ChannelMonitoring
IEC 61131 Library for ACSELERATOR RTAC® Projects

SEL Automation Controllers
Introduction

This library provides function blocks for performing data channel processing and supervision. The function blocks provide an alert that some aspect of a channel or indicator has deviated from the parameters defined by the user. Example applications include detecting maintenance conditions in a 3-phase CT/PT, alerting on an IED hardware failure, monitoring transformer through-fault current, or detecting protection communication channel failures.

The fb_MultiChannelAlert, fb_ChannelAlert, and fb_IndicatorAlert blocks focus on channel supervision. Each adheres to the same principles of operation. An alert is generated when a sustained excursion occurs or when repeated excursions are detected. An excursion is defined as a channel, indicator, or function block output exceeding the threshold limit. For function blocks that accept a Boolean data type input, an excursion begins with a transition from a FALSE to TRUE state. For function blocks that accept measured values (MV) or REAL data type inputs, the absolute difference is calculated between the instantaneous values of two channels or a channel and a reference value. An excursion in this context is when the absolute difference exceeds a threshold value. The excursion time is used to define when an alert occurs. If a single excursion is sustained for a length of time defined by the excursion time, an alert is generated (Figure 1 and Figure 2). If multiple excursions are detected equal to the chatter count within the excursion time, an alert is generated (Figure 3 and Figure 4).

Each function block can be used to provide simple alerting or can be combined into more complex monitoring schemes.

![Figure 1](image1.png)  
**Figure 1**  
An Excursion Defined by the Absolute Channel Difference Equaling or Exceeding the Threshold Value for the Excursion Time Generates an Alert
Special Considerations

> 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).

> 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.

```cpp
// This is bad and in most cases will provide a compiler error such as:
// "C0328: Assignment not allowed for type
class_fb_MultiChannelAlertObject"
myfb_MultiChannelAlertObject :=
    otherfb_MultiChannelAlertObject;
```
// This is fine
someVariable := myfb_MultiChannelAlertObject.value;
// As is this
pt_myfb_MultiChannelAlertObject :=
    ADR(myfb_MultiChannelAlertObject);

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.

### Supported Firmware Versions

You can use this library on any device configured using acSELERATOR RTAC® SEL-5033 Software with firmware version R143 or higher.

Versions 3.5.0.0 and older can be used on RTAC firmware version R132 and higher.

### Enumerations

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

#### enum_AlertType

This enumeration defines the type of events returned by the function block status output. This enumeration can be used interchangeably with DINT data types.

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_DEVIATION</td>
<td>0</td>
<td>No alerts detected</td>
</tr>
<tr>
<td>CHATTER</td>
<td>1</td>
<td>Multiple excursions occurred within the excursion time</td>
</tr>
<tr>
<td>EXPIRATION</td>
<td>2</td>
<td>A sustained excursion equaled or exceeded the excursion time</td>
</tr>
<tr>
<td>EXCURSION</td>
<td>3</td>
<td>An instantaneous excursion. Used where ExcursionTime input is not applicable.</td>
</tr>
<tr>
<td>BAD_QUALITY</td>
<td>4</td>
<td>Minimum number of inputs do not have good quality</td>
</tr>
<tr>
<td>RESET</td>
<td>5</td>
<td>Reset input is currently asserted</td>
</tr>
<tr>
<td>ERROR</td>
<td>6</td>
<td>Function block was unable to activate because of limited memory resources</td>
</tr>
<tr>
<td>COMPLETE</td>
<td>7</td>
<td>Operation complete</td>
</tr>
</tbody>
</table>
enum_ChannelAlert

This enumeration is used to define the channels responsible for a status alert and/or quality alert. This enumeration can be used interchangeably with DINT data types.

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_ALERTS</td>
<td>0</td>
<td>No alerts detected</td>
</tr>
<tr>
<td>CHANNEL_1_ALERT</td>
<td>1</td>
<td>Channel 1 is the responsible channel</td>
</tr>
<tr>
<td>CHANNEL_2_ALERT</td>
<td>2</td>
<td>Channel 2 is the responsible channel</td>
</tr>
<tr>
<td>CHANNEL_1_2_ALERT</td>
<td>3</td>
<td>Channel 1 and 2 are the responsible channels</td>
</tr>
<tr>
<td>CHANNEL_3_ALERT</td>
<td>4</td>
<td>Channel 3 is the responsible channel</td>
</tr>
<tr>
<td>CHANNEL_1_3_ALERT</td>
<td>5</td>
<td>Channel 1 and 3 are the responsible channels</td>
</tr>
<tr>
<td>CHANNEL_2_3_ALERT</td>
<td>6</td>
<td>Channel 2 and 3 are the responsible channels</td>
</tr>
<tr>
<td>MULTIPLE_CHANNEL_ALERT</td>
<td>7</td>
<td>All available channels are responsible</td>
</tr>
</tbody>
</table>

Functions

fun_GetAlertString

This function takes the status returned by the function blocks in this library as an input and returns a string value that can be used for logging.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alert</td>
<td>enum_AlertType</td>
<td>Function block status value</td>
</tr>
</tbody>
</table>

Return Value

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>Value matching the enum_AlertType</td>
</tr>
</tbody>
</table>

Processing

- If the status is valid, the function returns a string corresponding to the enum_AlertType.
- If the supplied status is not valid, the function returns Invalid Input.

fun_GetChannelString

This function takes as an input the alert returned by the fb_MultiChannel function block and returns a string value that can be used for logging.
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>enum_ChannelAlert</td>
<td>Function block alert value</td>
</tr>
</tbody>
</table>

**Return Value**

<table>
<thead>
<tr>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>String value matching the enum_ChannelAlert</td>
</tr>
</tbody>
</table>

**Processing**

- If `status` is valid, the function returns a string corresponding to the enum_ChannelAlert.
- If the supplied status is not valid, the function returns Invalid Input.

**Function Blocks**

**fb_MultiChannelAlert**

Compare two to three measured value (MV) tags to determine if one or more channels deviate outside a threshold value for a time period or if repeated deviations occur within a time period. This function block requires a minimum of two input channels.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enable the function block</td>
</tr>
<tr>
<td>Channel_1</td>
<td>MV</td>
<td>Data to monitor</td>
</tr>
<tr>
<td>Channel_2</td>
<td>MV</td>
<td>Data to monitor</td>
</tr>
<tr>
<td>Channel_3</td>
<td>MV</td>
<td>Data to monitor</td>
</tr>
<tr>
<td>ExcursionThreshold</td>
<td>REAL</td>
<td>Limit at which a deviation is detected</td>
</tr>
<tr>
<td>ChatterCount</td>
<td>UDINT</td>
<td>Number of deviations allowed within a time period defined by the ExcursionTime</td>
</tr>
<tr>
<td>ExcursionTime</td>
<td>TIME</td>
<td>Maximum time a sustained deviation is allowed</td>
</tr>
<tr>
<td>Reset</td>
<td>BOOL</td>
<td>Reset function block to default conditions</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>BOOL</td>
<td>Indication that the function block is enabled</td>
</tr>
<tr>
<td>Alert</td>
<td>SPS</td>
<td>Alert condition and associated metadata</td>
</tr>
<tr>
<td>Status</td>
<td>enum_AlertType</td>
<td>Enumeration describing the function block state</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelStatus</td>
<td>enum_ChannelAlert</td>
<td>Enumeration describing the channels that generated the status alert</td>
</tr>
<tr>
<td>QualityAlert</td>
<td>BOOL</td>
<td>Channel quality alert</td>
</tr>
<tr>
<td>QualityStatus</td>
<td>enum_ChannelAlert</td>
<td>Enumeration describing the channels that generated the quality alert</td>
</tr>
</tbody>
</table>

