dynamic Byte Vector.

### Assumptions

1. You have included the DynamicVectors library in your project.
2. The file "/FileToRead.txt" exists in the root of the RTAC public file system.

**NOTE:** See the DynamicVectors library documentation for more information about DynamicVectors. See the ACSELERATOR RTAC Library Extensions Instruction Manual (LibraryExtensionsIM) for explanation about the concepts used by the object-oriented extensions to the IEC 61131-3 standard.

### Solution

The file will first be read into an internal buffer and then copied into an empty user-supplied class_ByteVector using the program in *Code Snippet 4*.

**Code Snippet 4 prg_ReadToByteVector**

```
PROGRAM prg_ReadToByteVector
VAR
    TheByteVector : DynamicVectors.class_ByteVector;
    Filename : STRING(255) := '/FileToRead.txt';
    FileReader : class_FileReader2;
    FirstScan : BOOL := TRUE;
    Copied : BOOL := FALSE;
END_VAR
```

```
IF FirstScan THEN
    //Initiate the File Read.
    FileReader.ReadFile(Filename);
    FirstScan := FALSE;
ELSIF 0 < FileReader.BytesInBuffer AND NOT Copied THEN
    FileReader.AppendToVector(startByte := 0, vector := TheByteVector);
    Copied := TRUE;
END_IF
FileReader.Run(); // Run this every scan regardless.
```

## Reading File Contents Into Array of Strings

### Objective

You want to read the contents of a file into an array of strings.

### Assumptions

The file "/FileToRead.txt" exists in the root of the RTAC File Manager.

## Solution

The file will first be read into an internal buffer and then it will be copied into an empty user-supplied strings using the program in *Code Snippet 5*.

**Code Snippet 5 prg_ReadToArrayOfStrings**

```
PROGRAM prg_ReadToArrayOfStrings
VAR CONSTANT
   c_NumStringsInArray : UDINT := 1_000;
   c_StringSize : UDINT := 255;
END_VAR
VAR
    TheStringArray : ARRAY[1..c_NumStringsInArray] OF STRING(c_StringSize);
    Filename : STRING(255) := '/FileToRead.txt';
    FileReader : class_FileReader2;
    FirstScan : BOOL := TRUE;
    Copied : BOOL := FALSE;
    stringIter : UDINT;
    bufferTracker : UDINT := 0;
END_VAR
```

```
IF FirstScan THEN
    //Initiate the File Read.
    FileReader.ReadFile(Filename);
    FirstScan := FALSE;
ELSIF 0 < FileReader.BytesInBuffer AND NOT Copied THEN
    FOR stringIter := 1 TO c_NumStringsInArray DO
        IF bufferTracker <= FileReader.BytesInBuffer THEN
            FileReader.CopyToString(startByte := bufferTracker,
                             str := TheStringArray[stringIter]);
        ELSE
            // All of the file contents has been copied into strings.
            EXIT;
        END_IF
        bufferTracker := bufferTracker + c_StringSize;
    END_FOR
    Copied := TRUE;
END_IF
FileReader.Run(); // Run this every scan regardless.
```

## Reading Event Reports Retrieved From Relays

### Objective

You have set up the RTAC to automatically collect and buffer event records, and want to read the contents of these records.

### Assumptions

1. You have included the SELString and DynamicVectors libraries in your project.
2. The RTAC database contains events collected from the desired relays.

**NOTE:** See the SELString library documentation for more information about class_SELStrings and class_SELStringLists.

## Solution

You can locate qualifying files using a class_EventListing. Then you can select one as in *Code Snippet 6*.

**Code Snippet 6 prg_ReadEventReportFromRelay**

```
PROGRAM prg_ReadEventReportFromRelay
VAR
    EventReportListing : class_EventListing;
    EventReportList : class_BaseVector(SIZEOF(struct_EventDetails), 32);
    FileReader : class_FileReader2;
    FileContents : class_ByteVector;
    //A record of the first 255 characters of the read in file.
    FirstFileChars : STRING(255);
    EventReportFileDetails : struct_EventDetails;
    StepNumber : UDINT := 1; // Start off by running.
    EventIndexToRead : UDINT := 0; // The index of the file to read in
    EventIndexRead : UDINT := 0;
    Initiate : BOOL; // Force this value to TRUE in order to start reading
        files.
END_VAR
```

