




Greenleaf ViewComm® System

Serial Data Monitors & Protocol Analyzers
for Windows®

ProtocolWorks™ Methods Reference



| | |
|-------------|---|
| IP | 1 |
| TCP | 0 |
| ICMP | 0 |
| PPP | 1 |
| SNMP | 1 |
| UDP | 0 |
| GGP | 1 |
| ARP | 1 |
| HTTP | 1 |
| DNS | 0 |
| IGMP | 0 |
| IPX | 1 |
| FRAME RELAY | 0 |
| NETBIOS | 0 |
| LCP | 1 |
| IPCP | 0 |
| PAP | 0 |
| ... | 1 |

Group: BOOLEAN

Method: CurrentDataMatches

Description:

Returns true if the current data has the value of the "Value to match" parameter. This allows you to check the value of data at the pointer without explicitly decoding it with a FIELD statement. It requires two parameters: 1) the number of bits (1 to 64) to be extracted from the data for comparison and 2) the value to be matched against.

This method is particularly useful in two situations. The first is in a GROUP statement to check a field before decoding it. You might, for example, have a field that is defined as being a four-byte field if the first byte contains hex FF and a two-byte field otherwise. You can see examples in the SMB decoder.

The second case is where you want to decode a field only if it contains a certain value. Consider, for example, this pair of statements from the NCP decoder:

```
FIELD printer_off_line (Fixed 1) IF (CurrentDataMatches 8 0xff) (Constant "Halted") "Printer Status"  
FIELD printer_on_line (Fixed 1) IF (CurrentDataMatches 8 0x00) (Constant "Not Halted") "Printer  
Status"
```

Here a one-byte field is deemed to exist and is decoded only when its value is zero or hex FF.

Parameters:

| | | |
|------------|-------|---------------------------------------|
| [Required] | Int | "Number of bits to match (from 1-64)" |
| [Required] | Int64 | "Value to match" |

Method: Dce

Description:

Returns true if the frame comes from the DCE side. This method has relevance only for data received from a serial type connection.

Parameters:

None

Method: Dte

Description:

Returns true if the frame comes from the DTE side. This method has relevance only for data received from a serial type connection.

Parameters:

None

Method: FieldIs**Description:**

Returns true if the statement is true. The statement compares the value in "Field to test" against a specified value using the given operator.

Here is a typical example taken from the X.25 decoder (example has been edited):

FIELD called_addr (Fixed 2) IF (FieldIs GreaterThan called_addr_lgh 0) (StringOfBcd) "Called Address"

In this case a field exists only when its length (called_addr_lgh) is non-zero.

Parameters:

| | | | |
|------------|-------|-----------------|---|
| [Required] | List | "Operator" | |
| | | List values: | GreaterThan GreaterThanOrEqualTo LessThan LessThanOrEqualTo EqualTo NotEqualTo |
| [Required] | Int64 | "Value" | |
| [Required] | Field | "Field to test" | |

Method: FieldIsBetween**Description:**

Returns true if the value in the "Field to test" is between the specified low and high values. This test is inclusive. In other words, if the value in the "Field to test" is equal to either the low or high value, the method will return true.

Parameters:

| | | |
|------------|-------|-----------------|
| [Required] | Int64 | "Low value" |
| [Required] | Int64 | "High value" |
| [Required] | Field | "Field to test" |

Method: IndexedPersistentStaticExists**Description:**

This method checks to see if the static data exists. In other words, it checks to see if the static data value specified has been encountered yet or not. In the case of the IndexedPersistentField, it checks to see if the index value is good. It returns true if the value is good, and false if the value isn't there. This is useful in cases where you have saved a field value with an index, but the index field hasn't been encountered yet. Using the example from the StoreIndexedPersistentField method, this method would check to see if a command to the slave specified in the parameter has been seen. If not, you can do something appropriate.

Parameters:

[Required] StaticName "Static data"
[Required] Field "Field that contains the index"

Method: IntraframeStaticExists

Description:

This method checks to see if the Intraframe static data exists. In other words, it checks to see if the field value you want to retrieve has been encountered in this frame yet or not. It returns true if the value is good, and false if the value isn't there. This is useful in cases where you have saved a field value, but that field may not be encountered in every frame. You can use this method in conjunction with the IntraframeField method to see if the data exists and do something appropriate if it doesn't, like skip the field, or decode something else instead.

The method doesn't take a parameter, but you do need to specify which field value you're checking for. The syntax is:
IF (IntraframeStaticExists field_name).

This example uses the boolean to skip the field if the static data doesn't exist.

```
FIELD my_field_2 (Fixed 0) IF (IntraframeStaticExists field_name) RETRIEVE (IntraframeField field_name) (Decimal) "Field Name"
```

Parameters:

[Required] StaticName "Static data"

Method: MatchTableData

Description:

Returns true if the table data matches the parameter "Value to match".

A typical use of this method is to distinguish various classes of message, for example commands, responses and data. Examples of its use can be seen in the HTTP and SMTP decoders. One instance from the latter decoder is (the table has been edited for clarity):

```
TABLE cmd_reply  
0x48454c4f "HELO" "Hello" 1 hi  
0x4d41494c "MAIL" "Mail" 1 from  
0x51554954 "QUIT" "Quit" 1 end  
0x32313120 "Status" "System status, or system help reply" 2
```

```
0x32313420 "Help" "Help message" 2
Default "" "default" 3
ENDTABLE
```

FIELD type (Fixed 4) (Hex) SUPPRESS_DETAIL NO_MOVE

GROUP cmd IF (MatchTableData cmd_reply type 1)

This takes the value in the field "type" and looks it up in the table "cmd_reply". Then the table data for that entry is checked against the constant given in the parameter "Value to match". For example, if the value of "type" were 0x48454c4f, the test would return true because this value matches the "HELO" entry in the table, and the table data for "HELO" is 1, which matches the 1 given as the parameter.

Parameters:

| | | |
|------------|-------|------------------|
| [Required] | Table | "Which table" |
| [Required] | Field | "Which field" |
| [Required] | Int64 | "Value to match" |

Method: MoreBytes

Description:

Returns true if there are more bytes to be decoded in the layer, and false if the current bit is at the end of the layer. If the current pointer is not at a byte boundary it is moved up to the next byte boundary for the purposes of the calculation. The parameter "Offset from end in bytes" specifies how many bytes at the end of the layer to disregard when determining if there are more bytes remaining. This is useful if you want to ignore a CRC, for example.

This method is often used to test for optional fields at the end of a layer.

