



Metal Tower

Remote Telemetry Unit (RTU)

RTU Rules Scripting

March 2006

© Metal Tower Pty Ltd

Table of Contents

Purpose of This Document	3
Variables	4
Variable Names	4
Predefined Constants	5
Numeric Constants	6
Using Port State in Rules Script.....	7
Testing Port State	7
Assigning Port Status	8
Statements	9
Assignment Statements	9
Assigning and Testing Port State	9
Conditional Statements	10
AND and OR Keywords.....	11
Syntax Definition	13
Verifying the Rules Script	15
How the Script is Executed	17
Errors Reported by the Rules Scripting Engine.....	18
Sample Scripts	22
Temperature Monitoring	23
Switching Between Primary and Backup Equipment.....	26
Multiple Rules	27

Purpose of This Document

This document describes the RTU rules scripting facility.

The RTU has a built-in rules script interpreter. This interpreter adds versatility to the RTU by allowing it to independently respond to events.

The rules syntax has been kept simple and is based on the IEC 1131-3 standard for PLC languages. Programming or coding experience is not needed in order to make effective use of the scripted rules facility.

The rules allow individual port states to be altered. This is normally done as a result of testing the state of other, related, ports or testing the values of ports (i.e. analog and temperature ports).

Variables

The RTU's scripting language does not allow variables to be created; the variables have already been created by the interpreter. Therefore these variables are always available to the script. The following types of variables exist:

- *State* variables. There is a state variable automatically associated with each port in the RTU. The RTU continually updates the values of these variables with the current states of their associated ports. In some cases it may also be possible to update some of these variables from the script, directly affecting the states of the ports associated with the variables
- *Value* variables. There is a value variable automatically associated with each port in the RTU. The RTU continually updates the values of these variables with the current value of their associated ports. These values are useful with analog and temperature ports. It is not possible to set the values of these variables / ports, therefore these variables are read-only.
- *Programmer* variables. A number of programmer variables are also available. There are both state and value programmer variables and these can be used as needed by the script. They are not used or updated by the rules engine except as directed by the script.

Variable Names

Because all the variables used by the rules engine are already created, their names are set by the rules engine and cannot be changed.

State variables are referenced by the tag of their associated ports. The tag must be entered as the letter that identifies the port type

followed by the port number as three digits. Therefore the state variable that refers to the first digital port is defined as D001; similarly the state variable that identifies the first temperature port is defined as T001.

Value variables are referenced similarly to status variables, except that "_VALUE" is appended to the name of the state variable in order to reference the value variable. For example the value of the first analog port is referenced by using the variable A001_VALUE; similarly the value of the first temperature port is referenced using the variable T001_VALUE.

The programmer variables can be thought of as a special type of "port". The port type letter for these "ports" is "P". Therefore the first programmer variable that can store a state is P001; the first programmer variable that can store a value is P001_VALUE.

Predefined Constants

Most scripts will work heavily with the state of the RTU ports. The RTU rules script engine therefore defines a number of constants that are designed to test and manipulate port states. The predefined state variables are:

- ALM refers to the alarm state of a port. Note that it is only possible to test if a port is in alarm, it is not possible to directly place a port into alarm.
- ARA refers to the ARA state of a port; HST refers to the historic state of a port.
- NRM refers to a (normal) condition where a port is neither in alarm, ARA, or historic.
- ACT refers to the active state of a port. When a digital input port is active it could in alarm or the active seconds period may not yet

have elapsed, in which case it would not be in alarm. An output port becomes active when the port is activated, either via an operator command or via the rules script.

- OFF refers to a state of an output port that is not active.
- NO_ARA is used to remove the ARA state from a port's status, returning it to the alarm state.
- NO_HST is used to remove the historic state from a port's state.

Numeric Constants

A numeric constant is entered by typing in the number that represents the constant value. Numeric constants are useful for comparing the values of analog and temperature ports to specific values.

