Automating Protection System Monitoring and Verification With the SEL RTAC

Darrin Kite and Robin Jenkins

INTRODUCTION

The North American Electric Reliability Corporation (NERC), under the direction of the Federal Energy Regulatory Commission (FERC), is responsible for improving the reliability of the North American bulk electric system (BES). This responsibility includes creating a compliance program to improve the protection system reliability of generation and transmission facilities that can impact the BES. Under NERC PRC-005-2, the definition of protection systems includes protective relays, associated communications systems, voltage- and current-sensing devices (including their circuits), dc control circuitry, and station dc supplies associated with protection functions. Proposed future revisions of the NERC PRC-005 standard expand the scope of the maintenance programs to include additional protection system components.

Compliance with the standard entails the creation of a comprehensive protection system maintenance program and the execution of maintenance activities on a time- or performance-based interval or a combination of both. Leveraging the inherent self-test functionality and communications capability of modern intelligent electronic devices (IEDs), engineers can implement an automated system to perform real-time monitoring, verification, and reporting for power system components. This provides the following benefits [1]:

- Improves protection system awareness and identifies potential failed components that may otherwise go unnoticed.
- Supports maintenance and validation testing documentation requirements for each of the monitored critical protection system components.
- Documents the behavior of power system components during fault events and steady-state operations.

Beyond compliance, monitoring critical assets via predictive alarming gives engineers the necessary data to keep power system components online and avoid expensive downtime. This white paper describes how to use the SEL Real-Time Automation Controller (RTAC) to build an automated system to continuously monitor and validate many of the components identified in the NERC PRC-005 standard. This paper describes how to:

- Verify commissioned IED settings and firmware configurations.
- Verify CTs and PTs with continuous IED measurements.
- Verify communications-assisted protection channel integrity.
- Monitor IED diagnostics.
- Generate automated reports for both local and remote access.
- Identify and log maintenance conditions.
VERIFY COMMISSIONED IED SETTINGS AND Firmware Configurations

The Configuration Monitor is a feature in SEL clients that provides the capability to monitor ASCII command responses. With this feature enabled, SEL IEDs are continuously monitored to detect changes in configurations. Any ASCII command supported in the IED can be specified and sent on a periodic interval. When the RTAC receives a valid response, a 32-bit cyclic redundancy check (CRC) is calculated and stored in memory. If the RTAC receives a valid response in which the calculated CRC does not match the stored CRC, the Check_IED_Configuration_Mismatch POU pin pulses. Then, the most recent CRC is stored in memory. If the CRC cannot be calculated correctly, the Check_IED_Configuration_Error POU pin pulses. The settings for the Configuration Monitor are displayed by selecting the Advanced Settings check box in the upper-right corner of the IED view screen (as shown in Figure 1).

![Figure 1 Check IED Configuration Settings in an SEL Client](image)

The advanced settings are available to optimize the monitoring interval and command retry behavior. By default, messages are configured to allow users to monitor both the relay ID information and settings configuration. However, users should verify with the relay’s manual that the commands are supported in the IED firmware version. Users can click on the Check IED Configuration Commands tab to view and edit the default messages (as shown in Figure 2). Additional messages can be added at the user’s discretion. Also, by default, the Configuration Monitor is not enabled. To enable it, users click on the POU Pin Settings tab and change the Check_IED_Configuration POU pin to TRUE (as shown in Figure 3). In some instances, software flow control interferes with the Configuration Monitor. To avoid this, disable software flow control by toggling Xon\Xoff to False in the Communications section of the Settings tab.

![Figure 2 Default Messages Sent by the Configuration Monitor in the SEL Client](image)
acSELerator RTAC Library Extensions Introduction

The acSELerator RTAC® SEL-5033 Software library extensions are a set of libraries developed by SEL that offer additional functionality for programming applications in the RTAC’s feature-rich, integrated IEC 61131 programming environment. Two libraries are used in this white paper: 1) ChannelMonitoring and 2) FileIO. FileIO is a paid library, and an RTAC upgrade package can be ordered by contacting your local sales or customer service representative.