Parameters:

| | | | |
|------------|-----|----------------------------|--------------------|
| [Optional] | Int | "Offset from end in bytes" | (Default value: 0) |
|------------|-----|----------------------------|--------------------|

Method: PersistentStaticExists

Description:

This method checks to see if the static data exists. In other words, it checks to see if the static data value specified has been encountered yet or not. It returns true if the value is good, and false if the value isn't there. This is useful in cases where you have saved a field value, but a frame that uses that field hasn't been encountered yet. You can use this method in conjunction with the PersistentField method to see if the data exists and do something appropriate if it doesn't, like skip the field, or decode something else instead.

Parameters:

| | | |
|------------|------------|---------------|
| [Required] | StaticName | "Static data" |
|------------|------------|---------------|

Method: TestTableData

Description:

Returns true if the table data is anything other than zero, and false if the table data is zero or doesn't exist.

Here is an example from the TCP decoder (the table has been edited for clarity):

```
TABLE tcp_options
0 "End of Option List"
2 "" "Maximum Segment Size" 1 seg_size
6 "" "Echo" 1 misc_opt
8 "" "Time Stamp" 1 timestamp
Default " Unknown TCP option"
ENDTABLE
```

```
GROUP FIELD tcp_opt_type (Fixed 1) (Table tcp_options) "TCP Option"
```

```
FIELD opt_length (Fixed 1) IF (TestTableData tcp_options tcp_opt_type) (Decimal) "Length"
BRANCH (FromTable tcp_options tcp_opt_type)
```

This sequence deals with TCP options. The field "tcp_opt_type" identifies an option. Most options, but not all, are followed by parameters starting with a one-byte length field. The extra table data in the option table is either 1, meaning that a length field follows, or doesn't exist, meaning that nothing follows. The value of the field "tcp_opt_type" is looked up in the "tcp_options" table. Then the extra table data is checked for that entry. If table data exists, the Field "opt_length" is processed. If no table data exists, the field is not processed.

Parameters:

```
[Required] Table "Which table"
[Required] Field "Which field"
```

Group: Branch**Method:** DTEorDCE**Description:**

Chooses one of two branch targets depending on whether the current frame came from the DTE or the DCE. Clearly this has meaning only in contexts where there are two data streams distinguished as DTE and DCE. The first parameter names the statement to be branched to for a DTE frame and the second names the target for a DCE frame.

In the X.25 protocol, everything is defined in terms of DCE and DTE and so near the top of the X.25 decoder is the statement:

```
BRANCH (DTEorDCE dte_pkt dce_pkt)
```

This says: If the frame came from the DTE side, branch to the "dte_pkt" field, and if the frame came from the DCE side, branch to the "dce_pkt" field.

Parameters:

| | | |
|------------|-------|--------------------------------------|
| [Required] | Field | "Field present if this frame is DTE" |
| [Required] | Field | "Field present if this frame is DCE" |

Method: FromTable**Description:**

Causes a branch to the statement named in the branch item of a table entry. This is very commonly used and examples can be seen in many decoders. As in all similar cases, the field in question must have been previously decoded.

If the matching table entry has no branch item then no branch is made and control continues at the next statement in sequence. Omitting branch targets from tables is considered perfectly good practice when it is appropriate. If there is no matching entry in the table then the Default entry, if any, is used.

Parameters:

| | | |
|------------|-------|---------------|
| [Required] | Table | "Which table" |
| [Required] | Field | "Which field" |

Group: FORMAT**Method:** Binary**Description:**

Formats the current field value in binary notation. The method takes 3 optional parameters. If you want to set only the 2nd or 3rd parameter, you must include values for previous parameters as placeholders.

The "Separator" parameter allows you to specify a character to separate multi-byte fields. The default value of "0" will cause multi-byte fields to run together with no separator. To put a space between each byte, use (Binary " ").

The "Amount of Data to Display" parameter determines the maximum amount of data to display in the Decode pane. A value of 0 will display all data in the field, regardless of length. This parameter is used for limiting the amount of data displayed in the Decode pane for very large fields or fields where the size is unknown ahead of time.

The "Suppress leading zeroes" parameter tells the method whether or not to suppress leading zeroes. The default is "KeepPad", which means all digits in the number will be displayed, while "SuppressPad" causes leading zeroes to be converted to blanks.

FIELD flag_byte (Fixed 8 Bits) (Binary) "Flags" output: "Flags: 00000101"

FIELD flag_byte (Fixed 8 Bits) (Binary SuppressPad) "Flags" output: "Flags: 101"

Parameters:

| | | |
|------------|------|--|
| [Optional] | Str | "Separator" (Default value: 0) |
| [Optional] | Int | "Amount of data to display (0=unlimited)" (Default value: 0) |
| [Optional] | List | "Suppress leading zeroes?" (Default value: KeepPad) List values: KeepPad SuppressPad |

Method: Constant**Description:**

Constant is an exception among format methods in that it does not actually format anything or even make use of the field value. Rather it passes on for display a string passed as a parameter.

One time Constant is handy is when you want to add a units (or other) qualifier to a number as in:

```
FIELD port_speed (Fixed 2 Bytes) (Decimal) ALSO (Constant "bps")
```

One more valuable use for Constant is when we want to note if a particular field is present or not without needing to display its value. This typically fits when a PRINT_IF clause is used as in:

```
FIELD archive (Fixed 1 Bit) PRINT_IF (Fields EqualTo 1) (Constant "Archive") "Attribute"  
FIELD hidden (Fixed 1 Bit) PRINT_IF (Fields EqualTo 1) (Constant "Hidden") "Attribute"
```

Parameters:

| | | |
|------------|-----|-------------------|
| [Required] | Str | "String to print" |
|------------|-----|-------------------|

Method: Decimal**Description:**

Formats the field value as a signed decimal number. The field must be 64 bits or less in size. An error will be displayed if Decimal is used on fields greater than 64 bits in size.

Field values are inherently unsigned, so if there is a possibility that the field could have a negative value, you should use "RETRIEVE (SignExtension)" in addition to the Decimal format method. You can use this method to format a field value as an unsigned decimal number by not doing the RETRIEVE. For example:

```
FIELD signed_decimal (Fixed 2 Bytes) RETRIEVE (SignExtension 16) (Decimal) "Signed Decimal"  
FIELD unsigned_decimal (Fixed 2 Bytes) (Decimal) "Unsigned Decimal"
```

For the value 0xCAFE, signed_decimal displays -13570 while unsigned_decimal displays 51966.