Using Port State in Rules Script

Testing Port State

A port can have more than one state value at once. For example, when a digital port goes into alarm, the port has the following state values set:

- Active state.
- Alarm state.

A port must be active before it can go into alarm. If the *Active Seconds* setting is zero, the port will go into alarm immediately it goes active. If the setting specifies a number of seconds, the port will go into alarm after it has been active for the specified time. However even when it goes into alarm, the port still retains the active status value as well as taking on the alarm status value.

It is possible to test for any state value individually. The script engine will test for the specific state value regardless of the other state values that may be affecting the port. For example the following condition:

```
IF (D001 = ACT) THEN ....
```

will be *true* if the first digital port is active, regardless of its alarm, ARA or historic status.

The NRM (normal) status is a shortcut when testing an input port. If an input port is normal, then it is not in alarm or ARA state, and it is not in historic state either. Note that an input port is normal even if it is active because the active state is not part of the test for normal. Output ports are always normal and are never in alarm, ARA or historic states.

Assigning Port Status

When set a port's state is set by the script, the specific state value is set in addition to whatever other state values may exist for the port. For example if the historic state is assigned to a port, if the port was in alarm before the assignment then it remains in alarm but would now also be in historic after the assignment.

The NO_ARA status is used to remove the ARA state value from a port. The NO_HST status is used to remove the historic state value from a port. Not all states can be assigned to all ports. The following summarizes what is allowed:

- Analog ports can be assigned ARA, HST, NO_ARA, NO_HST.
- Digital input ports can be assigned ARA, HST, NO_ARA, NO_HST.
- Digital output ports can be assigned ACT, OFF.
- Temperature ports can be assigned ARA, HST, NO_ARA, NO_HST.
- Virtual input ports can be assigned ARA, ACT, HST, OFF, NO_ARA, NO_HST. Note that here NRM is a shortcut, equivalent to OFF and NO_HST.
- Virtual output ports can be assigned ACT, OFF.

Statements

There are only two types of statements in the scripted Rules syntax:

- Assignment, where a value is stored into / from a variable.
- Conditional, where a variable is tested and different code is executed depending on the variable's value.

All statements are terminated by a ';' semi-colon character.

Assignment Statements

A state or value is assigned using the ':=' assignment operator. For example a virtual port can be made active by assigning the **ACT** constant to the **V001** state variable using the following statement:

```
V001 := ACT;
```

If the script is used to acknowledge a specific digital port when it goes into alarm, the **ARA** state is assigned to the port's state variable like this:

```
D001 := ARA;
```

In order to store the value of an analog port for use later on in the script, assign it to one of the programmer value variables:

```
P001_VALUE := A001_VALUE;
```

Assigning and Testing Port State

A port can have more than one state value at once. For example, if a port goes into alarm and an ARA is subsequently issued on the port / unit, the port will have the following state values set:

- Active state.
- Alarm state.
- ARA state.

If the port subsequently goes out of alarm, then goes back into alarm, and an ARA is again issued on the port / unit, the port will have all of the above but will also have the historic state as well.

The script code can test for any of these states separately. Each test will check for that specific state regardless of the other states that may be affecting the port. For example the following condition

```
IF (D001 = ACT) THEN
```

will be *true* if the first digital port is active, regardless of whether it also happens to be in alarm / ARA / and/or historic state as well.

Similarly when setting a port's state, the specific state is being set in addition to whatever other states may exist for the port. For example if the script sets the historic port state using a statement like:

```
D001 := HST;
```

then if the port is in alarm it will remain in alarm but will now also be historic state as well.