To start programming with libraries, visit the acSELerator RTAC software page at https://www.selinc.com/SEL-5033/ and download the acSELerator RTAC library extensions. Once the library extensions are installed, navigate to the IEC 61131-3 ribbon button in acSELerator RTAC and select Library (as shown in Figure 4). From here, a wide range of libraries are available, including the ChannelMonitoring and FileIO libraries. Note that users can download and use the libraries in acSELerator RTAC; however, if a paid library is uploaded to an RTAC without the correct model option table (MOT), a warning will appear that will prevent the project upload from completing.
Figure 4  Adding a Library Through the IEC 61131-3 Menu in acSELERATOR RTAC

Each library has a corresponding instruction manual that is accessible through the acSELERATOR RTAC help button ( ) drop-down menu in the upper-right corner of the screen (see Figure 5). These manuals are installed automatically with the library extensions.

Figure 5  Accessing Library Documentation From the acSELERATOR RTAC Help Menu

CHANNELMONITORING LIBRARY INTRODUCTION

The RTAC is a data concentrator and communications gateway that includes an integrated IEC 61131 programming environment. The RTAC leverages these capabilities to monitor and validate IED measurements in a substation. The ChannelMonitoring library aids engineers in simplifying power system maintenance monitoring. The library includes tools that are used to create a monitoring system that works on any substation topology. The function block outputs provide Boolean alerts. The status outputs return information about the function block’s current state. These outputs are assigned to enumerations provided in the library or to DINT data types. Functions are provided to convert the enumerated outputs to strings for logging purposes. For detailed information regarding the inputs, outputs, and information processing for the function
blocks, refer to the library’s instruction manual. In the following sections, brief examples are provided that demonstrate the functionality of each function block.

Example Assumptions

Communications protocol clients are configured in ACSELERATOR RTAC to communicate with IEDs. The phase measurements and the applicable relay status points are polled at user-defined polling intervals. These measurements and status points are passed to the function block inputs for processing. Input data are monitored by the function block, which determines if any significant difference is detected based on the user-specified parameters. Function block outputs are stored in tags configured in a virtual tag list (a virtual tag list is used in all of the following examples; however, any tag matching the data type of the function block output can be used to store the data). Example programs are constructed using Continuous Function Chart (CFC). Similar examples are provided in Structured Text (ST) in the library’s instruction manual. For additional information regarding any of the topics described in this paper, refer to [2], or view the RTAC videos on the SEL website for quick tutorial videos produced by SEL automation engineers [3].

The tags used in some of the following examples are complex measured value (CMV) data types. When an SEL IED is queried via SEL protocol, some of the data returned include both magnitude and angle; hence, a CMV data type is used to store that information. However, the function blocks that use analog quantities as inputs only accept measured value (MV) data types because, in most cases, the phasor magnitudes are compared to detect possible maintenance alert conditions. The following function logic converts a CMV data type to an MV data type through direct mapping of the magnitude quantity.

```plaintext
FUNCTION fun_CMV_TO_MV_Mag : MV
VAR_INPUT
  CMV_IN : CMV;
END_VAR;

fun_CMV_TO_MV_Mag.instMag := CMV_IN.instCVal.mag;
fun_CMV_TO_MV_Mag.q := CMV_IN.q;
fun_CMV_TO_MV_Mag.range := CMV_IN.range;
fun_CMV_TO_MV_Mag.rangeC := CMV_IN.rangeC;
fun_CMV_TO_MV_Mag.t := CMV_IN.t;
fun_CMV_TO_MV_Mag.zeroDb := CMV_IN.zeroDb;
fun_CMV_TO_MV_Mag.db := CMV_IN.db;
```

**VERIFY CTs and PTs With Continuous IED Measurements**

Two function blocks are provided in the ChannelMonitoring library that encapsulate logic to monitor MV data and determine if critical system components are operating as designed. The two function blocks alert on a test failure. The test failure indicates that either a sustained divergence between two inputs has occurred or repeated divergences were detected. Both function blocks also indicate the input information quality.
Example 1: Monitor for Deviations Between Phases on the High and Load Side of a Breaker

This example demonstrates the use of the fb_MultiChannelAlert function block. Both the high and load side of a breaker phase measurement are obtained from the IED. The IED has two terminals that are connected to two separate current transformers (CTs) located on both sides of a breaker, as shown in Figure 6. The function block monitoring is enabled when the high-side power measurements are greater than or equal to 5 percent of the nominal value. Phases are compared to detect a significant difference in measurements. The purpose is to detect any difference between the phases of each CT that may indicate a maintenance condition.