Parameters:

None

Method: DosDate

Description:

Formats a 16-bit field value interpreted as a date expressed in DOS directory-entry format. An error will be displayed if DosDate is used with fields that are not 16 bits in size.

A DosDate is composed as follows:

| Bits | Contents |
|--------|-----------------------|
| 0 - 4 | Day of month (1-31) |
| 5 - 8 | Month (1-12) |
| 9 - 15 | Year relative to 1980 |

Parameters:

None

Method: DosTime

Description:

Formats a 16-bit field value as a time expressed in DOS directory-entry format. An error will be displayed if DosTime is used with fields that are not 16 bits in size.

A DosTime is made up as:

| Bits | Contents |
|---------|--|
| 0 - 4 | Number of two-second increments (0-29) |
| 5 - 10 | Minutes (0-59) |
| 11 - 15 | Hours (0-23) |

Parameters:

None

Method: Double

Description:

Formats the current field value as an IEEE-standard double-precision (64 bits or 8 bytes) floating-point number. Such a number is comprised of the following bits: 1 for sign, 11 for the exponent, and 52 for the mantissa. The optional parameter specifies the number of decimal places to be formatted with a default of two. An attempt to use Double with a value that has been sized at other than 64 bits will result in an error.

Parameters:

[Optional] Int "Number of places after the decimal point" (Default value: 2)

Method: Float**Description:**

Formats the current field value as an IEEE-standard single-precision (32 bits or 4 bytes) floating-point number. Such a number is comprised of the following bits: 1 for sign, 8 for the exponent, and 23 for the mantissa. The optional parameter specifies the number of decimal places to be formatted with a default of two. An attempt to use Float with a value that has been sized at other than 32 bits will result in an error.

Parameters:

[Optional] Int "Number of places after the decimal point" (Default value: 2)

Method: Hex**Description:**

Formats the current field value as a hexadecimal number. The method takes 3 optional parameters. If you want to set only the 2nd or 3rd parameter, you must include values for previous parameters as placeholders. See the description of the Binary method for information on the parameters.

The result of a Hex format does not include a "0x" preface or anything else to indicate hex notation; if it is not evident from the context that the display is hexadecimal then you may want to denote it as in this example:

FIELD id_code (Fixed 1 Byte) (Hex) ALSO (Constant "hex") "Id Code"

Parameters:

[Optional] Str "Separator" (Default value: 0)
[Optional] Int "Amount of data to display (0=unlimited)" (Default value: 0)
[Optional] List "Suppress leading zeroes?" (Default value: KeepPad)
 List values: KeepPad
 SuppressPad

Method: IpAddress**Description:**

Formats the current field value as an IP address. The field must be exactly 4 bytes in size. An error is displayed if IpAddress is used with fields that are not 4 bytes in size.

The display format is 123.123.123.123, where each number is a decimal number.

Parameters:

None

Method: MacAddress**Description:**

Formats a 6 byte field as a MAC Address. An error will be displayed if MacAddress is used with fields that are not 6 bytes in size.

The format is 00:01:02:03:04:05, where each digit is a hex digit.

Note that this method honors the host/network data order setting.

The "OUI table" parameter specifies a table with OUI identifiers in it. If an identifier is found, it will be used in place of the first 3 digits. This means that if you use the MacAddress method, you must have an OUI table specified in the decoder.

Parameters:

[Required] Table "OUI table"

Method: Octal**Description:**

Formats the current field value as an octal number. The method takes 3 optional parameters. If you want to set only the 2nd or 3rd parameter, you must include values for previous parameters as placeholders. See the description of the Binary method for information on the parameters.

No indication is included in the formatted number that it is expressed in octal. When it is not clear from the context that octal is being used you may want to note it in the same sort of way as in the example given for the Hex method.

Parameters:

| | | |
|------------|------|--|
| [Optional] | Str | "Separator" (Default value: 0) |
| [Optional] | Int | "Amount of data to display (0=unlimited)" (Default value: 0) |
| [Optional] | List | "Suppress leading zeroes?" (Default value: KeepPad) |
| | | List values: KeepPad |
| | | SuppressPad |

Method: Scale**Description:**

Multiplies the field by the scale factor, and formats as a decimal using the "digits after the decimal point" parameter. This method is useful because it multiplies the field value by a float, rather than an integer.

Parameters:

| | | | |
|------------|-------|----------------------------------|--------------------|
| [Required] | Float | "Factor" | |
| [Optional] | Int | "Digits after the decimal point" | (Default value: 2) |

Method: SecondsFrom1970

Description:

Interprets the field value as the number of seconds since 1970. The field must be 32 bits in size or less. An error will be displayed if this method is used with fields greater than 32 bits in size.

Parameters:

None

Method: StringOfAscii

Description:

Formats the current field as a string of ASCII characters. Non-printable characters are displayed as dots.

The optional parameter indicates how many characters to display in the Decode pane of the Frame Display window. Setting a value here does not affect the actual size of the field, merely prevents very long character strings from being displayed. If there is more data in the field when the parameter value is reached, an ellipsis (...) is displayed in the Decode pane. The default value of zero means that the entire contents of the field will be shown, regardless of length.

Parameters:

| | | | |
|------------|-----|---|--------------------|
| [Optional] | Int | "Amount of data to display (0=unlimited)" | (Default value: 0) |
|------------|-----|---|--------------------|

Method: StringOfBCD

Description:

Formats the current field as a string of binary-coded decimal.

See the description for the StringOfASCII Method for information on the "Amount of data to display" parameter.

Parameters:

[Optional] Int "Amount of data to display (0=unlimited)" (Default value: 0)

Method: StringOfBinary

Description:

Formats the current field as a string of binary digits.

See the description for the StringOfASCII Method for information on the "Amount of data to display" parameter.

Parameters:

[Optional] Int "Amount of data to display (0=unlimited)" (Default value: 0)

Method: StringOfDecimal

Description:

Formats the current field as a string of decimal numbers. Each hex byte is formatted in decimal, rather than interpreting all the bytes in the field as a single number.

See the description for the StringOfASCII Method for information on the "Amount of data to display" parameter.

Parameters:

[Optional] Int "Amount of data to display (0=unlimited)" (Default value: 0)

Method: StringOfHex

Description:

Formats the current field as a string of hexadecimal numbers, with a space between each byte.

See the description for the StringOfASCII Method for information on the "Amount of data to display" parameter.

Parameters:

[Optional] Int "Amount of data to display (0=unlimited)" (Default value: 0)

Method: StringOfUTF8

Description:

Formats the current field as a string of UTF8 characters. Non-printable characters are displayed as dots.