Conditional Statements

Conditional statements are introduced by the **IF** keyword. This is followed by the condition that is being tested and then the **THEN** keyword. Every **IF** statement always ends with an **END_IF** keyword. For example, it is possible to test whether a virtual port is in alarm by using the following statement:

```
IF (V001 = ALM) THEN
    ... other statements ...
END_IF;
```

The statements that need to be executed when the condition is satisfied are inserted in between the **THEN** and **END_IF** keywords. For example, assuming the third digital port is set up for input and the second virtual port is set up for output, the script could activate the

second virtual port when the digital port goes into alarm by using code like this:

```
IF (D003 = ALM) THEN
    V002 := ACT;
END_IF;
```

In this situation it would usually also be necessary to switch off the virtual output when the digital input port is not in alarm. This is done by introducing the **ELSE** keyword inside the **IF** statement:

```
IF (D003 = ALM) THEN
    V002 := ACT;
ELSE
    V002 := OFF;
END_IF;
```

Multiple ports and conditions can be tested using the **AND** and **OR** keywords, for example:

```
IF (D003 = ALM) OR (D003 = ARA) THEN
    V002 := ACT;
ELSE
    V002 := OFF;
END_IF;
```

or perhaps

```
IF (D003 = ALM) AND (D004 = ALM) THEN
    V002 := ACT;
ELSE
    V002 := OFF;
END_IF;
```

AND and OR Keywords

The **AND** and **OR** keywords have very specific meaning in the scripting language.

When an **AND** operator is used, if either of the operands is *false* then the result of the AND operation is also *false*.

Following is a truth table for the AND operator.

Operand 1	Operand 2	Operand 1 AND Operand 2
True	True	True
True	False	False
False	True	False
False	False	False

When an **OR** operator is used, if either of the operands is *true* then the result of the OR operation is also *true*.

Following is a truth table for the OR operator.

Operand 1	Operand 2	Operand 1 OR Operand 2
True	True	True
True	False	True
False	True	True
False	False	False

Syntax Definition

The full syntax of the RTU Rules is defined below using the following Extended Backus Naur Form (EBNF) notation. In EBNF:

- Text in double quotes (") are literals keyed in as-is.
- The pipe (|) character indicates choice.
- Parentheses ((and)) group items together.
- Brackets ([and]) indicate optional items.
- Braces ({ and }) indicate repetition 0 or more times.

```
PROGRAM = STATEMENT ";" { STATEMENT ";" }
```

```
STATEMENT = IF_STATEMENT | ASSIGN_STATEMENT
```

```
IF_STATEMENT = "IF" EXPRESSION "THEN" STATEMENT {
["ELSIF" STATEMENT] } ["ELSE" STATEMENT]
```

```
EXPRESSION = SIMPLE_EXPRESSION [ "<" | "<=" | ">"
| ">=" | "=" | "<>" SIMPLE_EXPRESSION ]
```

```
SIMPLE_EXPRESSION = [ "+" | "-" ] TERM [ "+" |
"- " | "OR" TERM ]
```

```
TERM = FACTOR [ "*" | "/" | "AND" FACTOR ]
```

```
FACTOR = STS_VAR | VAL_VAR | NUM_VAL | "("
EXPRESSION ")" | "NOT" FACTOR | "TRUE" | "FALSE"
```

```
ASSIGN_STATEMENT = STS_VAR | VAL_VAR ":" EXPRESSION
```

```
STS_VAR = (PORT_TYPE PORT_NUM) | STS_CONST
```

```
STS_CONST = "ALM" | "ARA" | "HST" | "NRM" |
"ACT" | "OFF" | "NO_ARA" | "NO_HST"
```

```
VAL_VAR = (PORT_TYPE PORT_NUM "_VALUE") | NUM_CONST
```

```
PORT_TYPE = "A" | "D" | "T" | "V"
```

```
PORT_NUM = DIGIT DIGIT DIGIT
```

```
NUM_CONST = DIGIT { DIGIT }
```

```
DIGIT = "0" | "1" | "2" | "3" | "4" | "5"  
| "6" | "7" | "8" | "9"
```

Verifying the Rules Script

The rules script is entered into the RTU via one of the external interfaces. The nature of the scripting language does not allow for the script to be entered via the front panel.

Once the script is entered into the RTU, the RTU carries out syntactical and semantic checks.

