Automated Processing

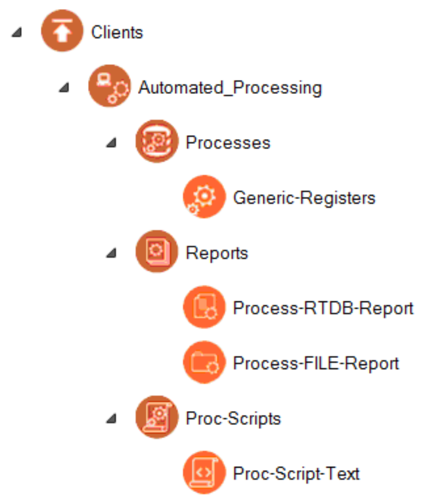
Documentation under development...

The **Automated Processing** section of the RediGate configuration is a collection of processes designed primarily to collect data from various sources (RTDB, text files) and format them into a "report" which may be published via MQTT, HTTP, and/or stored to the gateway for later retrieval.

Specifically, this feature allows the automation of data collection from various types of flow computers, such as Omni, Spirit, AutoPilot Pro, Control Microsystems, TotalFlow, or others. The Automated Processing section also includes miscellaneous features such as device time synch, PLC heartbeat, and other forms of script-driven logic, without requiring ISaGRAF or POD programming.

The Automated Processing configuration consists of the following elements:

- One or more **Process** objects, specifying a particular form of logic and data to act on (flow meter type, or other processing logic)
  (*under Clients Automated_Processing Processes*).
- One or more **Report** definitions, specifying the form of the output data to be generated (from RTDB or text file) and a process to publish the data
  (*under Clients Automated_Processing Reports*).
- Optional **Process Script**, allowing user-defined custom logic to assist in the report generation or publishing.
  (*under Clients Automated_Processing Proc-Scripts*).
- In most cases, the data will come from a Master Channel polling process, such as a Modbus Field Unit.
  Specific flow computers may require certain configuration objects under the Master Channel in order to function properly with the Automated Processing feature.
  See the RediGate Configuration Manual for details on other configuration properties.



(The Automated Processing feature became available in the RediGate beginning in 2019.)

Table of Contents - Automated Processing

Automated Processing Placeholder

The Automated Processing placeholder is the parent of all the other Process, Report, and Proc-Script definitions,

Attributes	Function
Object Type	Automated_Processing
Parent(s)	System Clients
Instance	Must be 0

Processes

The Processes object is the parent of all individual Process objects and contains a few general properties of the Automated Processing function.

There are many different types of child Process objects. Many contain device-specific functionality, such as methods for reading and writing Modbus registers to obtain an hourly or batch ticket from an Omni or AutoPilot Pro flow meter. Other Process types include more generic functionality for handing any sort of data based on customer-defined trigger conditions, such as the Generic Registers or Proc-User-Logic objects.

Each Process object includes a Data Process Table with 18 columns and one or more rows. Each row in each Data Process Table represents a specific set of "process" logic and in most cases results in a Report being generated. Although the details vary among the different process types, all of them are configured to reference a certain Master Channel number and Field Unit address.

For processes using an RTDB report type, a certain "trigger" condition may be used to indicate when to obtain data from the RTDB registers in that Field Unit and stored into the output report. For processes using a File report type, the data is obtained from a text file in the file system and either stored into RTDB registers or into the output report. In either case, the output report may be published by some means such as MQTT or stored locally. The output process may be just to call a script, which can perform other logic instead. Then, the next Process is examined for its "trigger" condition, and so on.

Attributes	Function
Object Type	Processes
Parent(s)	System Clients Automated_Processing
Instance	Must be 0

Properties	Values
Report Pacing	Delay (in milliseconds) each cycle through check the Process objects, and additionally a delay whenever a Process generates a published Report. The Report Pacing is intended to limit the pace of the Automated Processing, so as to limit consumption of system resources and to allow time for a report to be sent via MQTT before creating the next Report.
Report Directory	Location of output Report files to be generated. *Default option is "RAM Drive" (/tmp/director).*
Group ID	Configurable text field, which may be included in an MQTT topic for a Report using the ${GROUP} wildcard variable (see **Report Name** property of the Process-RTDB-Report or Process-FILE-Report objects). Group ID serves a similar function as the MQ-RBE "Console ID" or Sparkplug "Group Name" properties, to allow an MQTT topic level grouping of RediGate sites.
Spare	Spare property.
Reserved	Reserved for internal use.