**Code Snippet 6 prg_ReadEventReportFromRelay (Continued)**

```
IF Initiate THEN
    // Start the state machine to read the event.
    StepNumber := 1;
    Initiate := FALSE;
END_IF
CASE StepNumber OF
1:
    // Request a list of all events on the system.
    EventReportListing.CreateNewList(deviceName := '');
    StepNumber := StepNumber + 1;
2:
    // Wait for the list to be ready.
    IF EventReportListing.NewListReady THEN
        IF EventReportListing.GetList(list := EventReportList) THEN
            StepNumber := StepNumber + 1;
        END_IF
    END_IF
3:
    // Select a specific event if that many events exist on the system.
    IF EventIndexToRead < EventReportList.Size THEN
        EventReportList.GetCopyOfElement(EventIndexToRead,
                                         ADR(EventReportFileDetails));
        EventIndexRead := EventIndexToRead;
        FileReader.ReadEventFromDB(EventReportFileDetails.Handle,
                                   FileIo.sel_file.RAW_DATA);
        StepNumber := StepNumber + 1;
    END_IF
4:
    // The file reader has read the data in. Do any required work.
    IF 0 < FileReader.BytesInBuffer THEN
        FileReader.AppendToVector(0, FileContents);
        (*Update the output string by copying to it the first 255 bytes or
          the complete file, whichever is less.*)
        SysMemCpy(pDest := ADR(FirstFileChars),
                  pSrc := FileContents.pt_Data,
                  udiCount := MIN(FileContents.Size, 255));
        StepNumber := StepNumber + 1;
    END_IF
5:
    // Wait for until Initiate = TRUE for next file read request.
    StepNumber := 0;
END_CASE
EventReportListing.Run(); // Run this every scan regardless.
FileReader.Run(); // Run this every scan regardless.
```

## Reading COMTRADE Events Retrieved From Relays

### Objective

You have set up the RTAC to automatically collect COMTRADE events and want to read the contents of these records.

## Assumptions

1. You have included the SELString and DynamicVectors libraries in your project.
2. Collected events from the desired relays exist in the RTAC database.

**NOTE:** See the SELString library documentation for more information about class_SELStrings and class_SELStringLists.

## Solution

You can locate qualifying files using a class_EventListing. Then you can select one as in *Code Snippet 7*.

**Code Snippet 7 prg_ReadComtradeEventFromRelay**

```
PROGRAM prg_ReadComtradeEventFromRelay
VAR
    EventReportListing : class_EventListing;
    EventReportList : class_BaseVector(SIZEOF(struct_EventDetails), 32);
    cfgReader : class_FileReader2;
    datReader : class_FileReader2;
    //A record of the first 255 characters of the read in file.
    EventReportFileDetails : struct_EventDetails;
    StepNumber : UDINT := 1; // Start off by running.
    EventIndexToRead : UDINT := 0;
    Initiate : BOOL; // Force this value to TRUE in order to start reading
        files.
END_VAR
```

**Code Snippet 7 prg_ReadComtradeEventFromRelay (Continued)**

```
IF Initiate THEN
    StepNumber := 1; // Start executing the state machine
    Initiate := FALSE;
END_IF
CASE StepNumber OF
1:
    EventReportListing.CreateNewList(deviceName := '');
    StepNumber := StepNumber + 1;
2:
    IF EventReportListing.NewListReady THEN
        IF EventReportListing.GetList(list := EventReportList) THEN
            StepNumber := StepNumber + 1;
        END_IF
    END_IF
    EventIndexToRead := 0;
3:
    WHILE EventIndexToRead < EventReportList.Size DO // An event was found
        EventReportList.GetCopyOfElement(EventIndexToRead,
        ADR(EventReportFileDetails));
        EventIndexToRead := EventIndexToRead + 1;
        IF EventReportFileDetails.Handle.EventType =
             FileIo.sel_file.COMTRADE THEN
            cfgReader.ReadEventFromDB(EventReportFileDetails.Handle,
            FileIo.sel_file.CFG_FILE);
            datReader.ReadEventFromDB(EventReportFileDetails.Handle,
            FileIo.sel_file.DAT_FILE);
            EXIT;
        END_IF
    END_WHILE
    StepNumber := StepNumber + 1;
4:
    IF NOT cfgReader.InProgress AND NOT datReader.InProgress THEN
        //Extract data and perform desired actions on the data here.
        //cfgReader and datReader contain the contents desired.
        StepNumber := StepNumber + 1;
    END_IF
5:
    IF EventIndexToRead >= EventReportList.Size THEN
        //This branch represents having accessed all COMTRADE files found.
        StepNumber := 0;
    ELSE
        //This branch represents having more files to check.
        StepNumber := 3;
    END_IF
END_CASE
EventReportListing.Run(); // Run this every scan regardless.
cfgReader.Run(); // Run this every scan regardless.
datReader.Run();
```

## Downloading File to Local File System From Remote FTP Server

### Objective

You want to read a file onto the local file system from a remote FTP server and call the local file "FileFromFtpServer.csv".