![Figure 6 RTAC Polling Data From an IED Monitoring High and Load Sides of a Breaker](image)

To configure this example, a CFC program (see the following PROGRAM Example1 logic) is added to the ACSELERATOR RTAC project [2] [3]. The code block shown in Figure 7 demonstrates how to instantiate two fb_MultiChannelAlert function blocks, each monitoring a three-phase CT, in the declaration section of the program. In the body of the program, the CMV data reported by the IED are converted to MV data types. A check is then performed to determine if the generator real power exceeds 5 percent of nominal before activating the instantiated fb_MultiChannel function blocks. Once activated, the function blocks compare all of the individual phases against one another to detect an abnormal condition.

```
PROGRAM Example1
VAR
  //Instantiate Function Blocks
  HighSide : fb_MultiChannelAlert;
  LoadSide : fb_MultiChannelAlert;

  //Function Block Parameters
  GeneratorNominal : REAL := 100;
  EnableGeneratorMonitoring : BOOL;
END_VAR
```
Example 2: Monitor for Deviations in the Phase A Measurement From Multiple IEDs

This example demonstrates the use of the \texttt{fb\_ChannelAlert} function block. Phase A measurements are obtained from four IEDs within the substation, as shown in Figure 8. The IEDs are connected to CTs located throughout the substation. Phase A measurements are compared against a chosen reference. By testing the measured values from an IED, the function blocks also provide validation for additional critical protection system components, including the CT circuits and the IED analog-to-digital converters. The monitoring function blocks enable when the reference measurement has good quality. The purpose is to detect any difference between CT measurements that may indicate a maintenance condition.

To configure this example, a CFC program (see the following PROGRAM Example2 logic) is added to the ACSELERATOR RTAC project\cite{2}\cite{3}. The code block shown in Figure 9 demonstrates how to instantiate three \texttt{fb\_ChannelAlert} function blocks, each monitoring a CT’s
Phase A measurements, in the declaration section of the program. As in the previous example, in the body of the program the CMV data reported by the IED are converted to MV data types. A check is then performed to determine if the reference value has good quality (i.e., the RTAC is receiving valid data from the IED) before activating the instantiated function blocks. Once activated, the function blocks compare the input channel and reference data to determine if an abnormal condition is detected.

```
PROGRAM Example2
VAR
    // Instantiate Function Blocks
    IED_Monitor1 : fb_ChannelAlert;
    IED_Monitor2 : fb_ChannelAlert;
    IED_Monitor3 : fb_ChannelAlert;

    // Function Block Shared Parameters
    ReferenceGoodQuality : BOOL;

END_VAR
```

**Figure 9 Example 2 Configured in a CFC Program**
VERIFY COMMUNICATIONS-ASSISTED PROTECTION CHANNEL INTEGRITY

Protection communications channels require high availability and reliability to ensure that critical signals are available to other substation IEDs that participate in coordinated protection and control schemes. Continuously monitoring these channels for communications errors provides information on the health of the channel. Early alerting on a potential maintenance condition can aid engineers in identifying conditions that may result in a misoperation.

Example 3: Monitor a MIRRORED BITS® Communications Channel for Data Transmission Errors

This example demonstrates the use of the fb_IndicatorAlert function block. The RTAC and an IED exchange critical control information over a serial MIRRORED BITS communications channel. The channel health is monitored by the RTAC’s built-in diagnostic information included in all communications instances. The Rx_OK_Instantaneous POU pin is monitored by the function block. This pin deasserts the instant any of several types of transmission errors are detected or any time that a MIRRORED BITS message is not received in the time required to transmit three messages. Monitoring this pin provides a good indication of channel health. Note in the example that a NOT block inverts the Rx_OK_Instantaneous POU pin because the fb_IndicatorAlert function block detects a TRUE condition. Because the goal is to monitor the deasserting of the pin, a NOT is included in the logic.