A syntax check verifies that the rules for placing keywords and variables in the correct sequence have been followed correctly. A syntax check will pick up such things as forgetting to put a `;` (semi-colon) at the end of a statement, or omitting important keywords such as THEN in an IF statement.

A semantic check checks that some of the operations being attempted are valid. For example a syntax check will allow script to assign an alarm state to a digital input port using a statement like

```
D001 := ALM;
```

because the above statement follows the rules for an assignment statement: a port name ('D001') followed by the assignment operator (':=') followed by a port state ('ALM') followed a semi-colon (;). However the above statement will fail the semantic check because assigning an alarm state is not permitted.

Errors in syntax or script are reported back to the external environment through the status update XML that is retrieved using

```
http://<RTU\_IP\_Address>/rtustat.xml
```

The XML retrieved by this command has 3 attributes in the <RTUSTAT> tag that relate to the rules script:

- RL is "1" if the rules script is valid, or "0" if it is not able to be executed.
- RM gives an error code for the rules script.
- RP gives the zero-based index of the token causing a problem with the rules script.

Note that an error value is returned if there is no script. Similarly, an error value is also returned if a script is entered into the RTU but it is disabled.

How the Script is Executed

The RTU scans all the ports being monitored several times a second. In each scan it checks the value of each port to determine whether the port is still in the same state or has gone into alarm or some other state, etc.

At the end of every scan, the RTU checks if there is a valid rules and if it is enabled. If a script is present and enabled, the RTU executes the script.

The script is therefore executed several times a second. The RTU immediately acts on any changes made to port states by the script. Any changes remain in effect until changed again by a subsequent execution of the script or by external means.

Errors Reported by the Rules Scripting Engine

The following table gives a list of the error numbers reported by the rules scripting engine with a brief explanation of each error.

Error	Explanation
Error 1: Rules Disabled	The Rules Enabled setting is disabled.
Error 2: No Code	There is no script to process.
Error 3: Incomplete Program	The RTU has received part of a script, but not the complete script.
Error 4: Expected ELSIF, ELSE, Or END_IF Keyword	The scripting engine processing an IF statement but has come across a word that does not belong to the IF statement syntax. The word could be a keyword that belongs to another statement or an incorrectly spelled word.
Error 5: Expected IF Keyword	The scripting engine is expecting to find an IF-keyword but has found a different keyword.
Error 6: Expected Semi-	Each scripting statement must end with a semi-colon (;) symbol. The scripting engine has

Colon Symbol	detected the end of a statement but has not found the semi-colon character that marks the end of the statement.
Error 7: Expected Assignment Symbol	The scripting engine is expecting to find an assignment operator (':=') but has found a different symbol instead.
Error 8: Expected Close Bracket Symbol	Each opening bracket ('(') character must have a matching closing bracket (')') in the same expression. The scripting engine has come across the end of an expression with one or more unmatched brackets.
Error 9: Expected THEN Keyword	The scripting engine requires a THEN keyword for every IF statement. The engine has come across a variable or keyword without encountering the THEN keyword after an IF statement has been opened.
Error 10: Expected Variable	An unexpected symbol has been found where the scripting engine was expecting to find a variable. This would most likely be the variable to which an assignment is being made (the LValue).
Error 11: Invalid Port Number	A port has been addressed that does not exist in the RTU.
Error 12: Invalid Factor	The scripting engine is expecting a Factor symbol but has come across an inappropriate keyword or symbol. A Factor can be either a

	variable, a State constant, an Opening Bracket, the NOT keyword, or an unsigned number.
Error 13: Invalid LValue	This error is generated for assignment statements where the variable to which an assignment is being made (the LValue) is invalid. For example it is not allowed to assign values to port value variables
Error 14: Invalid RValue	This error is generated for assignment statements where a value being assigned is not allowed. For example it is not allowed to assign an alarm state to a port.
Error 15: Invalid Number Format	A numeric value has not been entered correctly or interpreted correctly by the scripting engine.
Error 16: Invalid State Value	A port state has not been entered correctly or interpreted correctly by the scripting engine.
Error 17: Invalid Variable Format	A program variable has not been entered correctly or interpreted correctly by the scripting engine.
Error 18: Type Mismatch, Boolean Required	The scripting engine is expecting a value of type Boolean (i.e. a value that evaluates to <i>true</i> or <i>false</i>).
Error 19: Type Mismatch, Numeric Required	The scripting engine is expecting a value of type Numeric (i.e. a value that evaluates to a number).