### Assumptions

1. An FTP server is set up, configured, and accessible by the RTAC over the network.
2. The FTP server configuration is as follows:
   - **IP address**: 192.168.0.2
   - **Username**: FTPUSER
   - **Password**: TAIL
3. The file "FileToFtp.csv" exists in the root of the FTP server file system.

### Solution

First, you must get the file from the remote server by performing an FTP download using code similar to that shown in *Code Snippet 8*.

Then you can manipulate the file at will. For example, you could read the file into an internal buffer, then copy it into an empty user-supplied class_ByteVector by using the program shown previously in *Code Snippet 4*.

**Code Snippet 8 prg_FtpDownload**

```
PROGRAM prg_FtpDownload
VAR
    FtpServerIP       : STRING(15)  := '192.168.0.2';
    FtpServerUsername : STRING(32)  := 'FTPUSER';
    FtpServerPassword : STRING(32)  := 'TAIL';
    FtpServerFileToGet : STRING(255) := 'FileToFtp.csv';
    RenameAsLocalFile : STRING(255)  := '/FileFromFtpServer.csv';

    FirstScan : BOOL  := FALSE;
    Timeout : UDINT := 10; // Time in seconds
    DownloadStatus : FileIo.sel_ftp_client.Enum_sel_ftp_client_errors := 0;
    CurrentStatus : FileIo.sel_ftp_client.Enum_sel_ftp_client_errors;
    DownloadAttemptCompleted : BOOL  := FALSE;
    DownloadAttemptFailed : BOOL  := FALSE;
END_VAR
```

**Code Snippet 8 prg_FtpDownload (Continued)**

```
CurrentStatus := DownloadStatus;
IF FirstScan THEN
    //Initiate the FTP Read.
    FileIo.sel_ftp_client.ftp_download(
            ftp_server := FtpServerIP,
            local_path := RenameAsLocalFile,
            remote_path := FtpServerFileToGet,
            username := FtpServerUsername,
            password := FtpServerPassword,
            timeout := Timeout,
            status := DownloadStatus); // This is passed in as a VAR_IN_OUT
    (* Note, making this call will cause the download status to be
        initialized
    to 'IN_PROGRESS'*)
    FirstScan := FALSE;

(* Because DownloadStatus was passed in as a VAR_IN_OUT,
it can be written to by the external FTP task.
Check it regularly to see if the download status changed to 0 *)
ELSIF FileIo.sel_ftp_client.NO_ERROR = CurrentStatus THEN
    DownloadAttemptCompleted := TRUE;
ELSIF CurrentStatus < FileIo.sel_ftp_client.IN_PROGRESS THEN
    // The operation has hit an error because NO_ERROR was already checked.
    DownloadAttemptFailed := TRUE;
END_IF
```

## Uploading File From Local File System to Remote FTP Server

### Objective

You want to write a file from the local file system to a remote FTP server and call the local file “FileFromRTAC.csv”.

### Assumptions

1. An FTP server is set up, configured, and accessible by the RTAC over the network.
2. The FTP server configuration is as follows:
   - **IP address**: 192.168.0.2
   - **Username**: FTPUSER
   - **Password**: TAIL
3. The file “FileToSend.csv” exists in the root of the RTAC File Manager.

### Solution

Get the file onto the remote server by performing an FTP upload.

**Code Snippet 9 prg_FtpUpload**

```
PROGRAM prg_FtpUpload
VAR
    FtpServerIP           : STRING(15)  := '192.168.0.2';
    FtpServerUsername     : STRING(32)  := 'FTPUSER';
    FtpServerPassword     : STRING(32)  := 'TAIL';
    FileNameForFtpServer : STRING(255)  := 'FileFromRTAC.csv';
    LocalFileToSend       : STRING(255)  := '/FileToSend.csv';
    FirstScan             : BOOL  := TRUE;
    Timeout               : UDINT  := 10; // Time in seconds
    UploadStatus          : FileIo.sel_ftp_client.Enum_sel_ftp_client_errors
        := 0;
    CurrentStatus         : FileIo.sel_ftp_client.Enum_sel_ftp_client_errors;
    UploadAttemptCompleted : BOOL  := FALSE;
    UploadAttemptFailed  : BOOL  := FALSE;
END_VAR
```

```
CurrentStatus  := UploadStatus;
IF FirstScan THEN
    //Initiate the FTP write.
    FileIo.sel_ftp_client.ftp_upload(
           ftp_server  := FtpServerIP,
           local_path := LocalFileToSend,
           remote_path  := FileNameForFtpServer,
           username  := FtpServerUsername,
           password  := FtpServerPassword,
           timeout  := Timeout,
           status  := UploadStatus); // This is passed in as a VAR_IN_OUT
    (* Note, making this call will cause the upload status to be initialized
    to 'IN_PROGRESS'*)
    FirstScan  := FALSE;

    (* Because UploadStatus was passed in as a VAR_IN_OUT,
    it can be written to by the external FTP task.
    Check it regularly to see if the upload status changed to 0 *)
ELSIF FileIo.sel_ftp_client.NO_ERROR = CurrentStatus THEN
    UploadAttemptCompleted  := TRUE;
ELSIF CurrentStatus < FileIo.sel_ftp_client.IN_PROGRESS THEN
    // The operation has hit an error because NO_ERROR was already checked.
    UploadAttemptFailed := TRUE;
END_IF
```