See the description for the StringOfASCII Method for information on the "Amount of data to display" parameter.

Parameters:

[Optional] Int "Amount of data to display (0=unlimited)" (Default value: 0)

Method: StringOfUnicode

Description:

Formats the current field as a string of Unicode characters. Because FTS can be run on systems that do not support Unicode, this method ignores the upper byte of each byte pair and interprets the lower byte as if it were an ASCII character. This works for the Latin alphabet but will not work for other alphabets. Non-printable characters are displayed with a dot.

See the description for the StringOfASCII Method for information on the "Amount of data to display" parameter.

Parameters:

[Optional] Int "Amount of data to display (0=unlimited)" (Default value: 0)

Method: Table

Description:

Extracts a string from a table for display. The method extracts either of two strings according to the context in which it is to be displayed. For display in the Summary pane, the first string in the table entry is retrieved. For display in the Decode pane, the second string is used. If only one string is present, it is used in both the Summary and Decode panes. If the field value fails to match an entry in the table, the Default entry is used.

When using the Table method, the field value must be 64 bits or less in size.

Parameters:

[Required] Table "Which table"

Method: UUID

Description:

Formats the field as a 16 byte Universally Unique Identifier (UUID). Also called Globally Unique Identifier (GUID). An error will be displayed if UUID is used with fields that are not exactly 16 bytes in size.

The display format is (where 0x represents a hex digit):
0x0x0x0x-0x0x-0x0x-0x0x-0x0x0x0x0x0x

Parameters:

None

Group: NEXT_PROTOCOL

Method: FromField

Description:

Returns the value in the field named in the parameter. FTS makes no check that the field contains a valid value; if the field has not been processed then the result will be an indeterminate value. This lack of a check is not due to laziness; it is because making such a check would be time consuming to a degree unacceptable when decoding data arriving from a fast line.

Parameters:

[Required] Field "Field that contains the next protocol"

Method: IsAlways

Description:

IsAlways is used when the next protocol is invariable. It takes one parameter which is the value to be returned.

Parameters:

[Required] Int64 "Value to return"

Group: NEXT_PROTOCOL_SIZE

Method: FromField

Description:

Returns the value from the field named in the parameter. The value is used to determine the size of the next protocol layer.

The "Offset" parameter adds or subtracts from the value in the "Field" parameter. Positive values are added, negative values are subtracted.

Parameters:

| | | |
|------------|-------|--|
| [Required] | Field | "Field that contains the next protocol size" |
| [Required] | Int | "Offset" |

Method: FromFieldIfLessThan

Description:

If the value of the field specified is less than the "Threshold" parameter, it is returned as the size of the next protocol layer. Otherwise, the size of the next protocol layer is the rest of the payload, minus the value of the "Offset" parameter.

Parameters:

| | | |
|------------|-------|--|
| [Required] | Field | "Field that contains the next protocol size" |
| [Required] | Int | "Threshold for length" |
| [Required] | Int | "Offset" |

Method: OffsetToEnd

Description:

This sets the size of the next protocol layer as the rest of the frame minus the value of the "Offset" parameter.

The Async PPP decoder, for example, includes the statement:

```
NextProtocolSize (OffsetToEnd 2)
```

The size of the next layer is thus the number of bytes in the rest of the frame minus two to allow for the 16-bit frame-check sequence (FCS) at the end.

Parameters:

| | | |
|------------|-----|--------------------------------|
| [Required] | Int | "Offset from the end of frame" |
|------------|-----|--------------------------------|

Method: ThisLayerLength

Description:

This sets the size of the next protocol layer to the value in the "Field" parameter minus the value of the "Offset" parameter. ThisLayerLength is used when a field in the current layer indicates the combined size of the current layer plus the payload, where the payload contains the next layer(s).

Parameters:

| | | |
|------------|-------|---|
| [Required] | Field | "Field that contains the length in bytes of this layer" |
| [Optional] | Int | "Offset" (Default value: 0) |

Group: PREPROCESSING

Method: InitField**Description:**

This initializes the passed field before starting the decoder. This is useful if you want to do a test on a field, but it is not always encountered in the frame. This forces a known value into it.

Parameters:

| | | | |
|------------|-------|----------------------|--------------------|
| [Required] | Field | "Field to set" | |
| [Optional] | Int64 | "Value to set it to" | (Default value: 0) |

Group: RETRIEVING_DATA

Method: AddField**Description:**

Adds the value of another field to the value of the current field, storing the result in the current field.

Parameters:

| | | |
|------------|-------|----------------|
| [Required] | Field | "Field to add" |
|------------|-------|----------------|

Method: AddFieldToField**Description:**

Adds the values of two other fields, storing the result in the current field.

Parameters:

| | | |
|------------|-------|----------------|
| [Required] | Field | "First field" |
| [Required] | Field | "Second field" |

Method: AddFieldToInteger**Description:**

Adds the value of another field to a constant, storing the result in the current field.

Parameters:

| | | |
|------------|-------|-------------------------|
| [Required] | Field | "Field" |
| [Required] | Int64 | "Constant value to add" |

Method: AddInteger

Description:

Adds an integer constant to the value of the current field.

Parameters:

| | | |
|------------|-------|-------------------------|
| [Required] | Int64 | "Constant value to add" |
|------------|-------|-------------------------|

Method: AddIntegerToField

Description:

Adds a constant to the value of another field, storing the result in the current field.

Parameters:

| | | |
|------------|-------|------------------|
| [Required] | Int64 | "Constant value" |
| [Required] | Field | "Field to add" |

Method: AndMask

Description:

Performs a bit-wise AND operation with the field value and a given mask. The single parameter is the mask, which can be up to 64 bits (8 bytes) long.

A bit-wise AND compares each bit in the field value to the corresponding bit in the mask. If both are 1, the result is 1. Otherwise the result is 0. For example, if a byte field contained 0x77 and a RETRIEVE was performed using (AndMask 0xF0), the result would be 0x70.

Parameters:

| | | |
|------------|-------|-------------------------|
| [Required] | Int64 | "Mask to apply to data" |
|------------|-------|-------------------------|

Method: DivideField

Description:

Divides the current field value by the value of another field, storing the result in the current field. An error will be displayed and a result of -1 returned if the divisor is zero.

Parameters:

[Required] Field "Field to divide by"

Method: DivideFieldByField

Description:

Divides the value of the first field by the second field, storing the result in the current field. An error will be displayed and a result of -1 returned if the divisor is zero.