Generic Registers Process

The Generic Registers process object can be used to extract any sort of data registers out of the RTDB, or data fields from a text file, and assemble them into a Report. It does not include any custom logic for handling specific flow computer models (Omni, AutoPilot, etc.). Device polling and/or Modbus SOS Table register manipulation would need to be done outside this Automation object, such as with PODs, etc.

Every row in the Process table links to a Report definition and creates a unique Report output based on a configurable trigger condition.

Note that for all Process objects, the Reports section and at least one Report definition must be defined before the **Data Process Table** can be configured.

Attributes	Function
Object Type	Generic-Registers
Parent(s)	System Clients Automated_Processing Processes
Instance	0

Data Process Table	Values
Channel	Master Channel used for process logic and obtaining RTDB data.
RTU	Field Unit address used for process logic and obtaining RTDB data.
Type	For the Generic Registers process, the options include: • Generic Data Block - generic process for creating Report out of RTDB or file data. • Ignore This Row - use this to temporarily disable one or more process rows.
CommStatReg	RTDB register address containing the first of 5 Field Unit communication status registers (register should contain 7=good, 0=bad comms). If the communication status is checked and found to be in a failed state, the processing on this row will be skipped. *Following the convention of the Field Unit's **Comm Status Holdreg**, use a 40,xxx register to point to the communication status register in the System Status RTU, or use a 30,xxx register to point to the communication status register in the Field Unit's own RTDB.* *Set CommStatReg to 0 to check protocol status of the Master Channel's protocol driver instead of a Comm Status Holdreg.* *Set CommStatReg to -1 to ignore the communication status. Process logic for this row will occur regardless of comms status.*
Report	Select one Report definition to define the structure of the output data report produced by this process row.

DataRegs	**If this row uses a Process-RTDB-Report type:** • If DataRegs is non-zero, that RTDB register in Channel/RTU is used as the "base register" for Report generation. The Report defines offsets from DataRegs. *This option may be used when more than one block of similar data is stored in the RTDB; thus, the same Report can be reused by setting DataRegs to the starting register of each block of registers. (e.g., Field Units 1 and 2 each have an analogous block of data at 40001-40020 and 40101-40120. Four rows can be defined in Generic Registers with different RTU (1 and 2) and DataRegs starting address (40001 and 40101), with each row calling the same Report that is defined with data elements at Offset 0 through 19).* • If DataRegs is zero, the Report must be defined using actual register addresses in the RTDB. *This option might be used, for example, if the one block of numbered addresses with similar data is used in multiple Channel/RTU/RTDB. (e.g., Field Units 1 through 10 each have an analogous block of data at 40001-40020. Ten rows can be defined in Generic Registers with different RTU address, DataRegs could be set to 0, and the Report could be defined with data elements at register 40001 through 40020).* **If this row uses a Process-FILE-Report type:** • DataRegs should be a String type RTDB register, where the Tag Name of the register contains the name of the text file containing data to process, plus "_CC_RRRRR" is automatically appended to the Tag Name (where CC is the Channel and RRRRR is RTU/Field Unit address configured on this row, each with leading zeros as needed). The contents of the DataRegs register is unused. *For instance, assume there is a text file containing ASCII data, stored with the filename of* `/tmp/director/MyData_01_0005`*. For this Process table row, configure Channel=1 and RTU=5. If DataRegs is configured as 48001, then the Channel 1, unit 5, register 48001 should have a Tag Name defined as "*`MyData`*". The above filename will be opened and used to generate the output Report.*
TrigMode	The TrigMode option selects a method for triggering the Report generation. • Compare **TrigReg to TrigValue REGISTER** (TrigValue field is a register address containing a value) Comparisons are: |= (not equal), == (equal0, >= (greater than or equal), or <= (less than or equal to TrigValue) • Compare **TrigReg to TrigValue CONSTANT** (TrigValue field contains a constant value for comparison) Comparisons are: |= (not equal), == (equal0, >= (greater than or equal), or <= (less than or equal to TrigValue) • **ANY CHANGE** in either the lower 16 bits or 32 bits of TrigReg register (change from the last time it was checked). This could be used, for instance, with the optional 32-bit Timestamp created by a Field Unit when new data is polled from a device or stored into the RTDB. • **Not Used** (trigger every cycle of looping through Automated Processing rows, limited only by the CommStatReg and the Report Pacing interval). • **Time Interval** - "MINUTES of TrigValue Interval" (TrigValue contains a constant number of MINUTES, or HOURS, or DAYS). • **COMPARE CRC-32** of DataRegs (On every loop through the Process rows, the block of registers starting at DataRegs is checked for changes from the last time. The "Count DataRegs" column tells how many sequential registers to check. DataRegs must be set to a non-zero RTDB register. Comparison is made by calculating a CRC-32 of the set of registers, and the CRC-32 value is stored to TrigReg, which must be a 32-bit integer register.)
TrigReg	RTDB address in Channel/RTU containing some value to trigger on. For "COMPARE CRC-32" option, TrigReg stores the CRC-32 value. TrigReg is unused for the "Not Used and "Time Interval" methods of TrigMode.
TrigValue	When using the "Compare REGISTER" TrigMode options, TrigValue should be a register address containing a value for the comparison. TrigValue is unused for the "Not Used" and "COMPARE CRC-32" options. Otherwise, TrigValue contains a constant integer used for the TrigMode comparison method, either to compare with the value in the TrigReg register, or containing an interval number of minutes, hours, or days to initiate Report generation.