To configure this example, a CFC program (see the following PROGRAM Example3 logic) is added to the ACSELERATOR RTAC project [2] [3]. The code block shown in Figure 10 demonstrates how to instantiate an fb_IndicatorAlert function block in the declaration section of the program. Although not shown in Figure 10, the function block EN value can be tied to the ENO POU pin of a MIRRORED BITS communications instance. This ensures that the MIRRORED BITS channel is enabled before activating the instantiated function block. Once activated, the function block monitors for a TRUE condition, which indicates that errors were detected in the communications channel.

```
PROGRAM Example3
VAR
  //Instantiate Function Blocks
  MirrorBitsMonitor : fb_IndicatorAlert;

  //Function Block Shared Parameters
  EnableMonitoring : BOOL := TRUE;
END_VAR
```

![Figure 10  Example 3 Configured in a CFC Program](image)

MONITOR IED DIAGNOSTICS

Each SEL IED runs continuous self-tests to monitor the internal health of its major components. If an out-of-tolerance condition is detected, the relay generates a warning or failure alarm. When a self-test determines that one or more internal components have exceeded an expected limit but have not compromised the relay operation, a warning alarm is generated and a contact pulsed. For
a severe out-of-tolerance condition, a failure alarm is issued and the relay enters a protection-disabled state.

**Example 4: Monitor the HALARM Relay Word Bit in an SEL IED**

This example also demonstrates the use of the `fb_IndicatorAlert` function block. The RTAC is polling an IED for the HALARM Relay Word bit. Many SEL IEDs have additional configurable alerts that can also be monitored. Monitoring this Relay Word bit provides assurance that the IED is functioning as designed. Using both the chatter and sustained event capability of the function block, alerts can be generated when multiple warnings are received (which indicates an out-of-tolerance condition) or if a failure alarm is generated. This example only monitors the HALARM bit when communication with the IED is good.

To configure this example, a CFC program (see the following PROGRAM Example4 logic) is added to the ACSELERATOR RTAC project [2] [3]. The code block shown in Figure 11 demonstrates how to instantiate an `fb_IndicatorAlert` function block in the declaration section of the program. The **SEL IED Offline** POU pin is monitored to ensure that the IED is responding correctly to poll requests. If the **Offline** POU pin is **FALSE**, the function block enables monitoring. This ensures that a false condition does not trigger the alert output. Once activated, the function block monitors for a true condition, which would indicate a hardware warning or failure.

```plaintext
PROGRAM Example4
VAR
  // Instantiate Function Blocks
  SEL_451_HardwareMonitor : fb_IndicatorAlert;

  // Function Block Shared Parameters
  EnableMonitoring : BOOL;
END_VAR
```

![Example 4 Configured in a CFC Program](image)

**Generate Automated Reports**

Note that users must purchase the FileIO library option in the SEL RTAC for the example in this section to work.

Alert conditions are monitored and reported daily. An alert could be associated with an IED self-test, bad CT and or potential transformer (PT) data values, a relay firmware or setting change, or bad data quality. If a component alert is generated, the RTAC identifies which of the components generated the alert, logs the failure time, and includes the status of the failed component in the daily report. This is accomplished by continually monitoring and processing the data required to identify if a monitored component parameter exceeds the user-specified limits. The status of all monitored protection system components is documented in a single automated report that can be
downloaded through the RTAC web interface or sent to a centralized server via File Transfer Protocol (FTP).

**Example 5: Daily Maintenance Report Generation**

The logic in this example demonstrates how to create a report that can be downloaded through the RTAC's web interface, as shown in Figure 12, or sent to an FTP server. Sending the report via FTP is not shown in this example, but examples are included in the FileIO instruction manual.