## Basic Directory Management With Persistent Log Files

### Objective

Use the basic directory manager to implement a six-file circular buffer in a target directory that is being populated by an independent FileIO class_FileWriter instance. Write application errors to a persistent log file in the target directory so the log file is not subject to the circular buffer.

## Assumptions

1. A user-programmed application is writing files to a designated folder, Dir1, on the RTAC file system by using FileIO class_FileWriter.
2. A Global Variable List, GVL1, has been defined to contain variables pertinent to error tracking activities for the application. This example GVL is shown in *Code Snippet 10*.
3. The user-programmed application populates the variables in GVL1.

**Code Snippet 10 Error Tracking: GVL1**

```
VAR_GLOBAL
    //Flag indicating error condition on the given application
    g_ApplicationError : BOOL;
    //User-defined numeric error category
    g_ApplicationErrorCode : DINT;
    //Specific error message
    g_ApplicationErrorDescription : STRING(255);
END_VAR
```

## Solution

Instantiate a class_BasicDirectoryManager to fulfill the stated directory management objectives. Also use class_FileWriter to generate a persistent error log file. Recall that class_BasicDirectoryManager ignores files with file names beginning with a period (.).

**Code Snippet 11 prg_ManageComplexDirectory**

```
PROGRAM prg_ManageComplexDirectory
VAR
    FirstScan : BOOL := TRUE;
    FolderName : STRING := 'Dir1';
    ErrorFileWriter : FileIO.class_FileWriter(filename :=
        'Dir1/.ErrorLog.txt');
    Manager : FileIO.class_BasicDirectoryManager;
    ApplicationErrorTrigger : R_TRIG;
    TempLogString : STRING(255);
END_VAR
```

**Code Snippet 11 prg_ManageComplexDirectory (Continued)**

```
IF FirstScan THEN
    (*Initialize the basic directory manager
    by calling the bootstrap_SetDirectory() method.
    Limit target directory to 1MB, 6 files, and
    no limit on the age of the files.*)
    Manager.bootstrap_SetDirectory(folderName := FolderName,
        maximumFolderSize := 1024*1024, maximumNumFiles := 6,
            maximumNumDays := 0);
    FirstScan := FALSE;
END_IF

//Look for the rising edge of the error flag.
ApplicationErrorTrigger(CLK := GVL1.g_ApplicationError);

(*IF error detected, write current state of GVL1 variables
to the persistent log file, preceded by the current system time.*)
IF ApplicationErrorTrigger.Q THEN
    TempLogString :=
        CONCAT(DT_TO_STRING(System_Time_Control_POU.System_Time_DateAndTime),
        ' Application error code:');
    TempLogString := CONCAT(TempLogString,
         DINT_TO_STRING(GVL1.g_ApplicationErrorCode));
    TempLogString := CONCAT(TempLogString, '$nError message: ');
    TempLogString := CONCAT(TempLogString,
         GVL1.g_ApplicationErrorDescription);
    TempLogString := CONCAT(TempLogString, '$n$r');
    ErrorFileWriter.AppendString(TempLogString);
END_IF

//Run the persistent file writer
ErrorFileWriter.Run();
//Manage the target directory
Manager.Run();
```

## Logging a History of Inputs and Outputs

### Objective

You want to keep a week's worth of input value history, as well as outputs from a protection algorithm that has been implemented.

### Assumptions

There exist some set of inputs and outputs to the work being done. Here these are delineated by adding the prefix g_, indicating that they exist in a GVL as shown in *Code Snippet 12*.

**Code Snippet 12   Global Variable List**

```
VAR_GLOBAL
    g_TriggerOne : BOOL;
    g_TriggerTwo : BOOL;
    g_WorkingVoltage : REAL;
    g_WorkingCurrent : REAL;

    g_OutputOne : BOOL;
    g_OutputTwo : BOOL;
END_VAR
```

## Solution

You can instantiate a class_LogDirectoryManager to manage rotation of the logs you want.