TrigAdjust	If the Report generation logic is "triggered," the trigger value in the TrigReg can be modified based on the TrigAdjust option (e.g., to prevent the Report from being immediately triggered again). TrigAdjust options are: • **None** - no modification to TrigReg after Report generation. • **REMOTE INCREMENT** - Increment the trigger value and write it to remote Channel/RTU/TrigReg register. Set TrigWrap and Parm1 to handle roll-over. • **REMOTE DECREMENT** - Decrement the trigger value and write it to remote Channel/RTU/TrigReg register. Once the value decrements to zero, it will not be decremented further. • **REMOTE ZERO** - Set the trigger value to zero and write it to remote Channel/RTU/TrigReg register. • **RTDB INCREMENT** - Increment the trigger value and set the Channel/RTU/TrigReg RTDB register. Set TrigWrap and Parm1 to handle roll-over. • **RTDB DECREMENT** - Decrement the trigger value and set the Channel/RTU/TrigReg RTDB register. Once the value decrements to zero, it will not be decremented further. • **RTDB ZERO** - Set the trigger value to zero and set the Channel/RTU/TrigReg RTDB register. For the "REMOTE INCREMENT/DECREMENT/ZERO" options, the TrigReg register in the Channel/RTU/Field Unit RTDB is also set, in addition to writing to the remote device.
TrigWrap	If the TrigAdjust "INCREMENT" option is used, TrigWrap tells at what point the value should roll over to its low ("WrapTo") value. If TrigWrap is greater than 20000, it is treated as a register address pointing to a maximum "Increment" value. If less than 20001, it is a constant value. *For instance, if TrigWrap is set to 40001, and the register 40001 contains a value of 30000, then once the TrigReg register is incremented up to 30000, after the next Report trigger it will be set to the value in Parm1.*
Run	The Run value is unused in processing, but it can be used as part of the Report filename (MQTT topic) using the ${RUN#} variable.
Parm1	If the TrigAdjust "REMOTE/RTDB INCREMENT" option is used, Parm1 contains the "WrapTo" value. Once the value rolls over at its maximum limit (from TrigWrap), it will be set to the constant integer configured in Parm1. *For instance, this may be used to set the minimum roll-over value to 1 instead of 0, if a particular trigger requires it.*
COUNT DataRegs	If the TrigMode "COMPARE CRC-32" option is used, this value specifies the number of **sequential registers** starting at DataRegs that will be calculated for the CRC, to determine if any of the registers has changed. Maximum count of registers is 1000. *The registers used in the CRC-32 calculation begin at DataRegs and go sequentially through the COUNT of registers. If the Report that is generated includes registers that aren't in this COUNT of registers, they will not be checked for changes to trigger the next Report.*
Inhibit Reg	Unused
Inhibit Test	Unused
Proc-Logic	Unused
Comment	Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.