## Processing

- *ExcursionThreshold*, *ChatterCount*, and *ExcursionTime* are set the first time the function block is called. They cannot be altered after that time.
- On a rising edge of *ENO*, the tracked chatter count and excursion time are reset to zero.
- Disabling the function block by setting *EN* to FALSE does not clear the function block *Alert*.
- When *ENO* is FALSE or *Reset* is TRUE, the *Alert* SPS quality reports as invalid.
- The function block adheres to the following processing if *ENO* is TRUE.
  - Good channel quality is required for input processing. This is determined by the input channel *validity_t structure*, i.e., \( \text{AnalogQuantity.q.validity} = \text{good} \).
  - If a channel has bad quality, it is excluded from the excursion calculations and a *QualityAlert* is generated.
  - Compare the instantaneous values of the input channels to determine if any channel deviates from any other available channel.
  - If a *QualityAlert* is generated, the *QualityStatus* reports the offending channels as described in *enum_ChannelAlert*.
  - If the minimum number of channels do not have good quality, *Status* is BAD_-QUALITY as defined in the *enum_AlertType*.
  - If a channel deviates by more than *ExcursionThreshold* from any other channel for a sustained period given by *ExcursionTime*, an *Alert* is generated.
  - If a channel repeatedly deviates by more than *ExcursionThreshold* from any other channel and the number of deviations exceeds *ChatterCount* within a period given by *ExcursionTime*, an *Alert* is generated.
  - If *Alert* is asserted, *Status* identifies the cause of the alert as described in *enum_-AlertType*.
  - If an *Alert* is generated, *ChannelStatus* identifies the offending channels as described in *enum_ChannelAlert*.
  - Once an *Alert* is generated, the function block maintains its state at the time of the alert until issued a *Reset*.
  - If *Reset* is asserted, the function block does not process any inputs and *Status* is RESET as defined in *enum_AlertType*.
  - A falling edge of *Reset* returns the function block to a default state.
**fb_ChannelAlert**

Compare one measured value (MV) tag against a reference value to determine if the channel deviates outside a threshold value for a time period or if repeated deviations occur within a time period.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enable the function block</td>
</tr>
<tr>
<td>Channel</td>
<td>MV</td>
<td>Data to monitor</td>
</tr>
<tr>
<td>ChannelReference</td>
<td>REAL</td>
<td>Channel reference value</td>
</tr>
<tr>
<td>ExcursionThreshold</td>
<td>REAL</td>
<td>Limit at which a deviation is detected</td>
</tr>
<tr>
<td>ChatterCount</td>
<td>UDINT</td>
<td>Number of deviations allowed within a time period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>defined by the ExcursionTime</td>
</tr>
<tr>
<td>ExcursionTime</td>
<td>TIME</td>
<td>Maximum time a sustained deviation is allowed</td>
</tr>
<tr>
<td>Reset</td>
<td>BOOL</td>
<td>Reset function block to default conditions</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>BOOL</td>
<td>Indication that the function block is enabled</td>
</tr>
<tr>
<td>Alert</td>
<td>SPS</td>
<td>Alert condition and associated metadata</td>
</tr>
<tr>
<td>Status</td>
<td>enum_AlertType</td>
<td>Enumeration describing the function block state</td>
</tr>
<tr>
<td>QualityAlert</td>
<td>BOOL</td>
<td>Channel quality alert</td>
</tr>
</tbody>
</table>

**Processing**

- *ExcursionThreshold*, *ChatterCount*, and *ExcursionTime* are set the first time the function block is called. They cannot be altered after that time.
- On a rising edge of *ENO*, the tracked chatter count and excursion time are reset to zero.
- Disabling the function block by setting *EN* to FALSE does not clear the function block *Alert*.
- When *ENO* is FALSE or *Reset* is TRUE, the *Alert* SPS quality is invalid.
- The function block adheres to the following processing if *ENO* is TRUE.
- Good channel quality is required for input processing. This is determined by the input channel *validity_t* structure, i.e., `AnalogQuantity.q.validity = good`.
- If *Channel* has bad quality, no excursion calculation occurs and *QualityAlert* is asserted.
- Compare the instantaneous values of *Channel* and *ChannelReference* to determine if an excursion occurred.
- If *QualityAlert* is asserted, *Status* is BAD_QUALITY, as defined in the *enum_* _AlertType*.
- If *Channel* deviates by more than *ExcursionThreshold* from the reference for a sustained period given by *ExcursionTime*, an *Alert* is generated.
If Channel repeatedly deviates from the reference by more than ExcursionThreshold and the number of deviations exceeds ChatterCount within a period given by ExcursionTime, an Alert is generated.

If Alert is asserted, Status identifies the cause of the alert as described in enum_AlertType.

Once an alert is generated, the function block maintains its state at the time of the alert until issued a reset.

If Reset is asserted, the function block does not process any inputs and Status is RESET as defined in enum_AlertType.

A falling edge of Reset returns the function block to a default state.

**fb_IndicatorAlert**

Monitors one Boolean value for a sustained or chattering TRUE value.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enable the function block</td>
</tr>
<tr>
<td>Indicator</td>
<td>BOOL</td>
<td>Data to monitor</td>
</tr>
<tr>
<td>ChatterCount</td>
<td>UDINT</td>
<td>Number of deviations allowed within a time period defined by ExcursionTime</td>
</tr>
<tr>
<td>ExcursionTime</td>
<td>TIME</td>
<td>Maximum time a sustained deviation is allowed</td>
</tr>
<tr>
<td>Reset</td>
<td>BOOL</td>
<td>Reset function block to default conditions</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>BOOL</td>
<td>Indication that the function block is enabled</td>
</tr>
<tr>
<td>Alert</td>
<td>SPS</td>
<td>Alert condition and associated metadata</td>
</tr>
<tr>
<td>Status</td>
<td>enum_AlertType</td>
<td>Enumeration describing the function block state</td>
</tr>
</tbody>
</table>

**Processing**

- ChatterCount and ExcursionTime are set the first time the function block is called. They cannot be altered after that time.
- On a rising edge of ENO, the tracked chatter count and excursion time are reset to zero.
- Disabling the function block by setting EN to FALSE does not clear the function block Alert.
- When ENO is FALSE or Reset is TRUE and an alert condition is not detected, the Alert SPS quality is invalid.
- The function block adheres to the following processing if ENO is TRUE.
- Monitor Indicator for a TRUE value.
- If Indicator is TRUE for a sustained period given by ExcursionTime, an alert is generated.
- If Indicator repeatedly switches between FALSE and TRUE and the number of deviations exceed ChatterCount within a period given by ExcursionTime, an Alert is generated.
- If Alert is asserted, Status identifies the cause of the alert as described in enum_AlertType.
- Once an Alert is generated, the function block maintains its state at the time of the alert until issued a reset.
- If Reset is asserted, the function block does not process any inputs and Status is RESET as defined in enum_AlertType.
- A falling edge of Reset returns the function block to a default state.

**fb_ChannelDerivative**

Calculates the time derivative (rate of change) of a channel using finite difference approximation and alerts upon excursion beyond a user-settable threshold.