![Figure 12 The RTAC Web Interface File Manager Menu Displaying Five Reports Generated Over a Five-Day Time Frame](image)

The following logic shows Example 5 configured in an ST program. The logic can be divided into four general steps:

1. Monitor the time that has elapsed since the last report.
2. Convert the data from the examples into human-readable strings.
3. Construct and format message strings to log in the report.
4. Trigger the report creation.

```plaintext
PROGRAM Example2
VAR
  //Report generated every interval: user-defined seconds
  ReportInterval : REAL := 86400; (*day*)

  //Variables used to control program flow
  NewDownload : BOOL := TRUE;
  GenerateReport : BOOL;
  DailyReportGenerated : BOOL;
  TimeSinceLastReport : REAL;
  LastReportGenerated : timestamp_t;

  //FileIO library LogDirectoryManager class
  MaintenanceReportManager : class_LogDirectoryManager(
    folderName := '/MaintenanceReports',
    logPostfix := 'ReportEx.txt',
    MaxFolderSize := 512000,
    maxNumFiles := 30,
    autoStartNewLogDaily := FALSE);
```

Date Code 20160111

SEL White Paper LWP0019-01
//Report messages
HighSideMessage : STRING(255);
LoadSideMessage : STRING(255);
IED_1_Ex2_Message : STRING(255);
IED_2_Ex2_Message : STRING(255);
IED_3_Ex2_Message : STRING(255);
ProtChMessage : STRING(255);
SEL451_HALARM_Message : STRING(255);
END_VAR

//Program generates a report based on a user-defined report interval
//Reset variables
IF NOT Simulation.Reset THEN
    GenerateReport := FALSE;
    DailyReportGenerated := FALSE;
ELSE
    DailyReportGenerated := TRUE;
END_IF

//Calculate time in seconds since last report
TimeSinceLastReport := TS_DIFF(System_Time_Control_POU.System_Time.value,
LastReportGenerated.value);

//Report trigger logic
IF NewDownload THEN
    NewDownload := FALSE;
    (*This resets the report time after a new project is downloaded *)
    LastReportGenerated := SYS_TIME();
ELSIF TimeSinceLastReport > ReportInterval THEN
    GenerateReport := TRUE;
END_IF

//Generate a report
IF GenerateReport THEN
    //Report Formatting
    HighSideMessage := '';
    HighSideMessage := Concat('$NHighSide Ex1', '$NMonitoring Enabled: ');
    HighSideMessage := Concat(HighSideMessage,
        BOOL_TO_STRING(VirtualTagList1.HighSide_ENO.stVal));
    HighSideMessage := Concat(HighSideMessage,
        '$NAlert: ');
    HighSideMessage := Concat(HighSideMessage,
        BOOL_TO_STRING(VirtualTagList1.HighSide_Alert.stVal));
    HighSideMessage := Concat(HighSideMessage,
        '$NStatus: ');
    HighSideMessage := Concat(HighSideMessage,
        fun_GetAlertString(VirtualTagList1.HighSide_Status.stVal));
    HighSideMessage := Concat(HighSideMessage,
        '$NChannelStatus: ');
    HighSideMessage := Concat(HighSideMessage,
        fun_GetChannelString(VirtualTagList1.HighSide_ChannelStatus.stVal));
    HighSideMessage := Concat(HighSideMessage,
        '$NQualityAlert: ');
    HighSideMessage := Concat(HighSideMessage,
        BOOL_TO_STRING(VirtualTagList1.HighSide_QualityAlert.stVal));
    HighSideMessage := Concat(HighSideMessage,
        '$NQualityStatus: ');
    HighSideMessage := Concat(HighSideMessage,
        fun_GetChannelString(VirtualTagList1.HighSide_QualityStatus.stVal));
    HighSideMessage := Concat(HighSideMessage, '$N');