Reports Placeholder

The Reports placeholder is the parent object of all individual Report objects. At least one child Report object must be defined before setting the properties for a Process, because the Data Process Table references the Reports in a dynamic drop-down list.

Attributes	Function
Object Type	Reports
Parent(s)	System Clients Automated_Processing

Instance	Must be 0

Examples of Report Output Formats

Below are samples of the kinds of Report contents that can be created with the Automated Processes **Report** feature, using different Output Mode selections.

Binary

If the Report is defined with three rows, representing three values, such as registers 40001 (UINT16 value=1001=0x03E9, No swapping), 41001 (UINT32 value=100,000=0x000186A0, with Word Swap), and 42001 (REAL32 value=3.1415=0x40490E56, with Byte and Word Swap), a 10-byte file will be created with the numeric values indicated. "No Swapping" indicates using the bytes in little-endian order. The contents of the file (shown here in ASCII equivalents) would be:

```
E9 03 01 00 A0 86 40 49 0E 56
```

XML (as Attributes Only) - Value is given as a "Value=" attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT Name=”ReportName”>
    <field Name="40001", Value="9999", Units="degF" />
    <array Name=”1stArray”>
        <field Name="40002", Value="9999", Units="degC" />
        <1stObject>
            <field Name="MyFloat", Value="3.1415", Units="psi" />
            <field Name="MyString", Value="This is a String32 register", Units="none" />
        </1stObject>
    </array>
    <field Name="Calc-CRC-32", Value="0x0", Units="none" />
</REPORT>
```

XML (Values and Attributes) - Value is include between start and end <field> tags.

```
<?xml version="1.0" encoding="UTF-8"?>
<REPORT Name=”ReportName”>
    <field Name="40001", Units="degF" >9999</field>
    <array Name=”1stArray”>
        <field Name="40002", Units="degC" >9999</field>
        <1stObject>
            <field Name="MyFloat", Units="psi" >3.1415</field>
            <field Name="MyString", Units="none" >This is a String32 register</field>
        </1stObject>
    </array>
    <field Name="Calc-CRC-32", Units="none" >0x0</field>
</REPORT>
```

JSON

```
{
    "40001": 9999,
    "1stArray":
    [
        "40002":9999,
        "1stObject":
        {
            "MyFloat": 3.1415,
            "MyString": "This is a String32 register"
        }
    ]
    "Calc-CRC-32": 0x0
}
```

CSV (Two rows, multiple columns)

```
40001,40002,MyFloat,MyString,Calc-CRC-32
9999,9999,3.1415,This is a String32 register,0x0
```

TXT (Property=Value)

```
40001=9999
40002=9999
MyFloat=3.1415
MyString="This is a String32 register"