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enable the function block</td>
</tr>
<tr>
<td>Reset</td>
<td>BOOL</td>
<td>Reset the function block to a default state</td>
</tr>
<tr>
<td>Channel</td>
<td>MV</td>
<td>Input signal to differentiate</td>
</tr>
<tr>
<td>DerivativeThreshold</td>
<td>REAL</td>
<td>Threshold, over which the absolute value of Derivative will assert Alert. Must be greater than or equal to 0.</td>
</tr>
<tr>
<td>PeriodicProcessing</td>
<td>BOOL</td>
<td>Set to TRUE to process Channel on a fixed interval. Set to FALSE to process Channel based on changes in the Channel time stamp.</td>
</tr>
<tr>
<td>Period</td>
<td>TIME</td>
<td>Channel evaluation period when PeriodicProcessing = TRUE. Should be greater than or equal to, and equally divisible by the RTAC task time.</td>
</tr>
<tr>
<td>FilterLength</td>
<td>INT</td>
<td>The number of calculated derivatives to be averaged in order to update the Derivative output. Can be any odd integer between 1 and 21.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>BOOL</td>
<td>Indication that the function block is enabled</td>
</tr>
<tr>
<td>Alert</td>
<td>SPS</td>
<td>Indication of derivative excursion beyond threshold</td>
</tr>
<tr>
<td>Status</td>
<td>enum_AlertType</td>
<td>Enumeration describing the function block state</td>
</tr>
<tr>
<td>QualityAlert</td>
<td>BOOL</td>
<td>Channel quality alert</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derivative</td>
<td>MV</td>
<td>Average derivative of Channel over ConditionedFilterLength + 1 Channel samples</td>
</tr>
<tr>
<td>ConditionedFilterLength</td>
<td>INT</td>
<td>Adjusted FilterLength to ensure the filter length used is an odd number bounded by 1 and 21.</td>
</tr>
</tbody>
</table>

## Processing

- The Derivative output is given in units of \( \text{X per second} \) where \( \text{X} \) is the units of the Channel.instMag input.
- PeriodicProcessing and FilterLength are set the first time the function block is called, regardless of the state of the \( EN \) input. They cannot be altered after that time.
- \( ENO \) is true when \( EN = \text{TRUE} \) and the function block initialization is completed successfully.
- Successful function block initialization is dependent on user input validation. If the function block fails to initialize, Status is set to ERROR.
- If DerivativeThreshold represents a floating point value of NAN, Inf, or -Inf when the function block is first called, the function block fails to initialize.
- Disabling the function block by setting \( EN \) to FALSE does not clear the Alert function block output variable.
- While the Reset input is asserted, all internal variables and outputs are set to a default value. The Status output is set to \( \text{RESET} \).
- When \( EN \) is FALSE or Reset is TRUE, the Alert SPS quality is invalid.
- If the State output equals EXCURSION, Channel is not processed. The outputs are held at their current state until a rising edge of the Reset input is detected.
- The FilterLength input is evaluated against the requirements specified in the Inputs table. If FilterLength does not conform to the requirements, ConditionedFilterLength becomes a bounded version of FilterLength and is used for processing the Channel input.
- For PeriodicProcessing = FALSE, Channel processing is triggered by changes in the \( Channel.t.value \) time stamp. For this mode, the incremental derivative is defined as the change in \( Channel.instMag \) divided by the change in the \( Channel.t.value \) time stamp between the current \( Channel \) sample \( (k) \) and previously processed \( Channel \) sample \( (k - 1) \). The incremental derivative is assigned a time stamp equal to the \( k \) sample \( t.value \) time stamp. This mode can be useful for real-time streaming data sources such as IEEE C37.118 synchrophasors or off-line processing of data sets containing time-stamped samples.
- For PeriodicProcessing = TRUE, the Channel state is evaluated periodically at the interval specified by the Period input. The timer runs while \( EN = \text{TRUE AND Reset = FALSE} \). In this mode, the incremental derivative is defined as the change in \( Channel.instMag \) between the \( k \) and \( k - 1 \) samples divided by the Period input. The incremental derivative is assigned a time stamp equal to the RTAC system time.
of processing the \( k \) sample. This mode can be useful for real-time processing of deadbanded data sources where no \( Channel \) update is meant to be interpreted as a derivative of zero. When using this mode, the applied \( Period \) setting should be greater than or equal to, and equally divisible by, the RTAC task time.

- The function block maintains a buffer of the \( ConditionedFilterLength \) most recent incremental derivative results. The \( Derivative \) output represents the average of the buffered results. The \( Derivative \) output updates only when the buffer is full. The buffer is full once \( ConditionedFilterLength \) plus one \( Channel \) samples are processed.

- While \( EN = FALSE \), \( Channel \) is not processed. The cached \( k - 1 \) sample is not updated.

- While \( Channel.instMag \) represents a floating point value of \( \text{NAN}, \text{Inf}, \text{or} -\text{Inf} \), \( Channel \) is not processed. The \( Status \) is set to \( ERROR \). The cached \( k - 1 \) sample is not updated.

- Negative time-stamp differences between consecutive \( Channel \) samples are ignored when \( PeriodicProcessing = FALSE \). \( Channel \) is not processed. However, the cached \( k - 1 \) sample is updated to avoid a negative calculated sample interval on the next incremental derivative calculation. \( Status \) equals \( ERROR \) until a positive time-stamp difference is detected or \( Reset \) is asserted.

- While \( Channel \) is not being processed, the output \( Derivative \) value and time stamp are held at the last calculated result.

- As previously noted, \( Channel \) is not processed when \( EN = FALSE \), \( Channel.instMag \) represents an invalid REAL quantity, or when a negative time-stamp difference is detected while \( PeriodicProcessing = FALSE \). However, the buffer is not cleared in these cases. The next \( Channel \) sample that is processed causes the buffer to be updated with the derivative between the current sample and the cached \( k - 1 \) sample. While the resultant \( Derivative \) update in this case still represents the average derivative over \( ConditionedFilterLength \) plus one samples, it may not accurately portray the average derivative over \( ConditionedFilterLength \) plus one expected sample intervals. It is the responsibility of the user to clear the buffer by asserting \( Reset \) if \( Channel \) processing is inhibited for a duration deemed unacceptable.

- The output \( Derivative.t \) structure is set equal to the time stamp of the incremental derivative result at the center position of the buffer. This is done for derivative approximation accuracy.

- The output \( Derivative \) is assigned a quality that represents the lowest quality indicators of all \( Channel \) samples processed in the calculation of the output derivative value.

- If the \( Derivative.q.validity \) does not equal \( \text{good} \) then the output \( QualityAlert \) is asserted.

- If the absolute value of the output \( Derivative.instMag \) exceeds the absolute value of \( DerivativeThreshold \), \( Alert.stVal \) is asserted. \( Alert.t \) is set equal to the RTAC system time. \( Status \) is set to \( EXCURSION \).

**fb_ChannelIntegral**

Calculates the area under the input channel magnitude and above a user-defined integration bound using trapezoidal approximation between samples.
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enable the function block</td>
</tr>
<tr>
<td>Reset</td>
<td>BOOL</td>
<td>Reset the function block to default conditions</td>
</tr>
<tr>
<td>Channel</td>
<td>MV</td>
<td>Signal to integrate</td>
</tr>
<tr>
<td>SetPoint</td>
<td>REAL</td>
<td>Channel threshold used to initiate integration.</td>
</tr>
<tr>
<td>PeriodicProcessing</td>
<td>BOOL</td>
<td>Set to TRUE to process Channel on a fixed interval.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set to FALSE to process Channel based on changes in the Channel time stamp.</td>
</tr>
<tr>
<td>Period</td>
<td>TIME</td>
<td>Channel evaluation period when PeriodicProcessing = TRUE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Should be greater than or equal to, and equally divisible by the RTAC task time.</td>
</tr>
<tr>
<td>LowerBound</td>
<td>REAL</td>
<td>Lower bound used in calculation of Integral.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Must be less than or equal to SetPoint.</td>
</tr>
<tr>
<td>DebounceTime</td>
<td>TIME</td>
<td>Time required for channel to be above or below SetPoint in order for the associated SetPoint excursion time to be considered the beginning or end of an integration period.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>BOOL</td>
<td>Indication that the function block is enabled</td>
</tr>
<tr>
<td>Alert</td>
<td>SPS</td>
<td>Indication of a completed integration period</td>
</tr>
<tr>
<td>Status</td>
<td>enum_AlertType</td>
<td>Enumeration describing the function block state</td>
</tr>
<tr>
<td>QualityAlert</td>
<td>BOOL</td>
<td>Asserts when channel quality is bad or integral output accuracy is suspect</td>
</tr>
<tr>
<td>ExcursionTimeOn</td>
<td>dateTime_t</td>
<td>Time stamp marking the start of an integration period</td>
</tr>
<tr>
<td>ExcursionTimeOff</td>
<td>dateTime_t</td>
<td>Time stamp marking the end of an integration period</td>
</tr>
<tr>
<td>Integral</td>
<td>MV</td>
<td>The integral of Channel below Channel.instMag and above LowerBound, bounded by ExcursionTimeOn and ExcursionTimeOff</td>
</tr>
<tr>
<td>Peak</td>
<td>MV</td>
<td>Peak value of Channel between ExcursionTimeOn and ExcursionTimeOff</td>
</tr>
</tbody>
</table>

## Processing

- The **Integral** output is given in units of \(X \times \text{seconds}\) where \(X\) is the units of the Channel.instMag input.