Parameters:

[Required] Field "First field"
[Required] Field "Second field"

Method: DivideFieldByInteger

Description:

Divides the value of another field by a constant, storing the result in the current field. An error will be displayed and a result of -1 returned if the divisor is zero.

Parameters:

[Required] Field "Field"
[Required] Int64 "Constant value to divide by"

Method: DivideInteger

Description:

Divides the current field value by an integer constant, storing the result in the current field. An error will be displayed and a result of -1 returned if the divisor is zero.

Parameters:

[Required] Int64 "Constant value to divide by"

Method: DivideIntegerByField

Description:

Divides a constant by the value of another field, storing the result in the current field. An error will be displayed and a result of -1 returned if the divisor is zero.

Parameters:

[Required] Int64 "Constant value"
[Required] Field "Field to divide by"

Method: IndexedPersistentField

Description:

Retrieves a field stored previously by a StoreIndexedPersistentField method. This method requires the name of the static to retrieve, and a field giving the index to use. See the description of StoreIndexedPersistentField for a full explanation of how this works.

Parameters:

[Required] StaticName "Static data"
[Required] Field "Field that contains the index"

Method: IntraframeField

Description:

Retrieves a field stored previously by the StoreIntraframeField method. If the value is unavailable because the stored field hasn't been encountered in this frame, an error is returned saying so.

The method doesn't take a parameter, but you do need to specify the name of the value to retrieve. The syntax is:
RETRIEVE (IntraframeField field_name).

Parameters:

[Required] StaticName "Static data"

Method: MultiplyField

Description:

Multiplies the current field value by the value of another field.

Parameters:

[Required] Field "Field to multiply by"

Method: MultiplyFieldByField

Description:

Multiplies the values of two fields, storing the result in the current field.

Parameters:

[Required] Field "First field"
[Required] Field "Second field"

Method: MultiplyFieldByInteger

Description:

Multiplies the value of another field by a constant, storing the result in the current field.

Parameters:

| | | |
|------------|-------|---------------------------------|
| [Required] | Field | "Field" |
| [Required] | Int64 | "Constant value to multiply by" |

Method: MultiplyInteger

Description:

Multiplies the current field value by an integer constant.

Parameters:

| | | |
|------------|-------|---------------------------------|
| [Required] | Int64 | "Constant value to multiply by" |
|------------|-------|---------------------------------|

Method: MultiplyIntegerByField

Description:

Multiplies a constant by the value of another field, storing the result in the current field.

Parameters:

| | | |
|------------|-------|------------------------|
| [Required] | Int64 | "Constant value" |
| [Required] | Field | "Field to multiply by" |

Method: OrMask

Description:

Performs a bit-wise inclusive OR operation with the field value and a given mask. The single parameter is the mask, which can be up to 64 bits (8 bytes) long.

A bit-wise inclusive OR compares each bit in the field value with the corresponding bit in the mask. If either bit is 1, the result is 1. If both bits are 1, the result is 1. Otherwise the result is 0. For example, if a byte field contained 0x77 and a RETRIEVE was performed using (OrMask 0xF0), the result would be 0xF7.

Parameters:

| | | |
|------------|-------|-------------------------|
| [Required] | Int64 | "Mask to apply to data" |
|------------|-------|-------------------------|

Method: PersistentField

Description:

Retrieves a field stored previously by a StorePersistentField method. When using this method, the first item after the method name must be the name of the static data object in which the field was stored.

To continue the example used in the StorePersistentField method description, if you wanted to retrieve the value, you would use this syntax:

```
FIELD retrieve_fieldname (Fixed 0 Bits) RETRIEVE (PersistentField KeepThisData) (Decimal) "Value of Persistent Field"
```

Parameters:

[Required] StaticName "Static data"

Method: ReplaceWithExtraData**Description:**

Looks up the value of the current field in the table specified in the parameter, and then replaces the value of the current field with the extra data from the table. If the table entry does not have extra data, zero is returned. Usually used in connection with the STORE keyword.

For example, assume a protocol has a field where the value maps to another value indicating a size. The method allows you to put the actual size in the field, where it can be used for other purposes. In the example below, storing the retrieved value under a different field name allows you to display the correct string from the table, and still use the correct value for the "next_field" field.

```
TABLE sizes
1 "1 Bit" "1 Bit" 1
2 "4 Bits" "4 Bits" 4
3 "1 Byte" "1 Byte" 8
ENDTABLE
```

```
FIELD size_lookup (Fixed 1 Byte) (Table sizes) "Next Field Size" STORE field_size RETRIEVE
(ReplaceWithExtraData sizes)
FIELD next_field (FromField Bits field_size) (Decimal) "Number"
```

Parameters:

[Required] Table "Which table"

Method: ShiftLeft**Description:**

Shifts the data to the left by the number of bits specified in the parameter.

For example, if a field contained the value 0x7f and a RETRIEVE (ShiftLeft 4) was done, the resulting value would be 0xf0.

Parameters:

[Required] Int "Number of bits to shift"

Method: ShiftRight

Description:

Shifts the data to the right by the number of bits specified in the parameter.

For example, if a field contained the value 0x7f and a RETRIEVE (ShiftRight 4) was done, the resulting value would be 0x07.

Parameters:

[Required] Int "Number of bits to shift"

Method: SignExtension

Description:

Extends the sign bit to 64 bits. This allows you to treat a field of any size between 1 and 63 bits as a signed integer rather than an unsigned one. The "Number" parameter indicates the number of bits in the value; usually this is the same as the field size. It must be explicitly provided in case a RETRIEVE alters the field.

Parameters:

[Required] Int "Number of bits in value"

Method: SplitField

Description:

This removes some bits from the middle of the current field and moves the left part of the field over to the remaining bits on the right. The first parameter is the number of bits on the right side to keep, and the second parameter is the number of bits in the middle to delete.

For example, if the raw data in a 16-bit field is 0x5A8F (binary 01011010 10001111) and is retrieved using (SplitField 7 1), the low 7 bits are preserved, the high bit of the low byte is discarded and the remaining bits are right-shifted one place to produce a result of 0x2D0F (binary 00101101 00001111).

Parameters:

[Required] Int "Number of bits on right side to preserve"
[Required] Int "Number of bits to delete"

Method: StoreField

Description:

Stores the value of another field to the current field.

Parameters:

[Required] Field "Field to store"

Method: StoreIndexedPersistentField

Description:

This method stores the current field value to a static data array, indexed by the value in the field specified in the parameter. This method is similar to the StorePersistentField method in that the data is saved forever rather than just for the length of the layer or frame. The difference between this method and StorePersistentField is that you can store multiple values to the same static name.