**Code Snippet 13   prg_LogApplicationActions**

```
PROGRAM prg_LogApplicationActions
VAR
    LogManager : class_LogDirectoryManager( folderName := '/WeekOfLogs/',
                                            logPostfix := 'InVsOuts.log',
                                            maxFolderSize := 512000,
                                            maxNumFiles := 7,
                                            autoStartNewLogDaily := TRUE);
    WorkspaceString : STRING(255);
END_VAR
```

```
IF g_TriggerOne THEN
    WorkspaceString := CONCAT( 'Trigger One received with a input voltage
        of ',
                               REAL_TO_STRING(g_workingVoltage));
    LogManager.WriteLogEntryString(WorkspaceString);
END_IF
IF g_TriggerTwo THEN
    WorkspaceString := CONCAT( 'Trigger Two received with a input current
        of ',
                               REAL_TO_STRING(g_workingCurrent));
    LogManager.WriteLogEntryString(WorkspaceString);
END_IF

(*At this point the user calls the application doing work so the outputs
    update.*)

IF g_OutputOne THEN
    LogManager.WriteLogEntryString('Action One requested');
END_IF
IF g_OutputTwo THEN
    LogManager.WriteLogEntryString('Action Two requested');
END_IF

LogManager.Run();
```

## Rotating Logs More Frequently

### Objective

You want to keep a week's worth of logs with creation of a new log file every eight hours.

### Solution

You can instantiate a class_LogDirectoryManager to manage rotation of the logs you want. To do this, you must track the time and issue `StartNewLog()` on the eight-hour shift boundaries.

**Code Snippet 14 prg_RotatingLogs**

```
PROGRAM prg_RotatingLogs
VAR CONSTANT
    c_ShiftChange1 : UDINT := 1;
    c_ShiftChange2 : UDINT := 9;
    c_ShiftChange3 : UDINT := 17;
END_VAR
VAR
    //Note that maxNumFiles allows for three files and the automated
        nightly rollover.
    LogManager : class_LogDirectoryManager( folderName :=
        '/WeekOfShiftLogs/',
                                            logPostfix := 'Shift.log',
                                            maxFolderSize := 512000,
                                            maxNumFiles := 28,
                                            autoStartNewLogDaily := TRUE);

    FirstScan : BOOL := TRUE;

    PresentTime : timestamp_t;
    TimeOfDay : TIME_OF_DAY;

    PreviousHours : UDINT;
    PresentHours : UDINT;
END_VAR
```

**Code Snippet 14 prg_RotatingLogs (Continued)**

```
PresentTime := SYS_TIME();
TimeOfDay := DT_TO_TOD(PresentTime.value.dateTime);

PreviousHours := PresentHours;
//Divide by 1000 to remove milliseconds and 3600 to remove seconds and
    minutes.
PresentHours := TOD_TO_UDINT(TimeOfDay)/3600000;

IF FirstScan THEN
   PreviousHours := PresentHours;
   FirstScan := FALSE;
END_IF

IF PreviousHours < c_ShiftChange1 AND PresentHours >= c_ShiftChange1 THEN
   LogManager.StartNewLog();
ELSIF PreviousHours < c_ShiftChange2 AND PresentHours >= c_ShiftChange2
    THEN
   LogManager.StartNewLog();
ELSIF PreviousHours < c_ShiftChange3 AND PresentHours >= c_ShiftChange3
    THEN
   LogManager.StartNewLog();
END_IF

//Do any work and logging desired.

LogManager.Run();
```

## Logging Events Via FTP

### Objective

You want to record event logs on a remote server.

### Assumptions

1. An FTP server is set up, configured, and accessible by the RTAC over the network.
2. The FTP server configuration is as follows:
   - **IP address**: 192.168.0.2
   - **Username**: FTPUSER
   - **Password**: TAIL
3. There are external variables g_EventOccurred and g_EventDescription, driven by other code, that cause an event to be populated and sent.
4. There exists some set of inputs and outputs for the work being done. Here these are delineated by adding the prefix g_, indicating that they exist in a GVL as shown in *Code Snippet 15*.

**Code Snippet 15 Global Variable List**

```
VAR_GLOBAL
    g_EventOccurred : BOOL;
    g_EventDescription : STRING(255);
END_VAR
```

## Solution

You can instantiate a class_LogDirectoryManager to accept the data for transmission and to manage the storage required to facilitate the transaction.