- **Period** and **PeriodicProcessing** are set the first time the function block is called, regardless of the state of the EN input. They cannot be altered after that time.

- **ENO** is true when \(EN = \text{TRUE}\) and the function block initialization is completed successfully.

- Successful function block initialization is dependent on user input validation. If the function block fails to initialize, **Status** is set to **ERROR**.
If any of the following conditions are true during the first call of the function block, the function block fails to initialize.

1. SetPoint represents a floating point value of \( \text{NAN}, \text{Inf}, \text{or} -\text{Inf} \).
2. LowerBound represents a floating point value of \( \text{NAN}, \text{Inf}, \text{or} -\text{Inf} \) or is a defined number greater than SetPoint.
3. PeriodicProcessing = TRUE and Period is less than or equal to zero.
4. DebounceTime is less than zero.

All inputs other than Period and PeriodicProcessing can be modified during run-time. However, SetPoint, LowerBound, and DebounceTime are held static while State = EXCURSION or EXPIRATION. While not held static, these inputs shall be validated against the previously stated conditions.

While Channel\_instMag represents a floating point value of \( \text{NAN}, \text{Inf}, \text{or} -\text{Inf} \) or any variable input is deemed invalid, the Status is set to ERROR. The cached \( k - 1 \) sample is not updated.

While ENO is FALSE or Reset is TRUE, Alert\_q\_validity, Integral\_q\_validity, and Peak\_q\_validity are set to invalid.

While the Reset input is asserted, all outputs are reset to default values. The Status output is set to RESET. A falling edge of the Reset input returns Status to NO-DEVIATION.

While Alert\_stVal = TRUE and Status is COMPLETE, the function block halts data processing. Outputs are frozen until Reset is asserted.

The function block adheres to the following processing if ENO is TRUE and Status is not COMPLETE.

For PeriodicProcessing = FALSE, Channel processing is triggered by changes in the Channel\_t\_value time stamp. This mode can be useful for real-time streaming data sources such as IEEE C37.118 synchrophasors or off-line processing of data sets containing time-stamped samples.

For PeriodicProcessing = TRUE, the Channel state is evaluated periodically at the interval specified by the Period input. The timer runs while EN = TRUE AND Reset = FALSE. In this mode, the input Channel is assigned time stamps from the RTAC system clock. This mode can be useful for real-time processing of deadbanded data sources where no time-stamp update is meant to be interpreted as a repeated value. When PeriodicProcessing = TRUE, the applied Period setting should be greater than and equally divisible by the RTAC task time.

The incremental update to the Integral output is defined as the area of the trapezoid bound by the following two points and the line defined by LowerBound:

- Channel\_instMag at Channel\_t time for the most recently processed sample (\( k \)).
- Channel\_instMag at the Channel\_t time for the previously processed sample (\( k - 1 \)).

Negative time-stamp differences between consecutive Channel samples are ignored. In this scenario, the Channel sample is not used in the integral approximation. However, the cached \( k - 1 \) sample is updated to avoid a negative calculated sample interval on the next incremental update to the Integral output. Status equals ERROR until a positive time-stamp difference is detected or Reset is asserted.
The integration period begins when \( \text{Channel} . \text{instMag} \) is in excess of \( \text{SetPoint} \). At this time \( \text{Status} \) is set to EXCURSION.

\( \text{Channel} . \text{instMag} \) must exceed \( \text{SetPoint} \) for a minimum time equal to \( \text{DebounceTime} \) in order for integration to complete.

If \( \text{Channel} . \text{instMag} \) is in excess of \( \text{SetPoint} \), but becomes equal to or less than \( \text{SetPoint} \) before \( \text{DebounceTime} \) is reached, the function block is reset on the first task cycle for which \( \text{Channel} . \text{instMag} \) is not in excess of \( \text{SetPoint} \).

If \( \text{Channel} . \text{instMag} \) exceeds \( \text{SetPoint} \) for a duration equal to \( \text{DebounceTime} \), \( \text{Status} \) is set to EXPIRATION and \( \text{ExcursionTimeOn} \) is assigned as described below.

While \( \text{Status} \) is set to EXPIRATION integration will continue until \( \text{Channel} . \text{instMag} \) falls below \( \text{SetPoint} \) for time equal to \( \text{DebounceTime} \).

If the preceding debounce time condition is met, \( \text{Alert} . \text{stVal} \) asserts, \( \text{Alert} . \text{t.value} \) is set equal to the RTAC system time. \( \text{Status} \) is set to COMPLETE and \( \text{ExcursionTimeOff} \) is assigned as described below.

\( \text{ExcursionTimeOn} \) and \( \text{ExcursionTimeOff} \), respectively, are assigned a derived time-stamp that is between the time stamp values associated with the two processed \( \text{Channel} \) samples that straddle \( \text{SetPoint} \). More specifically, this time stamp coincides with the intersection of the line drawn between the .\text{instMag} values of these two samples and \( \text{SetPoint} \).

\( \text{Integral} . \text{t.value} \) is set equal to \( \text{ExcursionTimeOn} \) and will not be updated until the function block is reset.

The \( \text{Integral} \) output represents the area under \( \text{Channel} . \text{instMag} \) and over \( \text{LowerBound} \) between \( \text{ExcursionTimeOn} \) and \( \text{ExcursionTimeOff} \) as shown in Equation 1.

\[
\text{Integral} . \text{instMag} \approx \int_{t=\text{ExcursionTimeOn}}^{t=\text{ExcursionTimeOff}} (\text{ChannelProcess}(t) - \text{LowerBound}) \, dt
\]

(Equation 1)

where \( \text{ChannelProcess}(t) \) is the physical process being measured and represented by \( \text{Channel} . \text{instMag} \) measurements.

The function block updates the \( \text{Integral} \) output continuously during the integration period. This enables external evaluation the current integral result against an auxiliary excursion threshold.

The \( \text{Peak} \) output contains the magnitude, quality and time-stamp information from the \( \text{Channel} \) sample in which \( \text{Channel} . \text{instMag} \) was at a maximum value over the integration period. For repeated maximums, the most recent maximum \( \text{Channel} \) value is applied to \( \text{Peak} \).

During integration, the \( \text{Integral} \) output is assigned a quality that represents the lowest quality indicators of all \( \text{Channel} \) samples used in the integration calculation.

If the \( \text{Integral} . \text{q.validity} \) does not equal good then the output \( \text{QualityAlert} \) is asserted.

If \( \text{Channel} . \text{instMag} \) is already in excess of \( \text{SetPoint} \) when \( \text{EN} \) is asserted or after a manual reset, \( \text{ExcursionTimeOn} \) is assigned the time stamp of the first processed \( \text{Channel} \) sample. This time stamp is not expected to represent the approximate time of \( \text{SetPoint} \) crossing. In this instance, the output \( \text{QualityAlert} \) is asserted.
fb_IndicatorTimeDelta

Monitors the time-stamp difference between the assertions of two Single Point Status (SPS) indicators and alerts upon time-difference excursion beyond a user-defined threshold.