For example, say you have a master device talking to multiple slaves. The master sends a Read command to Slave 1, a Write Configuration command to Slave 4, and a Read Configuration command to slave 7. The command frames contain both the command and the slave the command is directed to. The responses from the slaves, however, don't reference the command, though they do say which slave the response is from. You can't correctly decode the responses unless you can match them up to the commands.

You can do this if you save the command value indexed by the slave number. The command part of the decoder might look like this:

```
FIELD slave_number (Fixed 1) (Decimal) "Slave Number"  
FIELD command_code (Fixed 1) RETRIEVE (StoreIndexedPersistentField command slave_number)  
(Hex) "Command"
```

The response section might look like this:

```
FIELD slave_number (Fixed 1) (Decimal) "Slave Number"  
FIELD retrieved_command_code (Fixed 0) RETRIEVE (IndexedPersistentField command  
slave_number) SUPPRESS_DETAIL  
BRANCH (FromTable command_code retrieved_command_code)
```

Using the above example, after the first command frame is decoded, the array holds:

```
1     Read
```

After the second and third command frames are decoded, it holds:

```
1     Read  
4     Write Configuration  
7     Read Configuration
```

When a response comes back from Slave 7, the response section of the decoder looks up 7 in the array and retrieves the value "Read Configuration". It then branches to the Read Configuration section of the decoder.

Parameters:

[Required] StaticName "Static data"

[Required] Field "Field that contains the index"

Method: StoreInteger

Description:

Stores a constant value to the current field.

Parameters:

[Required] Int64 "Constant value to store"

Method: StoreIntraframeField

Description:

Saves the value of the current field for the duration of the current frame. This makes the field's value available to other layers within the frame (i.e. other decoders used in the frame). When the decoding of the frame is finished, the value is cleared prior to decoding the next frame. You can store as many fields as you wish, provided you use a different name for each one.

The method doesn't take a parameter, but you do need to specify a name to store the value under.

The syntax is:

RETRIEVE (StoreIntraframeField field_name).

Parameters:

[Required] StaticName "Static data"

Method: StorePersistentField

Description:

Saves the value of the current field in the static data object specified. The item after the method name must be the name of the static data object in which the field is to be stored. Data is retrieved using the PersistentField method. For example:

GROUP request IF (MatchTableData code_table req_or_resp 1)

 FIELD code (Fixed 1) RETRIEVE (StorePersistentField KeepThisCode) (Decimal) "Code"

GROUP response IF (MatchTableData code_table req_or_resp 2)

 FIELD response_code (Fixed 7 Bits) (TABLE response_code_table) IN_SUMMARY "Resp Code" 100 "Response Code"

 FIELD retrieved_code (Fixed 0 bits) RETRIEVE (PersistentField KeepThisCode) (Hex) "Retrieved Code"

 FIELD no_response (Fixed 1 Byte) IF (FieldIsBetween 0x00 0x1F retrieved_code) (Constant "No Response Required") "Response"

FIELD resp_req (Fixed 2 Bytes) IF NOT (FieldsBetween 0x00 0x1F retrieved_code) (Table resp_code_table) "Response"

Parameters:

[Required] StaticName "Static data"

Method: SubtractField

Description:

Subtracts the value of another field from the value of the current field, storing the result in the current field.

Parameters:

[Required] Field "Field to subtract"

Method: SubtractFieldMinusField

Description:

Subtracts the value of the second field from the value of the first field, storing the result in the current field.

Parameters:

[Required] Field "First field"
[Required] Field "Second field"

Method: SubtractFieldMinusInteger

Description:

Subtracts a constant from the value of another field, storing the result in the current field.

Parameters:

[Required] Field "Field value"
[Required] Int64 "Constant value to subtract"

Method: SubtractInteger

Description:

Subtracts an integer constant from the value of the current field, storing the result in the current field.

Parameters:

[Required] Int64 "Constant value to subtract"

Method: SubtractIntegerMinusField

Description:

Subtracts the value of another field from a constant, storing the result in the current field.

Parameters:

| | | |
|------------|-------|---------------------|
| [Required] | Int64 | "Constant value" |
| [Required] | Field | "Field to subtract" |

Group: SIZE

Method: Fixed

Description:

Returns the number of bits specified. Two parameters are used, a count and a unit specifier. The latter may be any of:

"Bit" or "Bits" = 1-bit units

"Nibble" or "Nibbles" = 4-bit units

"Byte" or "Bytes" = 8-bit units

"Octet" or "Octets" = 8-bit units

"Word" or "Words" = 16-bit units

"Dword" or "Dwords" = 32-bit units (double words)

"Qword" or "Qwords" = 64-bit units (quad words)

"Times" - used in REPEAT COUNT (Fixed 3 Times) constructions. Equivalent of saying (Fixed 3 Bits) but is more clear in the context of a REPEAT.

The singular and plural forms are distinguished only for readability; for instance, the method will return 8 whether you write "(Fixed 1 Byte)" or "(Fixed 1 Bytes)". The units parameter is optional and, if omitted, will default to "Bytes".

Parameters:

| | | |
|------------|------|--------------------------------|
| [Required] | Int | "Number (in units)" |
| [Optional] | List | "Units" (Default value: Bytes) |

List values:

- Bit
- Bits
- Times
- Nibble
- Nibbles
- Byte
- Bytes
- Octet
- Octets
- Word
- Words

Dword
Dwords
Qword
Qwords

Method: FixedUpTo

Description:

Returns the fixed value given in the first parameter "Number", unless that would put the pointer past the absolute end point given by the "Length field" and "Offset" parameters. The absolute end point is found by starting immediately after the field in "Length field", adding the value of "Length field" and subtracting the value of "Offset". If the value in "Number" puts the pointer past the absolute end point, the size is truncated to the absolute end point.

For example:

FIELD length (Fixed 1 Byte) (Decimal) "Length"

FIELD address (FixedUpTo 6 length 5) (Hex) "Address"

Assume the value in "length" is 10. Starting after the length field, the absolute end point is $10 - 5 = 5$ bytes from the length field. The address field, which is right after the length field, is 6 bytes long, except that this is 1 byte past the absolute end point. Instead of being set at 6 bytes, the size of "address" is set to 5. If the value of "length" was 14, the absolute end point would be $14 - 5 = 9$ bytes from the end of "length". In this case, the size of the address field would be set to 6, because this does not go past the absolute end point.

This method handles cases where bad data may cause a group to overflow, which is particularly a problem with repeated groups. The method ensures that the group is bounded to a maximum size so that the next section will be right, even if the current one is bad.