**Code Snippet 16 prg_RemoteEventLogs**

```
PROGRAM prg_RemoteEventLogs
VAR
    LogManager : class_LogDirectoryManager( folderName := '/RemoteLogs/',
                                            logPostfix := '',
                                            maxFolderSize := 10240,
                                            maxNumFiles := 10,
                                            autoStartNewLogDaily := FALSE);

    EventPostfix : String(16) := 'RTAC1.event';
    ServerIP : STRING(15) := '192.168.0.2';
    RemotePath : STRING := '/RTAC1_Event_Files/';
    FtpUser : STRING := 'FTPUSER';
    FtpPassword : STRING := 'TAIL';

    FirstScan : BOOL := TRUE;
    Workbench : STRING(255);
END_VAR
```

```
IF FirstScan THEN
    LogManager.SetFtpServerForArchiving( ftpServer := ServerIP,
                                         remotePath := RemotePath,
                                         username := FtpUser,
                                         password := FtpPassword,
                                         timeout := 5,
                                         schedule := ON_UPDATE);
    FirstScan := FALSE;
END_IF

//Do any work and logging desired.

IF g_EventOccurred THEN
    LogManager.EventLogFromString( str := g_EventDescription,
                                   eventPostfix := EventPostfix);
    g_EventOccurred := FALSE;
END_IF
LogManager.Run();
```

## Iterating Over All SOEs

### Objective

You need to programmatically iterate over all SOEs from the RTAC and be sure that every SOE after a certain time is addressed.

### Assumptions

Your workload requires assurances that every SOE is seen and that duplication of responses to SOEs is unacceptable. In this use case, the order of the events matters less than ensuring that each event is seen and addressed. For this method to work, the SOEs must either be all from the local RTAC or from external devices being logged on the RTAC. If both types of SOEs exist, they would need to be handled independently.

### Solution

You periodically query the system for the next *c_MaxSoeCount* SOEs that have not yet been addressed.

**Code Snippet 17 prg_SoeIterator**

```
PROGRAM prg_SoeIterator
VAR CONSTANT
    c_MaxSoeCount : UINT := 10;
END_VAR
VAR
    // Variables to control the program flow
    GetFirstSOE : BOOL := TRUE;
    SoeQueried : BOOL := FALSE;
    DoWork : BOOL := FALSE;
    i : UINT;

    // Input Filters
    StartTime : DT := DT#2000-1-1-0:0:0;
    Filters : FileIo.sel_file.Struct_soe_filter;

    // Variables to store function output
    Status : FileIo.sel_file.Enum_sel_file_errors;
    Content : ARRAY [1..c_MaxSoeCount] OF
        FileIo.sel_file.Struct_soe_content_id;
    LastOutput : FileIo.sel_file.Struct_soe_content_id;
    Count : UINT;
END_VAR
```

```
IF GetFirstSOE THEN
    // We need to get a starting SOE.
    IF NOT SoeQueried THEN
        FileIo.fun_LocalSoeGetID(StartTime, Filters, Status, Content[1]);
        SoeQueried := TRUE;
    ELSE
        IF Status = FileIo.sel_file.SYSTEM_BUSY THEN
            // The system was too busy try again.
            SoeQueried := FALSE;
        ELSIF Status = FileIo.sel_file.NO_ERROR THEN
```

```
            // We got a result, switch to group queries
            GetFirstSoe := FALSE;
            SoeQueried := FALSE;
            LastOutput := Content[1];
            StartTime := Content[1].TimeStamp;
            // This function only ever returns one result.
            Count := 1;
            // Trigger processing for the SOE data returned
            DoWork := TRUE;
        ELSIF Status = FileIo.sel_file.OPERATION_FAILED THEN
            ;(* The database was unable to find any SOEs matching the
                filters provided. *)
        ELSIF Status <> FileIo.sel_file.IN_PROGRESS THEN
            ;(* If we arrive here configuration is bad and needs to be
                manually adjusted to continue. *)
        END_IF
    END_IF
ELSE
    IF NOT SoeQueried THEN
        // Beginning from the last SOE received, query for the next set of
            SOEs.
        FileIo.fun_LocalSoeAscending( ADR(content[1]), LastOutput.ID,
            c_MaxSoeCount,
                                      Filters, Status, Count);
        SoeQueried := TRUE;
    ELSE
        IF Status = FileIo.sel_file.SYSTEM_BUSY THEN
            // The system was too busy try again.
            SoeQueried := FALSE;
        ELSIF Status = FileIo.sel_file.NO_ERROR THEN
            DoWork := Count > 0;
            SoeQueried := FALSE;
            IF DoWork THEN
                // Store next lookup information if there is any.
                LastOutput := Content[Count];
                StartTime := Content[Count].TimeStamp;
            END_IF
        ELSIF Status <> FileIo.sel_file.IN_PROGRESS THEN
            (* If we arrive here configuration is probably OK, as we got
                results above.
               Something probably affected the ID we were using. Start
                    over. *)
            SoeQueried := FALSE;
            GetFirstSOE := TRUE;
        END_IF
    END_IF
END_IF

IF DoWork THEN
    // Process any new data.
    FOR i := 1 TO Count DO
        ;(* Insert your code here to do work on the SOEs encountered. *)
    END_FOR
    DoWork := FALSE;
END_IF
```

## Querying a Subset of SOEs

### Objective

You want to display the 10 most recent SOEs each minute.