Error 20: Type Mismatch, Same Data Type Required	The scripting engine has come across a situation where it is expecting a value to be of the same type as other values in the same expression.
Error 21: Type Mismatch, State Required	The scripting engine is expecting a value of port state type (i.e. a value that evaluates to one of the port states).
Error 22: Unexpected Program End	The scripting engine is processing a statement, but no more program code can be found to complete the statement.
Error 23: Unknown Data Type	A value has not been entered correctly or interpreted correctly by the scripting engine.
Error 24: Unknown Variable	A program variable has not been entered correctly or interpreted correctly by the scripting engine.
Error 25: Too Many Iterations	The scripting engine uses up valuable processing time in evaluating a script. This error is generated if the engine detects it has spent too much evaluating a script.

Sample Scripts

The sample scripts shown below assume an RTU set up with the following port settings:

Port	Name	Direction	Active Secs	Add. Info
D001	COOLING FAN	OUT	0	
D005	COOLING FAN	IN	0	
D011	MAIN DEV A	OUT	0	
D012	BACKUP DEV A	OUT	0	
D013	MAIN DEV B	OUT	0	
D014	BACKUP DEV B	OUT	0	
D015	MAIN DEV C	OUT	0	
D016	BACKUP DEV C	OUT	0	
A001	AIRCO DUCT	IN	0	
A002	RACK TOP TEMP	IN	0	
A003	RACK BOTTOM TEMP	IN	0	
T002	ROOM TEMP WARNING			High trigger = 22
V001	ROOM TEMP ESCALATED	IN	0	
V002	ROOM TEMP CRITICAL	IN	0	

V003	AIRCO PROBLEM	IN	0	
V007	SWITCH TO MAIN RACK	OUT	0	
V008	SWTCH TO BACKUP RACK	OUT	0	

Temperature Monitoring

The following script activates a cooling fan when the room temperature alarm is triggered.

<pre> IF T002 = ALM THEN D001 :- ACT; ELSE D001 := OFF; END_IF; </pre>	<p>If the room temperature sensor goes into alarm, activate D001 to turn on a cooling fan. Turn the fan on when the sensor comes out of alarm.</p>
--	--

The following script extends the previous one by introducing some feedback. In this scenario the cooling fan is wired into the D005 digital input port. This input becomes active when the fan is rotating at a reasonable speed. The script uses this information to automatically acknowledge the temperature alarm, indicating that something has been done about it.

<pre> IF T002 = ALM THEN D001 :- ACT; </pre>	<p>If the room temperature is</p>
--	-----------------------------------

<pre>IF (D005 = ACT) THEN T002 := ARA; ELSE T002 := NO_ARA; END_IF; ELSE D001 := OFF; END_IF;</pre>	<p>in alarm but the cooling fan is in operation, acknowledge that the alarm is being dealt with. If the cooling fan is stopped, remove the ARA state from the alarm.</p>
---	--

The following script extends the previous one further. This script escalates the alarm (by introducing a new one and removing the temperature's ARA state) if the temperature continues to rise.

<pre>IF T002 = ALM THEN D001 :- ACT; IF T002_VALUE > 25 THEN V001 := ACT; T002 := NO_ARA; ELSE V001 := OFF; IF (D005 = ACT) THEN T002 := ARA; ELSE T002 := NO_ARA; END_IF; END_IF; ELSE V001 := OFF; D001 := OFF; END_IF;</pre>	<p>If the room temperature exceeds 25° then activate V001 (which will raise an alarm) and remove the ARA state from the temperature alarm. Turn off the escalated alarm if the temperature falls below 25°.</p>
--	---

The next script extends the scenario a little further by raising another alarm if the temperature keeps rising, in order to highlight that the situation has become critical.