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>BOOL</td>
<td>Enable the function block</td>
</tr>
<tr>
<td>Reset</td>
<td>BOOL</td>
<td>Reset the function block to default conditions</td>
</tr>
<tr>
<td>Indicator1</td>
<td>SPS</td>
<td>Indicator anticipated to assert first</td>
</tr>
<tr>
<td>Indicator2</td>
<td>SPS</td>
<td>Indicator anticipated to assert second</td>
</tr>
<tr>
<td>TimeDiffThreshold</td>
<td>REAL</td>
<td>Absolute time-stamp difference at which alert condition is triggered. Must be greater than 0. Units are in seconds.</td>
</tr>
<tr>
<td>WaitTime</td>
<td>TIME</td>
<td>Maximum time allowed between indicator assertions before internal variables are cleared. Must be greater than TimeDiffThreshold.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>IEC 61131 Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENO</td>
<td>BOOL</td>
<td>Indication that the function block is enabled</td>
</tr>
<tr>
<td>Alert</td>
<td>SPS</td>
<td>Indication that the calculated TimeDifference has exceeded TimeDiffThreshold</td>
</tr>
<tr>
<td>QualityAlert</td>
<td>BOOL</td>
<td>Indicator quality alert</td>
</tr>
<tr>
<td>TimeDifference</td>
<td>REAL</td>
<td>Signed time-stamp difference: Indicator2.t.value minus Indicator1.t.value. Units are in seconds</td>
</tr>
<tr>
<td>Status</td>
<td>enum_AlertType</td>
<td>Enumeration describing the function block state</td>
</tr>
</tbody>
</table>

Processing

- TimeDiffThreshold and WaitTime inputs are held static on the first task cycle. Therefore, they can not be changed during runtime.
- ENO is true when \( EN = \text{TRUE} \) and the function block initialization is completed successfully.
- Successful function block initialization is dependent on user input validation. If the function block fails to initialize, Status is set to ERROR.
- If any of the following conditions are true during the first call of the function block, the function block fails to initialize.
  1. WaitTime is less than TimeDiffThreshold.
  2. TimediffThreshold is less than zero seconds.
- The function block can be reset either from a user asserted RESET or an internal reset because of the expiration of WaitPeriod. If reset, outputs return to a default state. Outputs are held in this state while RESET is TRUE.
- Disabling the function block by setting EN to FALSE does not clear the function block Alert.
When ENO is FALSE or Reset is TRUE, the Alert SPS quality is set to invalid.

The function block adheres to the following processing if ENO is TRUE.

Good Indicator quality is required for input processing. This is determined by the input indicator validity_t structure, i.e., SPS.q.validity = good.

If either input indicator has bad quality, processing is halted, QualityAlert is asserted, Status is set to BAD_QUALITY, and Alert.q.validity is set to invalid. Note that this does not trigger a reset, nor does it require a reset to clear.

If Status is EXPIRATION, input processing stops until a user-initiated RESET is executed.

The WaitPeriod timer is initiated by the rising edge of Indicator1.stVal or Indicator2.stVal while the other indicator’s .stVal is deasserted.

If the WaitPeriod timer expires before the remaining indicator .stVal asserts, a reset is initiated and the function block returns to normal operation.

If the remaining indicator .stVal asserts before the WaitPeriod timer has elapsed, the signed time difference between the input indicators is calculated and assigned to TimeDifference.

TimeDifference is defined as Indicator2.t.value minus Indicator1.t.value, where each respective time stamp is recorded at the rising edge of the indicators’ .stVal.

The TimeDifference output is accurate to within plus or minus 500 microseconds.

If Indicator2.stVal asserts before Indicator1.stVal, the output TimeDifference represents a negative time difference.

If ABS(TimeDifference) > TimeDiffThreshold, Alert.stVal asserts and Alert.t is set equal to the RTAC system time.

If Alert.stVal is TRUE, Status is set to EXPIRATION.

---

Benchmarks

Benchmark Platforms

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

- SEL-3530
  - R135-V1 firmware
- SEL-3505
  - R135-V1 firmware
- SEL-3555
  - Dual-core Intel i7-3555LE processor
  - 4 GB ECC RAM
  - R135-V1 firmware
Benchmark Test Descriptions

Each benchmarking test is performed 1000 times and the average run time is recorded here. Each test is intended to give insight into the expected cost of running the given command.

fun_GetAlertString

The cost of a call to fun_GetAlertString.

fun_GetChannelString

The cost of a call to fun_GetChannelString.

fb_MultiChannelAlert No Alert

The cost of a call to fb_MultiChannelAlert when all channels are active and no alert is generated.

fb_MultiChannelAlert 1 Channel Timed

The cost of a call to fb_MultiChannelAlert when all channels are active and one channel differs from the others long enough to generate an alert. This is the run time on the scan the alert begins.

fb_MultiChannelAlert All Channel Timed

The cost of a call to fb_MultiChannelAlert when all channels are active and all three channels differ from each other long enough to generate an alert. This is the run time on the scan the alert begins.

fb_MultiChannelAlert 1 Channel Chatter

The cost of a call to fb_MultiChannelAlert when all channels are active and one channel differs from the others often enough to generate an alert. This is the run time on the scan the alert begins.

fb_MultiChannelAlert All Channel Chatter

The cost of a call to fb_MultiChannelAlert when all channels are active and all three channels differ from each other often enough to generate an alert. This is the run time on the scan the alert begins.

fb_ChannelAlert No Alert

The cost of a call to fb_ChannelAlert when no alert is generated.
**fb_ChannelAlert Timed**

The cost of a call to fb_ChannelAlert when the input differs from the reference long enough to generate an alert. This is the run time on the scan the alert begins.

**fb_ChannelAlert Chatter**

The cost of a call to fb_ChannelAlert when the input differs from the reference often enough to generate an alert. This is the run time on the scan the alert begins.

**fb_IndicatorAlert No Alert**

The cost of a call to fb_IndicatorAlert when no alert is generated.

**fb_IndicatorAlert Timed**

The cost of a call to fb_IndicatorAlert when the input is true long enough to generate an alert. This is the run time on the scan the alert begins.

**fb_IndicatorAlert Chatter**

The cost of a call to fb_IndicatorAlert when the input is true often enough to generate an alert. This is the run time on the scan the alert begins.

**fb_ChannelDerivative Active Periodic**

The cost of a call to fb_ChannelDerivative during active derivative calculation on a Channel input while in periodic processing mode (PeriodicProcessing = TRUE).

**fb_ChannelDerivative Active Not Periodic**

The cost of a call to fb_ChannelDerivative during active derivative calculation on a Channel input while in not in periodic processing mode (PeriodicProcessing = FALSE). In this mode sample processing is triggered by Channel time-stamp changes.

**fb_ChannelDerivative Alert**

The cost of a call to fb_ChannelDerivative while Status = EXCURSION and Alert.stVal = TRUE.
**fb_ChannelIntegral No Deviation Periodic**

The cost of a call to fb_ChannelIntegral while it is in an idle state, and while it is in periodic processing mode (PeriodicProcessing = TRUE).

**fb_ChannelIntegral No Deviation Not Periodic**

The cost of a call to fb_ChannelIntegral during an idle state while not in periodic processing mode (PeriodicProcessing = FALSE). In this mode sample processing is triggered by Channel time-stamp changes.

**fb_ChannelIntegral Active Periodic**

The cost of a call to fb_ChannelIntegral during an active integration state while in periodic processing mode (PeriodicProcessing = TRUE).

**fb_ChannelIntegral Active Not Periodic**

The cost of a call to fb_ChannelIntegral during an active integration state while not in periodic processing mode (PeriodicProcessing = FALSE). In this mode sample processing is triggered by Channel time-stamp changes.