Calc-CRC-32=0x0
```

Process-RTDB-Report

The Process-RTDB-Report object generally defines the structure of a packet of data to be reported using MQTT or some other process on a non-real-time basis. The Report may be defined as either a binary (byte) structure or text format (CSV, JSON, XML, etc.). The Automation “Process” objects contain one or more table rows which are responsible for triggering on certain conditions and calling the Report generation, using data from the Channel/RTU, and then publishing the report using a defined publisher process (in most cases). A Report may be called by more than one Process table row.

Each Report includes a Report Name, which doubles as the MQTT topic that will be used if the Report file is published using MQTT. The Report may also specify a Process Script to be run, rather than a publisher process. The Process Script can perform a variety of actions on the Report file or Linux system commands after the Report has been created.

It is possible to define an empty Report with no table rows, which will create a blank file. For instance, this might be used in cases where the Report is being called by an Automated Process for the sole purpose of running a Process Script.

Attributes	Function
Object Type	Process-RTDB-Report object
Parent(s)	System Clients Automated_Processing Reports
Instance	Must be between 0 and 999

It is important to note that the Instance Number of a Report is used in the Data Process Table. If the Report object's Instance Number is changed after it has been used in a Process table, you must go back to the Process objects and ensure that the correct Report name is selected in the "Report" column of every row.

Properties	Values

Report Name	This property defines what filename will be used to store the Report contents after it has been triggered by the Process. It is also used as the MQTT topic name if the Report is published using MQTT.
	
	The Report Name may contain replacement variables, allowing the Report filename to be used by more than one Process, resulting in differently named files or MQTT topics. See below this table for a list of Report Name variables.
	
	Forward slash characters (/) in the Report Name will be converted to tilde (~) characters in the filename. Tilde (~) characters in the filename will be converted to a forward slash (/) in the MQTT topic when it is published.
	
	Any part of the filename before a double tilde (~~} will be excluded from the MQTT topic.
	
	For instance, if the Report Name is: ${TIME}~~MyData~${CHAN##}~${RTU#####}/Group${RUN#}/json
	then the Report will be saved to a filename something like:
	`/tmp/director/20180228T090537.000Z~~MyData~00~00001~Group2~json`
	`/tmp/director/20181231T235937.000Z~~MyData~03~00044~Group5~json`
	when triggered by a Process with Channel,RTU,Run that is configured for 0,1,2 or 3,44,5.
	
	The above files would be published via MQTT using the topic names:
	`MyData/00/00001/Group2/json`
	`MyData/03/00044/Group5/json`
	
	When using the ProcessScript option for Publisher Process, the Report Name (with full path) is passed into the script as the first command-line parameter.

Output Mode	Select a data format for the Report triggered by an Automated Process. Options are: • **Binary (no tag names)** - Output file contains packed bytes of raw data in the order and byte/word orientation defined by the Report. • **XML (as Attributes Only)** - Output file is XML text, starting with Report Header and with values contained in a "Value=" attribute. • **XML (Values and Attributes)** - Output file is XML text, starting with the Report Header and with values contained inside XML tags. • **JSON** - Output file is JSON text. • **CSV (Two rows, multiple columns)** - Output file is a two-line comma-separated text document, with the first column containing the tag names, and the second row containing the values. • **TXT (Property=Value)** - Output file is text document, with each line containing "Property=Value", where Property is the tag name of the register. • **HTML** - Output file is HTML text (not presently supported). See above for examples of each Report type using the Output Mode selection.