Parameters:

| | | |
|------------|-------|--------------------------------|
| [Required] | Int | "Number (in units)" |
| [Required] | Field | "Length field" |
| [Optional] | Int | "Offset" (Default value: 0) |
| [Optional] | List | "Units" (Default value: Bytes) |

List values:

- Bit
- Bits
- Times
- Nibble
- Nibbles
- Byte
- Bytes
- Octet
- Octets
- Word
- Words
- Dword
- Dwords

Method: FromField

Description:

Derives a size from the value in a field that has already been processed. Three parameters are used, the first two being required. The first parameter is the unit, that is one of the terms listed for the Fixed method ("Bits", "Bytes", etc.). The second is the name of the field that contains the size. The optional third parameter is an adjustment, a (possibly negative) integer that is added to the field value.

Suppose the field "data_count" contains the value 6. (FromField Words data_count 1) would extract the value 6, add 1, multiply by 16 (remember all Size methods return the length in bits), and return the result 112.

Parameters:

| | | | |
|------------|-------|--|--|
| [Optional] | List | "Units" (Default value: Bytes) List values: | Bit Bits Times Nibble Nibbles Byte Bytes Octet Octets Word Words Dword Dwords Qword Qwords |
| [Required] | Field | "Which field" | |
| [Optional] | Int | "Offset (in units)" | (Default value: 0) |

Method: OddTerminated

Description:

Counts bytes up to and including the first odd byte found.

Parameters:

None

Method: OneToken

Description:

Finds the first space, tab, CR, LF, or null character. The returned size is never larger than the maximum passed in. If no maximum is given, the maximum returned size is assumed to be the size of the rest of the frame.

Parameters:

[Optional] Int "Maximum size (in bytes)" (Default value: -1)

Method: String

Description:

String is provided for sizing null-terminated ASCII strings. The method searches forward for a null character and returns the size of the string including the null. If the search reaches the end of the layer without encountering a null then no error is given and the field is sized to the layer's end.

Parameters:

None

Method: ToEndOfLayer

Description:

Returns the number of bits from the pointer to the end of the layer, excluding the number of bytes in the "Bytes" parameter. For example, (ToEndOfLayer -2) would return the number of bits left in the frame minus 16. This method is useful when a field subsumes all data left in the layer except for a CRC or checksum at the end. The parameter may be omitted in which case no adjustment is made.

Parameters:

[Optional] Int "Bytes from end" (Default value: 0)

Method: ToTerminatingValue

Description:

Scans ahead for the value specified in the "Value" parameter, and returns the number of bits between the value and the current bit. If the terminating value is not found, the size is set to the number of bytes remaining in the layer. The "Inclusive/Exclusive" parameter specifies whether to include the terminating value in the count or not. Thus (ToTerminatingValue Inclusive 0xFF) would count bytes until one was found containing 0xFF and return a count that included the 0xFF. If the pointer is not at a byte boundary when the scan starts, then the pointer is moved to the beginning of the next byte and the scan starts from there.

Parameters:

| | | |
|------------|-------|---|
| [Required] | List | "Inclusive/Exclusive" List values: Inclusive Exclusive |
| [Required] | Int64 | "Value that signifies the start of the next field or end of this one" |

Method: UnicodeString**Description:**

Determines the length of a string of Unicode characters by scanning alternate bytes until it finds one that is NULL. The null byte is included in the count. Note that the data order specified for the decoder is not used here; the comparison is performed in host data order. For example, if the pattern at the pointer is (in hex): 00 48 00 45 00 4C 00 4C 00 50 00 00 ... then (UnicodeString) would find 6 characters, 12 bytes and return 96.

Parameters:

None

Group: START_BIT**Method:** FromEnd**Description:**

Moves the pointer to a given offset from the end of the layer. The first parameter is the number of units to move and the second is the unit (see the Fixed Size Method). Thus (FromEnd 4 Bits) would move the pointer 4 bits prior to the end of the layer while (FromEnd 4 Bytes) would move the pointer 32 bits from the end of the layer. Both parameters are optional and, if omitted, will default to zero and Bytes.

Parameters:

| | | |
|------------|------|--|
| [Optional] | Int | "Offset" (Default value: 0) |
| [Optional] | List | "Units" (Default value: Bytes) List values: Bit Bits Times Nibble Nibbles Byte Bytes Octet Octets Word Words Dword Dwords |

Qword
Qwords

Method: FromStart

Description:

Moves the pointer to a given offset from the start of the layer. Parameters are the same as for the FromEnd method.

Parameters:

| | | | |
|------------|------|--------------------------------|--|
| [Optional] | Int | "Offset" | (Default value: 0) |
| [Optional] | List | "Units" (Default value: Bytes) | |
| | | List values: | Bit Bits Times Nibble Nibbles Byte Bytes Octet Octets Word Words Dword Dwords Qword Qwords |

Method: Move

Description:

Moves the pointer the requested number of units from the current position. Use negative numbers to move backwards.

The first parameter is the number of units to move and the second is the unit (as listed for the Fixed Size Method). Thus, (Move -1 byte) would move the pointer back one byte.

Parameters:

| | | | |
|------------|------|--------------------------------|--|
| [Required] | Int | "Amount to move" | |
| [Optional] | List | "Units" (Default value: Bytes) | |
| | | List values: | Bit Bits Times Nibble Nibbles Byte Bytes |

Octet
Octets
Word
Words
Dword
Dwords
Qword
Qwords

Method: MoveAbsoluteFromField

Description:

This method takes the value in the "Field that contains offset" parameter and starting from the beginning of the layer, moves the pointer that many bytes plus the number of bytes in the "Additional Offset" parameter. A positive value in "Offset" moves the pointer forward, while a negative value moves back. An error will be triggered if the result points outside the frame. This method is useful for ensuring that decoding begins again at the proper location in case bad data prior in the frame causes trouble in a REPEAT loop or similar construction.

Parameters:

| | | | |
|------------|-------|------------------------------|--------------------|
| [Required] | Field | "Field that contains offset" | |
| [Optional] | Int | "Additional Offset In Bytes" | (Default value: 0) |

Method: MoveRelativeFromField

Description:

This method takes the value in the "Field that contains offset" parameter and starting from that field, moves the pointer that many bytes plus the number of bytes in the "Additional Offset" parameter. A positive value in "Offset" moves the pointer forward, while a negative value moves back. An error will be triggered if the result points outside the frame. This method is useful for ensuring that decoding begins again at the proper location in case bad data prior in the frame causes trouble in a REPEAT loop or similar construction.