    LoadSideMessage := '';
    LoadSideMessage := Concat('$NLoadSide Ex1', '$NMonitoring Enabled: ');
    LoadSideMessage := Concat(LoadSideMessage,
        BOOL_TO_STRING(VirtualTagList1.LoadSide_ENO.stVal));
    LoadSideMessage := Concat(LoadSideMessage,
        '$NAlert: ');
    LoadSideMessage := Concat(LoadSideMessage,
        BOOL_TO_STRING(VirtualTagList1.LoadSide_Alert.stVal));
    LoadSideMessage := Concat(LoadSideMessage,
        '$NStatus: ');
    LoadSideMessage := Concat(LoadSideMessage,
        fun_GetAlertString(VirtualTagList1.LoadSide_Status.stVal));
    LoadSideMessage := Concat(LoadSideMessage,
        '$NChannelStatus: ');
    LoadSideMessage := Concat(LoadSideMessage,
        fun_GetChannelString(VirtualTagList1.LoadSide_ChannelStatus.stVal));
    LoadSideMessage := Concat(LoadSideMessage, '$N');
LoadSideMessage := Concat(LoadSideMessage, "$NQualityAlert: ");
LoadSideMessage := Concat(LoadSideMessage, BOOL_TO_STRING(VirtualTagList1.LoadSide_QualityAlert.stVal));
LoadSideMessage := Concat(LoadSideMessage, "$NQualityStatus: ");
LoadSideMessage := Concat(LoadSideMessage, fun_GetChannelString(VirtualTagList1.LoadSide_QualityStatus.stVal));
LoadSideMessage := Concat(LoadSideMessage, "$N");
IED_1_Ex2_Message := '';
IED_1_Ex2_Message := Concat('$$NIED 1 Ex2', '$NMonitoring Enabled: ');
IED_1_Ex2_Message := Concat(IED_1_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL351_ENO.stVal));
IED_1_Ex2_Message := Concat(IED_1_Ex2_Message, '$NAlert: ');
IED_1_Ex2_Message := Concat(IED_1_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL351_Alert.stVal));
IED_1_Ex2_Message := Concat(IED_1_Ex2_Message, '$NStatus: ');
IED_1_Ex2_Message := Concat(IED_1_Ex2_Message, fun_GetAlertString(VirtualTagList1.SEL351_Status.stVal));
IED_1_Ex2_Message := Concat(IED_1_Ex2_Message, '$NQualityAlert: ');
IED_1_Ex2_Message := Concat(IED_1_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL351_QualtiyAlert.stVal));
IED_1_Ex2_Message := Concat(IED_1_Ex2_Message, '$N");
IED_2_Ex2_Message := '';
IED_2_Ex2_Message := Concat('$$NIED 2 Ex2', '$NMonitoring Enabled: ');
IED_2_Ex2_Message := Concat(IED_2_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL501_ENO.stVal));
IED_2_Ex2_Message := Concat(IED_2_Ex2_Message, '$NAlert: ');
IED_2_Ex2_Message := Concat(IED_2_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL501_Alert.stVal));
IED_2_Ex2_Message := Concat(IED_2_Ex2_Message, '$NStatus: ');
IED_2_Ex2_Message := Concat(IED_2_Ex2_Message, fun_GetAlertString(VirtualTagList1.SEL501_Status.stVal));
IED_2_Ex2_Message := Concat(IED_2_Ex2_Message, '$NQualityAlert: ');
IED_2_Ex2_Message := Concat(IED_2_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL501_QualtiyAlert.stVal));
IED_2_Ex2_Message := Concat(IED_2_Ex2_Message, '$N");
IED_3_Ex2_Message := '';
IED_3_Ex2_Message := Concat('$$NIED 3 Ex2', '$NMonitoring Enabled: ');
IED_3_Ex2_Message := Concat(IED_3_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL421_ENO.stVal));
IED_3_Ex2_Message := Concat(IED_3_Ex2_Message, '$NAlert: ');
IED_3_Ex2_Message := Concat(IED_3_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL421_Alert.stVal));
IED_3_Ex2_Message := Concat(IED_3_Ex2_Message, '$NStatus: ');
IED_3_Ex2_Message := Concat(IED_3_Ex2_Message, fun_GetAlertString(VirtualTagList1.SEL421_Status.stVal));
IED_3_Ex2_Message := Concat(IED_3_Ex2_Message, '$NQualityAlert: ');
IED_3_Ex2_Message := Concat(IED_3_Ex2_Message, BOOL_TO_STRING(VirtualTagList1.SEL421_QualtiyAlert.stVal));
IED_3_Ex2_Message := Concat(IED_3_Ex2_Message, '$N");
ProtChMessage := '';
ProtChMessage := Concat('$$Prot Ch Ex3', '$NMonitoring Enabled: ');
ProtChMessage := Concat(ProtChMessage, BOOL_TO_STRING(VirtualTagList1.ProtectionChannelMonitoring_1_ENO.stVal));
ProtChMessage := Concat(ProtChMessage, '$NAlert: ');
ProtChMessage := Concat(ProtChMessage, BOOL_TO_STRING(VirtualTagList1.ProtectionChannelMonitoring_1_Alert.stVal));
ProtChMessage := Concat(ProtChMessage, '$NStatus: ');
ProtChMessage := Concat(ProtChMessage, fun_GetAlertString(VirtualTagList1.ProtectionChannelMonitoring_1_Status.stVal));
ProtChMessage := Concat(ProtChMessage, '$N");
SEL451_HALARM_Message := '';
SEL451_HALARM_Message := Concat('$$SEL451 HALARM Ex4', '$NMonitoring Enabled: ');
SEL451_HALARM_Message := Concat(SEL451_HALARM_Message, BOOL_TO_STRING(VirtualTagList1.SEL451_HardwareMonitor_ENO.stVal));
SEL451_HALARM_Message := Concat(SEL451_HALARM_Message, '$NAlert: ');
SEL451_HALARM_Message := Concat(SEL451_HALARM_Message, BOOL_TO_STRING(VirtualTagList1.SEL451_HardwareMonitor_Alert.stVal));
SEL451_HALARM_Message := Concat(SEL451_HALARM_Message, '$NStatus: ');}
SEL451_HALARM_Message := Concat(SEL451_HALARM_Message, 
    fun_GetAlertString(VirtualTagList1.SEL451_HardwareMonitor_Status.stVal));
SEL451_HALARM_Message := Concat(SEL451_HALARM_Message, '$N');