Publisher Process	Select what to do with the Report file once it has been generated. Options are: • **MQTT Client** - Publish using the MQclient3_1 process. • **MQTT-Extra Client 0** - Publish using the MQ_Extra_Client3_1 (instance #0). • **MQTT-Extra Client 1** - Publish using the MQ_Extra_Client3_1 (instance #1). • **HttpPost Client** - Publish via HTTP using the HttpPost object (not presently supported). • **SparkPlugB** - Publish via MQTT using the SparkplugB_RBE process. The Report Name must have the "@" symbol and a register address appended to the name (will be excluded from the MQTT topic), such as "@49001". The register must be a String-32 or String-256 type, which will be published to Ignition using that register's Tagname and the contents of the file as the tag data. • **Not publishing** - Create the Report file, but don't publish it. • **ProcessScript0** through **ProcessScript9** - Create the Report file, but instead of publishing it, run a Bash script. The Bash script may use the Bash script or Proc-Script-Text object and must have the script name configured as "ProcessScript#", where "#" is the number 0 through 9. The Process logic will attempt to run the script from /usr/director/bin (Bash scripts created in ACE are stored there, with an extension ".blnk" added). *When running a Bash script for the Report process, the name and location of the Report file is passed in as the first script parameter, followed by the contents of the Report Header field. These items may be used by the Bash script logic as optional command-line parameters that can be used within the script. For instance:* `/usr/director/bin/ProcessScript0.blnk` *+/tmp/director/MyData~03~00222~json Report Header Contents*

Report Header	When using an XML type Report, the Report Header will be used in the first line of XML text between the "<? ?>" tags, such as: `<?xml version="1.0" encoding="UTF-8"?>` When using the ProcessScript option for Publisher Process, the Report Header is passed into the script as command-line parameters, after the Report filename. This might be used by the Bash script as (space-separated) variables $2, $3, etc.
Number of Archives	When using a Report Name that allows for multiple Reports of the same type/instance to be created (i.e., using the ${TIME} variable), the Number of Archives determines how many copies to retain before eliminating old files, to avoid filling up the file system.
Definition Table	The Definition Table defines the content of the Report, which will generally consist of one data element (RTDB register) per row in the Definition Table. See below for a description of the RTDB Report Definition Table.

Note that every row in the Report Definition table represents one data value to be included in the generated Report. For text-based reports, each data item is identified with a Tagname, or register address if no Tagname is available. (In some cases, a table row represents a start or end of a sub-group or array, rather than a data value. JSON elements within an ARRAY include the value only, without a Tagname.)

RTDB Report Definition Table	Values
DataReg Offset	If the Process row that calls this Report uses a non-zero DataReg register address, then the Report will reference RTDB registers that are **offset** from that location. The DataReg Offset may be positive or negative (-65535 to 256,000). *For instance, if DataReg=40101, then if the Report includes rows with DataReg Offset = 0, 2, and -1, the Report data will be taken from RTDB registers 40101, 40103, and 40100.* If the Process row that calls this report has DataReg=0, then the Report must reference **absolute** RTDB register addresses. *For instance, if DataReg=0, then the Report's would reference DataReg Offset registers of 40101, 40103, and 40100 explicitly.*
SrcReg Count	Number of source registers starting at DataReg Offset which are concatenated together into a single data value. *If the data value is a 32-bit integer and comes from two 16-bit registers, the SrcReg Count=2. If the data value is a String and the ASCII values of the string come from multiple registers, SrcReg Count is the number of registers making up a single text string.*
Output Format	Format of data to be stored into the Report output (not necessarily the format of the source data registers). Formats include:

Type	Output Mode=Binary	Output Mode=text (JSON, CSV, etc.)
Boolean	Output a whole 8-bit byte of 0 or 1.	Boolean becomes a string value of "false" or "true", depending on a zero or non-zero source value.
Char-8	Store lower 8 bits of register (or a different set of 8 bits, depending on the Byte and Word swapping option	Don't use this format option – use the Text String format instead.
Unsigned or Signed Integer, 16-bit or 32-bit	Only used in ACE for visual indication of the source data type. If reading 16 bits from a 32-bit register, use a 16-bit type	Output either a 16-bit or 32-bit signed or unsigned number.
32bit Floating Point	Only used in ACE for visual indication of the source data type.	Output a floating point number.

Text String	• If source is STRING-32 register(s), store 32*SrcReg_Count bytes. • If the source register type is an integer, use SrcReg_Count registers and apply byte/word swapping.	• If source is STRING-32 register(s), concatenate SrcReg_Count registers together as one string. • For integer source register type, assume registers contain ASCII characters. Concatenate up to SrcReg_Count registers until reaching a Null character.