Parameters:

| | | | |
|------------|-------|------------------------------|--------------------|
| [Required] | Field | "Field that contains offset" | |
| [Optional] | Int | "Additional Offset In Bytes" | (Default value: 0) |

Method: MoveToField

Description:

Moves the pointer to the beginning of the field specified. This must be a field that has already been decoded. MoveToField is handy when you need to process a field twice.

Parameters:

[Required] Field "Field to move to"

Method: NextByteBoundary**Description:**

Moves the pointer to the beginning of the next byte. If the pointer is already at a byte boundary, it doesn't move.

Parameters:

None

Method: NextWordBoundary**Description:**

Moves the pointer to the next word boundary. If the pointer is already at a word boundary, it doesn't move.

Parameters:

None

Method: PreviousByteBoundary**Description:**

Moves the pointer to the beginning of the current byte. If the pointer is already at a byte boundary, it doesn't move.

Parameters:

None

Group: VERIFY**Method:** Check_CrcCCITT**Description:****Parameters:**

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcCCITTrev

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcCRC16

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcCRC16rev

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcCRC32

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcDF1

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcDF1_FD_BCC

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcDF1_HD_BCC

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcHDLC**Description:****Parameters:**

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcHDLCerr**Description:****Parameters:**

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcLRC**Description:**

Longitudinal Redundancy Check. This method XORs each byte in the layer, starting with the byte specified in the first parameter. The result is an 8-bit value.

Parameters:

| | | |
|------------|-----|---|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |

[Optional] List "Swap Result" (Default value: NoSwap)
List values: NoSwap
Swap

Method: Check_CrcModBus

Description:

Parameters:

[Required] Int "First byte to include (0=first byte in layer)"
[Required] Int "Seed"
[Optional] List "Swap Result" (Default value: NoSwap)
List values: NoSwap
Swap

Method: Check_CrcModBus_LRC

Description:

Parameters:

[Required] Int "First byte to include (0=first byte in layer)"
[Required] Int "Seed"
[Optional] List "Swap Result" (Default value: NoSwap)
List values: NoSwap
Swap

Method: Check_CrcSum

Description:

Summed checksum. Adds each byte to the next byte. Returns a 2-byte result.

Parameters:

[Required] Int "First byte to include (0=first byte in layer)"
[Required] Int "Seed"
[Optional] List "Swap Result" (Default value: NoSwap)
List values: NoSwap
Swap

Method: Check_CrcSum1comp

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcSum2comp

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcXOR1comp

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Check_CrcXOR2comp

Description:

Parameters:

| | | |
|------------|------|--|
| [Required] | Int | "First byte to include (0=first byte in layer)" |
| [Required] | Int | "Seed" |
| [Optional] | List | "Swap Result" (Default value: NoSwap) List values: NoSwap Swap |

Method: Fields

Description:

Returns OK if the statement comparing the field value with a given constant using the operator specified is true. If the comparison fails, the field value will be displayed in red in the Decode pane and the expected value(s) will be displayed next to it. The optional "Output Radix" parameter specifies whether to show the expected value(s) in decimal or hex.

Here is a typical example taken from the IP decoder. This serves to check that the version of IP used is at least 4:

FIELD version (Fixed 4 Bits) (TABLE ver_nums) "Version" VERIFY (Fields GreaterThanOrEqualTo 4)

Parameters:

| | | |
|------------|-------|--|
| [Required] | List | "Operator" List values: GreaterThan GreaterThanOrEqualTo LessThan LessThanOrEqualTo EqualTo NotEqualTo |
| [Required] | Int64 | "Match value" |
| [Optional] | List | "Output radix" (Default value: AsDecimal) List values: AsDecimal AsHex |

Method: FieldsBetween

Description:

Returns OK if the value of the current field is between the minimum and maximum values. This method is inclusive. In other words, if the value of the current field is exactly equal to the value of either the maximum or minimum, the method returns OK.

The comparison can be reversed by using the NOT keyword after the VERIFY. For example, VERIFY NOT (FieldsBetween 5 10) will return OK if the field value is less than 5 or greater than 10, and fail if the value is 5, 6, 7, 8, 9 or 10.

Parameters:

| | | |
|------------|-------|-----------------|
| [Required] | Int64 | "Minimum value" |
| [Required] | Int64 | "Maximum value" |

Method: FieldLengthIs

Description:

This makes sure that the field length is correct. This is useful for those fields that have variable lengths, but not all of the lengths are legal. The parameters and functionality are the same as for FieldIs, except that the method is verifying the length of the field rather than the contents.

Parameters:

| | | | |
|------------|-------|---|---|
| [Required] | List | "Operator" | List values: GreaterThan GreaterThanOrEqualTo LessThan LessThanOrEqualTo EqualTo NotEqualTo |
| [Required] | Int64 | "Match value" | |
| [Optional] | List | "Units" (Default value: Bytes) | List values: Bit Bits Times Nibble Nibbles Byte Bytes Octet Octets Word Words Dword Dwords Qword Qwords |
| [Optional] | List | "Output radix" (Default value: AsDecimal) | List values: AsDecimal AsHex |

Method: IsInTable

Description:

Returns OK if the value of the current field exists in the specified table and is not the default.

This method is used in the IP decoder to check that the next layer protocol is defined:

FIELD protocol (Fixed 1) (Table protocol) IN_SUMMARY "Protocol" 100 "Protocol" VERIFY (IsInTable protocol)

Parameters:

| | | |
|------------|-------|---------|
| [Required] | Table | "Table" |
|------------|-------|---------|

Method: IsLengthOfLayer

Description:

Returns OK if the value in the current field is equal to the length of the layer minus the value of the "Offset" parameter. This method is handy for verifying the internal size of a layer against the amount of data collected; the adjustment can be used to exclude a frame check or anything else of the sort.

Parameters:

| | | |
|------------|------|-------------------------------------|
| [Optional] | List | "Operator" (Default value: EqualTo) |
| | | List values: GreaterThan |
| | | GreaterThanOrEqualTo |
| | | LessThan |
| | | LessThanOrEqualTo |
| | | EqualTo |
| | | NotEqualTo |
| [Optional] | Int | "Offset" (Default value: 0) |
| [Optional] | List | "Units" (Default value: Bytes) |
| | | List values: Bit |
| | | Bits |
| | | Times |
| | | Nibble |
| | | Nibbles |
| | | Byte |
| | | Bytes |
| | | Octet |
| | | Octets |
| | | Word |
| | | Words |
| | | Dword |
| | | Dwords |
| | | Qword |
| | | Qwords |