**fb_ChannelIntegral Complete Periodic**

The cost of a call to fb_ChannelIntegral during a Status = COMPLETE state while in periodic processing mode (PeriodicProcessing = TRUE).

**fb_ChannelIntegral Complete Not Periodic**

The cost of a call to fb_ChannelIntegral during a Status = COMPLETE state while not in periodic processing mode (PeriodicProcessing = FALSE). In this mode sample processing is triggered by Channel time-stamp changes.

**fb_IndicatorTimeDelta No Deviation**

The cost of a call to fb_IndicatorTimeDelta while it is in a Status = NO_DEVIATION state (Both indicators’ inputs are deasserted).

**fb_IndicatorTimeDelta Bad Quality**

The cost of a call to fb_IndicatorTimeDelta during a Status = BAD_QUALITY state (either indicator input has .q.validity that is not equal to good).
**fb_IndicatorTimeDelta Waiting For Second Indicator**

The cost of a call to fb_IndicatorTimeDelta while one indicator input is asserted and the function block is waiting for the second indicator input to assert.

**fb_IndicatorTimeDelta Alert State**

The cost of a call to fb_IndicatorTimeDelta during a Alert.stVal = TRUE state (time difference has exceeded the threshold. Alert is held high until a user reset).

### Benchmark Results

<table>
<thead>
<tr>
<th>Operation Tested</th>
<th>SEL-3530</th>
<th>SEL-3505</th>
<th>SEL-3555</th>
</tr>
</thead>
<tbody>
<tr>
<td>fun_GetAlertString</td>
<td>7</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>fun_GetChannelString</td>
<td>8</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>fb_MultiChannelAlert No Alert</td>
<td>16</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>fb_MultiChannelAlert 1 Channel Timed</td>
<td>21</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>fb_MultiChannelAlert All Channel Timed</td>
<td>21</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>fb_MultiChannelAlert 1 Channel Chatter</td>
<td>24</td>
<td>34</td>
<td>4</td>
</tr>
<tr>
<td>fb_MultiChannelAlert All Channel Chatter</td>
<td>24</td>
<td>39</td>
<td>4</td>
</tr>
<tr>
<td>fb_ChannelAlert No Alert</td>
<td>10</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>fb_ChannelAlert Timed</td>
<td>16</td>
<td>22</td>
<td>3</td>
</tr>
<tr>
<td>fb_ChannelAlert Chatter</td>
<td>18</td>
<td>26</td>
<td>3</td>
</tr>
<tr>
<td>fb_IndicatorAlert No Alert</td>
<td>8</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>fb_IndicatorAlert Timed</td>
<td>14</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>fb_IndicatorAlert Chatter</td>
<td>16</td>
<td>22</td>
<td>3</td>
</tr>
<tr>
<td>fb_ChannelDerivative Active Periodic</td>
<td>56</td>
<td>123</td>
<td>10</td>
</tr>
<tr>
<td>fb_ChannelDerivative Active Not Periodic</td>
<td>105</td>
<td>138</td>
<td>18</td>
</tr>
<tr>
<td>fb_ChannelDerivative Alert</td>
<td>6</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>fb_ChannelIntegral No Deviation Periodic</td>
<td>26</td>
<td>90</td>
<td>5</td>
</tr>
<tr>
<td>fb_ChannelIntegral No Deviation Not Periodic</td>
<td>22</td>
<td>41</td>
<td>3</td>
</tr>
<tr>
<td>fb_ChannelIntegral Active Periodic</td>
<td>31</td>
<td>111</td>
<td>4</td>
</tr>
<tr>
<td>fb_ChannelIntegral Active Not Periodic</td>
<td>28</td>
<td>78</td>
<td>3</td>
</tr>
<tr>
<td>fb_ChannelIntegral Complete Periodic</td>
<td>7</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>fb_ChannelIntegral Complete Not Periodic</td>
<td>7</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>fb_IndicatorTimeDelta No Deviation</td>
<td>9</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>fb_IndicatorTimeDelta Bad Quality</td>
<td>7</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>fb_IndicatorTimeDelta Waiting For Second Indicator</td>
<td>11</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>fb_IndicatorTimeDelta Alert State</td>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

### 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.

Monitor Phase-A Measurements for a Maintenance Condition

Objective

Create a program to monitor and verify the measurements obtained from three protective relays to determine if the components are functioning within expected limits.

Solution

This solution uses the fb_ChannelAlert function block to monitor for difference between CTs. The Phase A measurements are obtained from the relays and compared against a reference measurement (see Code Snippet 1).
Code Snippet 1  prg_MonitorPhaseA_Components

PROGRAM prg_MonitorPhaseA_Components
VAR
  (*Function block monitoring IED 1-3*)
  IED_1_PhA : fb_ChannelAlert;
  IED_2_PhA : fb_ChannelAlert;
  IED_3_PhA : fb_ChannelAlert;
  (*Function block parameters*)
  PhA_Reference : MV; Allowed_Deviation : REAL; Allowed_Chatter : UDINT;
  AlertTime : TIME; MonitorReset : BOOL;
  (*Criterion to enable the monitoring block*)
  EnableMonitoring : BOOL;
  (*Alert status*)
  IED_1_Enabled : BOOL; IED_2_Enabled : BOOL; IED_3_Enabled : BOOL;
  (*Placeholder for a data communications tag*)
  IED_1_Data : MV; IED_2_Data : MV; IED_3_Data : MV;
  (*Alert status*)
  IED_1_Alert : SPS; IED_2_Alert : SPS; IED_3_Alert : SPS;
  (*Alert condition*)
  IED_1_Status : DINT; IED_2_Status : DINT; IED_3_Status : DINT;
  (*Quality status*)
  IED_1_Quality : BOOL; IED_2_Quality : BOOL; IED_3_Quality : BOOL;
END_VAR

(*Check the quality of the reference signal*)
EnableMonitoring := (PhA_Reference.q.validity = good);

(*Configure and monitor the function block parameters*)
IED_1_PhA(EN := EnableMonitoring, Channel := IED_1_Data,
  ChannelReference := PhA_Reference.instMag,
  ExcursionThreshold := Allowed_Deviation,
  ChatterCount := Allowed_Chatter, ExcursionTime := AlertTime,
  Reset := MonitorReset, ENO => IED_1_Enabled, Alert => IED_1_Alert,
  Status => IED_1_Status, QualityAlert => IED_1_Quality);

IED_2_PhA(EN := EnableMonitoring, Channel := IED_2_Data,
  ChannelReference := PhA_Reference.instMag,
  ExcursionThreshold := Allowed_Deviation,
  ChatterCount := Allowed_Chatter, ExcursionTime := AlertTime,
  Reset := MonitorReset, ENO => IED_2_Enabled, Alert => IED_2_Alert,
  Status => IED_2_Status, QualityAlert => IED_2_Quality);

IED_3_PhA(EN := EnableMonitoring, Channel := IED_3_Data,
  ChannelReference := PhA_Reference.instMag,
  ExcursionThreshold := Allowed_Deviation,
  ChatterCount := Allowed_Chatter, ExcursionTime := AlertTime,
  Reset := MonitorReset, ENO => IED_3_Enabled, Alert => IED_3_Alert,
  Status => IED_3_Status, QualityAlert => IED_3_Quality);
Creating an Object to Verify and Monitor IED Operation

Objective

Create a program to monitor for deviations between phases on the generator and load sides of a breaker.

Solution

This solution uses the fb_MultiChannelAlert function block to monitor the three phases of a CT. The phase measurements are obtained from the relays on both the generator and load sides of a breaker. All the phases are compared against each other to detect damage or a maintenance condition in CT/PT windings.