### Assumptions

You have some code for presenting or communicating the SOE content at some other location. In this use case, the order of events is more important than hard guarantees of seeing each event occur.

### Solution

You periodically query the system for the most recent SOE data.

**Code Snippet 18 prg_SoeUpdater**

```
PROGRAM prg_SoeUpdater
VAR CONSTANT
    c_MaxSoeCount : UINT := 10;
END_VAR
VAR
    // This query has no filters, so leave all values as default empty
    // strings.
    Filters : FileIo.sel_file.Struct_soe_filter;
    Status : FileIo.sel_file.Enum_sel_file_errors;

    SoeData : ARRAY [1..c_MaxSoeCount] OF
        FileIo.sel_file.Struct_soe_content;
    SoesFound : UINT;

    Timestamp : timestamp_t;
    Now : DT;
    Last : DT;

    // These are the arrays populated with display data
    Devices : ARRAY [1..c_MaxSoeCount] OF STRING(255);
    Messages : ARRAY [1..c_MaxSoeCount] OF STRING(255);
    Times : ARRAY [1..c_MaxSoeCount] OF DT;

    // Flag indicating that SOE data has been requested and not copied.
    Running : BOOL := FALSE;

    i : UDINT;
END_VAR
```

**Code Snippet 18 prg_SoeUpdater (Continued)**

```
(*Check for the first run after the SOE query completes.*)
IF Running AND Status <> FileIo.sel_file.IN_PROGRESS THEN
    (*Loop across all found SOEs. This is guaranteed to be less than our
        array
     *sizes c_MaxSoeCount because of the arguments passed to the function
        below*)
    FOR i := 1 TO SoesFound DO
        Devices[i] := SoeData[i].DeviceName;
        Messages[i] := SoeData[i].Message;
        Times[i] := SoeData[i].TimeStamp;
    END_FOR
    Running := FALSE;
END_IF

Timestamp := Sys_Time();
Now := Timestamp.value.dateTime;

(*Only query for SOEs on even minute intervals*)
IF ((DT_TO_UDINT(Now) MOD 60) = 0) AND
        (*Only query for SOEs if any previous request has completed.*)
        Status <> FileIo.sel_file.IN_PROGRESS AND
        (*Only allow one query per second even if the last one completed.*)
        Now > Last THEN
    fun_SoeDescending(ADR(SoeData), Now, c_MaxSoeCount, Filters, Status,
        SoesFound);
    Running := TRUE;
    Last := Now;
END_IF
```

## Listing the Content of a Directory

### Objective

You want to list the the files contained in the `/TestDirectory`.

You want the listing to automatically occur when the project is uploaded. In addition, you want the ability to refresh the directory listing after the project is downloaded by forcing a value in the online editor.

### Assumptions

The directory content to be listed is uploaded.

### Solution

Create the file list using the FileIo.class_DirectoryListing, and write that content into an array to make it easier to see.

**Code Snippet 19 prg_ListDirectory**

```
PROGRAM prg_ListDirectory
VAR CONSTANT
    c_MaxFilesToList  : UDINT := 10;
        c_MaxFilenameLength : UDINT := 255;
END_VAR
VAR
    Lister          : FileIo.class_DirectoryListing;
    DirList         : FileIo.class_SELStringList;
    ArrayList       : ARRAY[1..c_MaxFilesToList] OF
        STRING(c_MaxFilenameLength);
        Stage                   : UDINT := 1; // Force to 1 with <Ctrl> <F6>
            to run again
END_VAR
VAR_TEMP
        i : UDINT;
        pt_SelStr : POINTER TO FileIo.class_SELString;
END_VAR
```

```
CASE Stage OF
1: // Clear from last run, and request a new list
    DirList.Clear();
    FOR i := 1 TO c_MaxFilesToList DO
        ArrayList[i] := ''; // Empty the array
    END_FOR
    Lister.CreateNewList(directoryName := '/TestDirectory',
                         filter := '');
    Stage := Stage + 1;
2: // Wait until done
    IF Lister.NewListReady THEN
        Lister.GetList(DirList);
        // Read the list into the array
        DirList.Begin(); // Start at the beginning of the list
        FOR i := 1 TO DirList.Size DO
            IF (i > c_MaxFilesToList) THEN
                EXIT; // No more room in the array
            END_IF
            pt_SelStr := DirList.Next();
            IF 0 <> pt_SelStr THEN // Always check pointers aren't 0
                ArrayList[i] := pt_SelStr^.ToString();
            END_IF
        END_FOR
        Stage := 0; // Reset to start
    END_IF
ELSE
    ; // Do nothing
END_CASE
Lister.Run(); // Always run the worker method
```