//Write the formatted reports
MaintenanceReportManager.WriteLogEntryString(HighSideMessage);
MaintenanceReportManager.WriteLogEntryString(LoadSideMessage);
MaintenanceReportManager.WriteLogEntryString(IED_1_Ex2_Message);
MaintenanceReportManager.WriteLogEntryString(IED_2_Ex2_Message);
MaintenanceReportManager.WriteLogEntryString(IED_3_Ex2_Message);
MaintenanceReportManager.WriteLogEntryString(ProtChMessage);
MaintenanceReportManager.WriteLogEntryString(SEL451_HALARM_Message);

//Update report time
LastReportGenerated := SYS_TIME();

//Reset the function blocks
DailyReportGenerated := TRUE;
END_IF

//This has to be included every scan for the manager to process actions
MaintenanceReportManager.Run();

This example shows the creation of five daily reports (see Figure 13). Each daily report demonstrates an alert condition in one or more of the function blocks. The maintenance report includes the function block outputs from the previous examples and examples for monitoring the following protection system component attributes:

- Relay configuration.
- IED, CT (PT), and associated circuitry.
- Protection communications.
- Relay hardware alarms.
IDENTIFY AND LOG MAINTENANCE CONDITIONS

Tracking and archiving events is an important function provided by a data concentrator. In the event that a potential maintenance condition is detected, it can be logged into the Sequence of Events (SOE) record, which provides a nonvolatile storage location for maintenance alerts. The SOE log in the SEL RTAC can store up to 30,000 log items. These logs are accessible through the web interface or an ODBC connection and can be sent via Syslog protocol to a syslog server for archiving. This provides a mechanism to store maintenance report results without the addition of the FileIO library. For detailed information on how to log and collect SOE data into the SEL RTAC, refer to [2] and/or [4]. Both resources provide detailed instructions on logging SOE records and methods to retrieve that information.

CONCLUSION

Generator and transmission protection relays can be described as silent sentinels that only demonstrate their designed function when a protection event occurs. Typically, an event may not happen for an extended period of time. This can increase the possibility of a false operation or a failure to operate on a protection event due to an undetected critical protection system component failure, noncommissioned setting, or other potential issues. The RTAC helps mitigate these possibilities by continuously monitoring many critical protection system components, recording
their statuses through daily reports, and increasing protection system engineering awareness through data aggregation and logging tools.

REFERENCES


BIOGRAPHY

Darrin Kite received his Bachelor of Science in Renewable Energy Engineering from the Oregon Institute of Technology in 2012. Before joining Schweitzer Engineering Laboratories, Inc. (SEL) in 2012, he worked as an engineering intern at Bonneville Power Administration. He is presently working as an automation engineer in SEL research and development.