Code Snippet 2  prg_MonitorBreakerHighLoadSideComponents

```plaintext
PROGRAM prg_MonitorBreakerHighLoadSideComponents
VAR
  (*Function block monitoring generator side of breaker*)
  Gen_Monitor : fb_MultiChannelAlert;
  (*Function block monitoring bus side of breaker*)
  Bus_Monitor : fb_MultiChannelAlert;
  (*Generator nominal current*)
  GenNominal : REAL;
  (*Actual generator output*)
  GenOutput : REAL;
  (*Set the limit the channels are allowed to deviate by*)
  AllowedDeviation : REAL;
  (*Criterion to enable the monitoring block*)
  Enable_FB : BOOL;
  (*Placeholder for a data communications tag*)
  PhaseA_X_Terminal : MV; PhaseB_X_Terminal : MV; PhaseC_X_Terminal : MV;
  (*Placeholder for a data communications tag*)
  PhaseA_Y_Terminal : MV; PhaseB_Y_Terminal : MV; PhaseC_Y_Terminal : MV;
  (*Clear the alert condition and restore block to default condition*)
  FB_Reset : BOOL;
  (*Function block successfully enabled*)
  GenFB_Enabled : BOOL; BusFB_Enabled : BOOL;
  (*Gen_FB alert information*)
  GenAlert : SPS; GenFB_Status : enum_AlertType; GenAlertCause :
    enum_ChannelAlert;
  GenQualityAlert : BOOL; GenQualityCause : enum_ChannelAlert;
  (*Bus_FB alert information*)
  BusAlert : SPS; BusFB_Status : enum_AlertType; BusAlertCause :
    enum_ChannelAlert;
  BusQualityAlert : BOOL; BusQualityCause : enum_ChannelAlert;
  (*Detect an alert condition*)
  Gen_Alert_Generated : R_TRIG; Bus_Alert_Generated : R_TRIG;
  Gen_Status_Message : STRING; Bus_Status_Message : STRING;
  Gen_Channel_Message : STRING; Bus_Channel_Message : STRING;
END_VAR
```
ChannelMonitoring

Examples

Code Snippet 2  prg_MonitorBreakerHighLoadSideComponents (Continued)

(*If the generator output exceeds 5% of nominal, enable the monitoring blocks*)
Enable_FB := (GenOutput >= 0.05 * GenNominal);

(*Function block monitoring the X terminal - high side*)
Gen_Monitor(EN := Enable_FB, Channel_1 := PhaseA_X_Terminal,
Channel_2 := PhaseB_X_Terminal, Channel_3 := PhaseC_X_Terminal,
ExcursionThreshold := AllowedDeviation, ChatterCount := 3,
ExcursionTime := T#1M, Reset := FB_Reset, ENO => GenFB_Enabled,
Alert => GenAlert, Status => GenFB_Status, ChannelStatus =>
GenAlertCause,
QualityAlert => GenQualityAlert, QualityStatus =>
GenQualityCause);

(*Function block monitoring the Y terminal - load side*)
Bus_Monitor(EN := Enable_FB, Channel_1 := PhaseA_Y_Terminal,
Channel_2 := PhaseB_Y_Terminal, Channel_3 := PhaseC_Y_Terminal,
ExcursionThreshold := AllowedDeviation, ChatterCount := 3,
ExcursionTime := T#1M, Reset := FB_Reset, ENO => BusFB_Enabled,
Alert => BusAlert, Status => BusFB_Status, ChannelStatus =>
BusAlertCause,
QualityAlert => BusQualityAlert, QualityStatus =>
BusQualityCause);

//If an alert condition is detected, generate a message for logging
Gen_Alert_Generated(CLK := GenAlert.stVal);
Bus_Alert_Generated(CLK := BusAlert.stVal);

IF Gen_Alert_Generated.Q THEN
Gen_Status_Message := fun_GetAlertString(GenFB_Status);
Gen_Channel_Message := fun_GetChannelString(GenAlertCause);
END_IF

IF Bus_Alert_Generated.Q THEN
Bus_Status_Message := fun_GetAlertString(BusFB_Status);
Bus_Channel_Message := fun_GetChannelString(BusAlertCause);
END_IF

Creating an Object to Verify and Monitor Communications Channels and Hardware Alarms

Objective

Monitor and verify that a communications channel is functioning properly and that no hardware failures are detected for an IED.
Solution

This solution uses the fb_StatusAlert function block to detect a TRUE condition in either a communications diagnostic or hardware indicator. An appropriate communications channel diagnostic, such as the Offline bit in GOOSE, is monitored for communications channel failure. The HALARM Relay Word bit in an IED is monitored for hardware failures only if the communications channel is online.

Code Snippet 3  

```
PROGRAM prg_MonitorIED_Components
VAR
  (*Monitor the HALARM Relay Word bit*)
  IED_1_HardwareMonitor : fb_IndicatorAlert;
  (*Monitor a Mirrored Bits or GOOSE communications channel*)
  ProtectionChannelMonitor : fb_IndicatorAlert;
  (*Criterion to enable the monitoring block*)
  EnableHardwareMonitoring : BOOL;
  (*Reset after results are recorded*)
  DailyReset : BOOL;
  (* Placeholder for a data tag*)
  HALARM : BOOL;
  Communication_Client_Offline : BOOL;
  ProtectionChannelDiagnostic : BOOL;
  (*HALARM monitoring status*)
  IED_1_HALARM_MonitorEnabled : BOOL;
  IED_1_HALRM_Alert : SPS;
  IED_1_HALRM_Status : DINT;
  (*Protection monitoring status*)
  ProtectionChannelMonitor_Enabled : BOOL;
  (*Alert status*)
  ProtectionChannel_Alert : SPS;
  (*Alert condition*)
  ProtectionChannel_Status : DINT;
END_VAR

(*If the offline status is false, monitor the HALARM Relay Word bit. Note this is a separate offline bit than that used in the ProtectionChannelMonitor*)
EnableHardwareMonitoring := NOT Communication_Client_Offline;

(*Configure and monitor the function block parameters*)
IED_1_HardwareMonitor(EN := EnableHardwareMonitoring, Indicator := HALARM,
  ChatterCount := 1, ExcursionTime := T#10S,
  Reset := DailyReset, ENO => IED_1_HALARM_MonitorEnabled,
  Alert => IED_1_HALRM_Alert, Status =>
  IED_1_HALRM_Status);

ProtectionChannelMonitor(EN := TRUE, Indicator :=
  ProtectionChannelDiagnostic,
  ChatterCount := 2, ExcursionTime := T#5S,
  Reset := DailyReset,
  ENO=>ProtectionChannelMonitor_Enabled,
  Alert => ProtectionChannel_Alert,
  Status => ProtectionChannel_Status);
```
Creating an Object to Calculate Kilowatt-Hours Delivered During a Peak Demand Period

Objective

Calculate kilowatt-hours delivered during a period of peak demand.