# Release Notes

| Version | Summary of Revisions | Date Code |
|---|---|---|
| 3.5.4.1 | ➤ Added facility to filter a directory listing to only return files newer than or equal to the date and time specified.<br>➤ Added new class_BasicDirectoryManager, which rotates the contents of a given directory based on maximum size constraints.<br>➤ Must be used with R144-V1 firmware or later. | 20190201 |
| 3.5.3.0 | ➤ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types "Cannot convert" messages.<br>➤ Must be used with R143 firmware or later.<br>➤ Increased default g_p_FileIo_MaxBufferSize size from 1 MB to 10 MB | 20180921 |
| 3.5.2.2 | ➤ Added note recommending not using FileIO with RTAC firmware versions R136-V0 and R136-V1.<br>➤ Added class_TimeBasedDirectoryManager to provide directory management capabilities for the user to keep files for a set number of days rather than a maximum number of files.<br>➤ Fixed an issue in class_DirectoryManger and class_LogDirectoryManager in which a system performing large quantities of FileIO tasks could result in early deletion of managed files.<br>➤ Added class_TimeBasedDirectoryManager to provide directory management capabilities for the user to keep files for a set number of days rather than a maximum number of files.<br>➤ Fixed an issue in class_DirectoryManger and class_LogDirectoryManager in which a system performing large quantities of FileIO tasks could result in early deletion of managed files.<br>➤ Rebranded the FileIo library to uppercase the "o" and be: "FileIO" (literature change only, no changes made to actual library name or namespaces). | 20161221 |
| 3.5.2.0 | ➤ Added class_DirectoryManager to provide directory management capabilities without the file content helps and constraints found in class_LogDirectoryManager.<br>➤ Added ability to write to directory manager files from vectors, byte arrays, and SELStrings in addition to the strings previously possible.<br>➤ Added functions to facilitate accessing SOEs monotonically in order of SOE creation.<br>➤ Added function to query for available file system free space.<br>➤ Modified file deletion algorithm for the directory manager classes to recover space faster in the case that the directory size is exceeded.<br>➤ Modified class_LogDirectoryManager to remove metadata for deleted files from the .unsent file both when FTP is configured and when it is not.<br>➤ Modified library to ensure that calling the `StartNewLog()` method on class_LogDirectoryManger multiple times during startup always appends the time-stamped file close message.<br>➤ Modified BytesLeft in class_FileWriter to show all pending work, where before it did not include bytes to be written to a new file name.<br>➤ Modified class_LogDirectoryManager to prevent a condition where dates before the year 2000 cause constant writing and rotating of files. | 20160610 |

| Version | Summary of Revisions | Date Code |
|---|---|---|
| | ➤ Modified class_Filewriter property *Filename* to no longer require file content before allowing *Filename* to be modified again.<br>➤ Removed deprecated features from class_FileReader because of loss of file system support. Use of these features now results in compilation errors.<br>➤ Removed deprecated class_EventReportListing because of loss of file system support. Use of this class now results in compilation errors. | |
| 3.5.1.0 | ➤ Added class_FileReader2 that replaces deprecated dependency with new sel_file features for accessing events.<br>➤ Added functions and documentation to wrapping underlying firmware API.<br>➤ Removed documentation of underlying firmware API.<br>➤ Added ability to read from the SOE database into the logic engine. | 20150930 |
| 3.5.0.10 | ➤ Made class_LogDirectoryManager able to delete files more quickly, facilitating a faster file creation rate and larger number of files.<br>➤ Fixed issue in class_FileWriter where changing *Filename* in quick succession caused the class to get stuck writing to a single file. | 20150717 |
| 3.5.0.9 | ➤ Fixed issue where an invalid *Filename* followed immediately by a valid *Filename* locked out class_FileWriter. | 20150213 |
| 3.5.0.7 | ➤ Added the *Filename* property to class_FileWriter.<br>➤ Added a new class_LogDirectoryManager. | 20141205 |
| 3.5.0.4 | ➤ Internal parts of the library are now hidden.<br>➤ Tested with sel_file V1.0.1.0 -released with R133 firmware (see RTAC firmware release notes for more details). | 20141008 |
| 3.5.0.3 | ➤ Initial release. | 20140812 |