Hexadecimal Integer	n/a	Represent 16-bit or 32-bit source registers as ASCII hex bytes "00" through "FF" in the text Report.
Unsigned or Signed Integer, 64-bit	Same as above for 16-bit or 32-bit integers.	Same as above for 16-bit or 32-bit integers.
64bit Floating Point	Same as above for 32-bit floating point.	Same as above for 32-bit floating point.
Hexadecimal 64bit Integer	n/a	Same as above for 16-bit or 32-bit hexadecimal.
YYYY-MM-DDTHH:MM:00 from UINT32 EPOCH Seconds or (2) INT16	n/a	Store one UINT32 or two UINT16 registers as Linux ISO time with "00" seconds.
YYYY-MM-DDTHH:MM:00+TZN:00 from UINT32 EPOCH Seconds or (2) INT16	n/a	Same, with timezone.
YYYY-MM-DDTHH:MM:00 from (2)UINT32 Date(MMDDYYYY), Time(HHmm)	n/a	Store two UINT32 registers as Linux ISO time with "00" seconds (first register contains MMDDYYYY, second contains (HHmm).
YYYY-MM-DDTHH:MM:00+TZN:00 from (2) UINT32 Date(MMDDYYYY), Time(HHmm)	n/a	Same, with timezone.
YYYY-MM-DDTHH:MM:SS from (2)UINT32 Date(MMDDYY), Time(HHmmSS)	n/a	Store two UINT32 registers as Linux ISO time with "SS" seconds (first register contains MMDDYY, second contains (HHmmSS).
YYYY-MM-DDTHH:MM:SS+TZN:00 from (2) UINT32 Date(MMDDYY), Time(HHmmSS)	n/a	Same, with timezone.
YYYY-MM-DDTHH:MM:SS from Omni STRING-16 MM-DD-YYHH:mm:SS	n/a	Take date from String register in Omni text report format ("MM-DD-YYHH:mm:SS") and store to Report as Linux ISO time with "SS" seconds.
YYYY-MM-DDTHH:MM:SS+TZN:00 from Omni STRING-16 MM-DD-YYHH:mm:SS	n/a	Same, with timezone.
Text-8 Chars only Text-16 Chars only Text-24 Chars only Text-32 Chars only Text-48 Chars only Text-64 Chars only Text-128 Chars only Text-All Chars	Output text fields in fixed length (8, 16 characters, etc.) to ensure the binary Report payload fixed width is not exceeded for a string.	n/a - use Text String type.

NESTING START	n/a	***For JSON Report type***, create a start of JSON object, using text from the ReplaceName column as the element name. ***For XML Report type***, create a nested XML start tag using ReplaceName. *Examples:* *JSON: MyTag: {* *XML: <MyTag>* SEARCH-TEXT is ignored for this row. The object or tag should be closed with a "NESTING END" row. All rows in between will be nested data elements in the structure.
NESTING END	n/a	Close a JSON or XML nested structure (SEARCH-TEXT is ignored). *Examples:* *JSON: }* *XML: </MyTag>*
ARRAY START	n/a	***For JSON Report type only***: create a JSON array object by adding an opening square bracket (SEARCH-TEXT is ignored). This array should be closed with an "ARRAY END" row. All rows in between will be comma-separated elements in the array, with values only (no keywords/tag names).
ARRAY END	n/a	***For JSON Report type only***: add closing square bracket to an array. SEARCH-TEXT is ignored.
IP-ADDRESS	n/a	Take four numeric octets (from 32-bit integer, or two 16-bit integer registers) and output IP address in dotted decimal notation.

Swapping	Swapping is sometimes needed to reverse byte or word order of the source registers in order to produce the correct byte order for a Binary Report type, or to display the correct value for a text Report type. Swapping options are: • **No Swapping** (little-endian byte order, LSB first) • **Swap Bytes** (within 16-bit words) • **Swap 16-bit Words** (applies to 32bit registers only) • **Swap Bytes and Words** (big-endian byte order, MSB first) • **Swap DWords** (64bit data, change order of high and low 32-bit words) • **Swap DWords,Bytes** (64bit data) • **Swap DWords,Words** (64bit data) • **Swap DWords,Words,Bytes** (64bit data, bit-endian byte order, MSB first)
Name	Not used for Binary Report format. For text Report types (JSON, XML, CSV, TXT), the following options are used: • "+" indicates that the Tagname of the DataReg register will be used for identifying the data value. If no Tagname exists, the register number will be used for the tag. • Otherwise, the text in the Name column will be used to identify the data value.
Units	Units code (in text) is added to XML Report as an Attribute. Not used for all other Report types.
Scale_1 Scale_2 Scale_3	Scale fields are used for all Report types to apply engineering scaling before storing the value into the Report. Only one operation is allowed per field, but up to three Scale fields may be used, which are applied sequentially. Scale should be entered with operator first, then a number. Operators include: + (add), - (subtract), * (multiply), / (divide), or i (invert source value, then multiple by a number) *Examples (Scale_1, 2, and 3 fields are shown):* *Convert Celsius to Fahrenheit: *9 /5 +32* *Convert Fahrenheit to Celsius: -32 /9 *5* *Divide 5.9 over the register value, then add 20.3: i5.9 +20.3 ____*
Spare	Unused property.