<pre>IF T002 = ALM THEN D001 := ACT; IF T002_VALUE > 28 THEN V002 := ACT; V001 := NO_ARA; T002 := NO_ARA; ELSIF T002_VALUE > 25 THEN V002 := OFF; V001 := ACT; T002 := NO_ARA; ELSE V002 := OFF; V001 := OFF; IF (D005 = ACT) THEN T002 := ARA; ELSE T002 := NO_ARA; END_IF; END_IF; ELSE V002 := OFF; V001 := OFF; D001 := OFF; END_IF;</pre>	<p>If the room temperature continues to rise above 28°, activate V002 to immediately raise a new alarm; also remove any ARA state that may have been applied to the escalated V001 and T002 alarms.</p>
--	---

The following script introduces an additional check. A temperature probe has been placed right at the lip of the air conditioning duct and wired to input analog port A001. Other temperature probes have been wired to analog ports A002 and A003 and placed at the top and bottom of a rack respectively.

In this particular environment, when the air conditioning is working, the probe at the duct is normally 5° cooler than the probe at the top of the rack. If the air conditioning stops the probe in the duct warms up very quickly. Therefore a difference in temperatures of 2° or less is an indication that the air conditioning has failed. The difference in temperature between the top and bottom of the rack is normally 3°, so if this difference increases then it also is a clear that air is not being blown clear of the rack.

<pre> IF (A002_VALUE - A001_VALUE) < 2 THEN V003 := ACT; ELSIF (A003_VALUE - A002_VALUE) > 5 THEN V003 := ACT; ELSE V003 := OFF; END_IF; </pre>	<p>Compare the temperature at the duct with the temperature at the bottom of the rack. Also compare the temperature at the bottom of the rack to the temperature at the top of the rack.</p>
---	--

Switching Between Primary and Backup Equipment

The following script assumes that devices can be switched from a main rack to a backup rack through user intervention. The RTU has been configured so that ports V007 and V008 are output ports, so they can be activated from the Configuration or Operations page. The following script reacts to one of these ports being activated.

<pre> IF V007 = ACT THEN D011 := ACT; D013 := ACT; D015 := ACT; D012 := OFF; D014 := OFF; </pre>	<p>If V007 is activated then switch on all devices in the main rack, then deactivate V007; if V008 is activated</p>
--	---

<pre>D016 := OFF; V007 := OFF; ELSIF V008 = ACT THEN D012 := ACT; D014 := ACT; D016 := ACT; D011 := OFF; D013 := OFF; D015 := OFF; V008 := OFF; END_IF;</pre>	<p>then switch on all devices in the backup rack, then deactivate V008.</p>
---	---

Multiple Rules

Multiple rules can be coded one after the other in a script. They will be executed in the sequence they are coded. The following example checks primary and backup devices making sure that only one or the other, but not both, is active at any one time.

<pre>IF D011 = ACT THEN D012 := OFF; ELSE D012 := ACT; END_IF; IF D012 = ACT THEN D011 := OFF; ELSE D011 := ACT; END_IF; IF D013 = ACT THEN D014 := OFF; ELSE D014 := ACT; END_IF;</pre>	<p>Check that for each device, either the device in the primary rack or the device in the backup rack, but not both, is active.</p>
--	---

```
IF D014 = ACT THEN
    D013 := OFF;
ELSE
    D013 := ACT;
END_IF;
IF D015 = ACT THEN
    D016 := OFF;
ELSE
    D016 := ACT;
END_IF;
IF D016 = ACT THEN
    D015 := OFF;
ELSE
    D015 := ACT;
END_IF;
```