Solution

This solution uses the fb_ChannelIntegral function block to monitor a measured power quantity and calculate the integral over time while the power is in excess of a user-defined peak-demand threshold. This example assumes the following:

1. Power measurements are received from a SEL-351 Modbus client, using a holding register named “KW3DI” (type = APC), polling interval of two seconds.
2. A virtual tag list called HMI_Controls was created for program control and status outputs. Virtual tag list tags shown in this example are defined as the following data types.
   - Aggregation_Complete: SPS
   - Aggregator_Reset: SPC
   - Demand_Threshold: MV (Analog Control)
   - KWH_During_Peak: MV
   - MaxKWDuringPeak: MV
   - Monitor_Enabled: SPS
   - Monitor_Quality_Alert: SPS
   - Peak_End_Time: STR
   - Peak_Start_Time: STR
   - Peak_Time_Active: SPS

 (*This example demonstrates the calculation of Kilowatt-hours over a period of high demand, given a power measurement in units of Kilowatts. This program sets the PeriodicProcessing input of the fb_ChannelIntegral instance to TRUE since the Modbus source will not update the Channel.t.value time-stamp on its own. The Period input is set to two seconds which corresponds with the Modbus holding register poll interval. Kilowatts integrated over time will produce a result in units of Joules. Where one Joule = one Watt-Second. To convert Joules to Kilowatt-Hours, the result must be divided by (3600 seconds/hour * 1000Watts/KiloWatts) = 3,600,000 Watt-Seconds/Kilowatt-Hour.*)

```plaintext
PROGRAM prg_KWH_Track
VAR
  Enable : BOOL;
  Aggregator : fb_ChannelIntegral;
```
Joules_to_KWH : REAL := 3600000;
KWH_During_Peak : REAL;
Peak_Start_Time : dateTime_t;
Peak_End_Time : dateTime_t;
QualityAlert : BOOL;

END_VAR

//Determine Enable condition
Enable := NOT SEL_351_1_MODBUS_POU.Offline
AND SEL_351_1_MODBUS.KW3DI.status.q.validity = good;

//Run the Integrator function block
Aggregator( EN := Enable,
Reset := HMI_Controls.Aggregator_Reset.operSet.ctlVal,
Channel := SEL_351_1_MODBUS.KW3DI.status,
SetPoint := HMI_Controls.Demand_Threshold.oper.setMag,
PeriodicProcessing := TRUE,
Period := T#2S, //Set to 2 seconds to match the
//Holding Register poll interval.
LowerBound := 0,
DebounceTime := T#10S);

//Load monitor status variables
HMI_Controls.Monitor_Enabled.stVal := Aggregator.ENO;
HMI_Controls.Monitor_Quality_Alert.stVal := Aggregator.QualityAlert;
HMI_Controls.Peak_Time_Active.stVal := Aggregator.Status = EXPIRATION
OR Aggregator.Status = EXCURSION;
HMI_Controls.Aggregation_Complete := Aggregator.Alert;

//Load outputs
KWH_During_Peak := Aggregator.Integral.instMag / Joules_to_KWH;
HMI_Controls.KWH_During_Peak := Aggregator.Integral;
HMI_Controls.KWH_During_Peak.instMag := KWH_During_Peak;
HMI_Controls.KWH_During_Peak.mag := KWH_During_Peak;
HMI_Controls.MaxKWDuringPeak := Aggregator.Peak;

//Update Peak demand on and Peak demand off time-stamps
HMI_Controls.Peak_Start_Time.strVal :=
DT_TO_STRING(Aggregator.ExcursionTimeOn.dateTime);
HMI_Controls.Peak_End_Time.strVal :=
DT_TO_STRING(Aggregator.ExcursionTimeOff.dateTime);

//Force good quality on tags with no other quality source.
HMI_Controls.Peak_End_Time.q.validity := good;
HMI_Controls.Peak_Start_Time.q.validity := good;
HMI_Controls.Aggregator_Reset.status.q.validity := good;
HMI_Controls.Demand_Threshold.status.q.validity := good;
HMI_Controls.Monitor_Enabled.q.validity := good;
HMI_Controls.Monitor_Quality_Alert.q.validity := good;
HMI_Controls.Peak_Time_Active.q.validity := good;
Creating an Object to Monitor a Client's Go-Online Time Delay

Objective

Calculate the time delta between RTAC runtime initiation and the deassertion of a communication client Offline POU pin. Assert an alert if the time delta exceeds a user-settable threshold.

Solution

This solution uses the fb_IndicatorTimeDelta function block to monitor the state of the Offline POU output pin of an SEL client. If a user-settable time period elapses before the Offline pin deasserts, the function block will output an alert. This example assumes that an SEL-735 SEL client named SEL_735_2_SEL was previously added to the RTAC project.

```
PROGRAM Go_online_timer
VAR
  TimeTracker : fb_IndicatorTimeDelta;
  Control : SPS;
  OfflineTrack : SPS;
  MaxAllowedTime : REAL := 30; //In seconds
  TimeToGoOnline : REAL;
  GoOnlineTimerAlert : BOOL;
END_VAR

//Set control variable
Control.q.validity := good;
Control.t := SYS_TIME();
Control.stVal := TRUE; //Control should always be true to ensure that the timer starts
//on the first cycle.

//Load variable to be monitored
OfflineTrack.q.validity := good;
OfflineTrack.stVal := NOT SEL_735_2_SEL_POU.Offline;
OfflineTrack.t := SYS_TIME();

//Run fb_IndicatorTimeDelta function block
TimeTracker(EN := TRUE,
RESET := FALSE,
Indicator1 := Control,
Indicator2 := OfflineTrack,
TimeDiffThreshold := MaxAllowedTime,
WaitTime := T#5M); //If OfflineTrack.stVal hasn't asserted after 5 minutes
//assume something else is wrong and stop the timer.

//If client has gone online, load the outputs
IF NOT SEL_735_2_SEL_POU.Offline THEN
  TimeToGoOnline := TimeTracker.TimeDifference;
  GoOnlineTimerAlert := TimeTracker.Alert.stVal;
END_IF
```
Creating an Object to Monitor the Rate of Remote Access Failures

Objective

Monitor the number of failed attempts to log-in to the RTAC and assert an HMI alarm if there have been more than five failed attempts within one minute.

Solution

This solution uses the fb_Derivative function block to monitor the Number_Of_Logon_Errors system tag and set an alarm if the rate of change in logon errors exceeds a settable threshold. This example assumes the following:

1. A virtual tag list called HMI_Controls was created for program control and status outputs. Virtual tag list tags shown in this example are defined as the following data types.
   - RemoteAccessTracker_Reset: operSPC
   - RemoteAccessAlarm: SPS
   - RemoteAccessAlarmDetails: STR

Code Snippet 6  prg_AuthenticationAlarm

```plaintext
PROGRAM prg_AuthenticationAlarm
VAR
  LoginErrorRateTracker : fb_ChannelDerivative;
  ErrorAccumulator : MV;
  Threshold : REAL := 0.08333; // In units of login failures per second. Equals 5 login failures divided by 60 seconds.
END_VAR

//Load the input Channel MV
ErrorAccumulator.q.validity := good;
ErrorAccumulator.instMag :=
  UDINT_TO_REAL(SystemTags.Number_Of_Logon_Errors.stVal);

//Run the fb_ChannelDerivative block
LoginErrorRateTracker(EN := TRUE,
  Reset := HMI_Controls.RemoteAccessTracker_Reset.status.stVal,
  Channel := ErrorAccumulator,
  DerivativeThreshold := Threshold,
  PeriodicProcessing := TRUE,
  Period := T#60S,
  FilterLength := 1
  Alert => HMI_Controls.RemoteAccessAlarm);

//Display alarm details while in alarm state, otherwise, clear alarm details.
IF LoginErrorRateTracker.Alert.stVal THEN
  HMI_Controls.RemoteAccessAlarmDetails :=
    SystemTags.Unsuccessful_Log_On_Attempt;
ELSE
  HMI_Controls.RemoteAccessAlarmDetails.strVal := '';
```

Date Code 20181001  Instruction Manual  ChannelMonitoring
## Release Notes

<table>
<thead>
<tr>
<th>Version</th>
<th>Summary of Revisions</th>
<th>Date Code</th>
</tr>
</thead>
</table>
| 3.5.1.1   | ➤ Allows new versions of ACSELERATOR RTAC to compile projects for previous firmware versions without SEL IEC types “Cannot convert” messages.  
➤ Must be used with R143 firmware or later.  
➤ Added fb_ChannelDerivative.  
➤ Added fb_ChannelIntegral.  
➤ Added fb_IndicatorTimeDelta. | 20180921  |
| 3.5.0.0   | ➤ Initial release.                                                                   | 20151223  |