Comment	Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.

Report Name Variables

The Report Name may contain a combination of fixed text and replacement variables, which are listed below. When the Report is created, the associated value will be substituted in the filename in place of the ${ } variable. Any variables shown with one or more “#” symbols indicate that a number will be substituted for the variable, with a fixed number of digits, using leading zeros where necessary.

Be careful when naming the Report, to make sure that multiple reports created by the Process objects yield a unique filename/topic, if it matters to your application.

- ${GATE} - Director/RediGate name configured in the System object.
- ${GATE#####} - Director/RediGate number configured in the System object (with leading zeros, if more than one #).
- ${RTU} - Field Unit name from the Channel/RTU in the Process row calling this Report.
- ${RTU#####} - Field Unit address from the Channel/RTU in the Process row (with leading zeros, if more than one #).
- ${CHAN} - Channel name from Process row.
- ${CHAN#####} - Channel number from the Process row (with leading zeros, if more than one #)
- ${ETH0} - IP address of eth0 (3 digits per octet), e.g. 010.020.030.040
  (presently, only ${ETH0} is supported, but other interfaces might be included in the future).
- ${SERIAL} - RediGate serial number, e.g. 54310-0001.
- ${RUN##} - Value from the "Run" column in the Process row calling this Report (with leading zeros, if more than one #).
- ${PARM1#} - Value from the Parm1 column in the Process row (may be named differently in some Process types).
- ${PARM2#} - Value from the Parm2 column in the Process row (may be named differently in some Process types).
- ${GROUP} - Substitute the Group ID from the "Processes" object.
- ${TIME} - Date/time of system, as "YYYYMMDDThhmmss.000Z" (e.g. 20181231T235937.000Z).
  Note: ${TIME} has to be the first element in the topic.Typically, it should be followed by double tilde, such as "${TIME}~~", so the time is excluded from MQTT publish topic.

Example of a ProcessScript

Below is one example of using a Process row to generate a Report file from RTDB, followed by calling a Bash script (ProcessScript#) instead of publishing the file directly. In this example, the following items are illustrated:

1. A Field Unit is configured with several registers pre-defined with Tagnames and values.
2. The Generic Registers automated process calls a Report with the "TXT (Property=Value)" text format.
3. The Report creates a text file, using register tag names and values for its data content.
4. After creating the Report, the ProcessScript0 is executed. In this example, the script uses the TXT Report as Bash script parameters, which is unique to the TXT report type (Property=Value pairs are 'source'd by the script, which turns the properties into Bash variables).
5. The ProcessScript executes some Bash logic and stores certain information into a different FieldUnit's RTDB.

Item 1

A Virtual RTU is defined under Internal Channel 15, Unit Address 3, having the following registers defined:

Register	TagName	Pre-initialized Value
40001		
48001	NEWFILE	

Item 2

A Generic Registers automated process is defined with Channel=15, RTU=3, TrigMode=MINUTES of TrigValue Interval, TrigValue=5. The Report is selected as "ScriptTest" (a Process-RTDB-Report type), defined below under Step 3.

[picture]

Every 5 minutes, on a fixed time interval, this process row will trigger the "ScriptTest" Report to be generated.

Item 3

The Process-RTDB-Report named "ScriptTest" (with Instance number 0, although it doesn't matter) is defined to take the registers listed above (Step 1) and create a text file with the Output Format "TXT (Property=Value)". The Report is defined as follows:

(name, report header, ProcessScript0, and table rows)

When the Report is created every 5 minutes, the data is stored into a text file /tmp/director/ScriptParms_15_00005. The contents of the file (unless the above registers are changed from their pre-initialized values) is:

(contents property=value)

Item 4

After creating the Report file, the ProcessScript0.blnk is executed, with the command:

/usr/director/bin/ProcessScript0.blnk (parms)

Item 5

(script contents and output)

Proc-Scripts Placeholder

The Reports placeholder is the parent object of all individual Report objects.

Attributes	Function
Object Type	Reports
Parent(s)	System Clients Automated_Processing
Instance	Must be 0

Proc-Script-Text

The Process-RTDB-Report object is

Attributes	Function
Object Type	Process-RTDB-Report object
Parent(s)	System Clients Automated_Processing Reports
Instance	Must be between 0 and 999

Properties	Values
Report Name	
Output Mode	
Publisher	
Report Header	
Number of Archives	