Data Validation Language (DVL) Guide

© 2017 Glasswall Solutions Ltd ALL RIGHTS RESERVED

Information contained herein is the property of Glasswall Limited and is proprietary and confidential.

Martin O’Brien

Glasswall Solutions Ltd.

(e): support@glasswallsolutions.com

Creation Date – 25/09/13

Version – 1

**Document History**

Table 1: Document Change History

|  |  |  |  |
| --- | --- | --- | --- |
| ***Issue Date*** | ***Issue Number*** | ***Author*** | ***Description*** |
| 25/09/13 | 1.0 | Martin O’Brien | Version 1 |
| 08/02/17 | 1.1 | Elvin Barreto-White | Updated to use latest Glasswall template. |

**Document Distribution**

Table 2: Document Distribution

|  |  |  |  |
| --- | --- | --- | --- |
| ***Name*** | ***Position*** | ***Date*** | ***Signature*** |
| Sam Hutton | CTO |  |  |
|  |  |  |  |

**Copyright and Contact Details**

The copyright in this work is vested in Glasswall Solutions Ltd, and the document is issued in confidence for the purpose for which it is supplied. It must not be reproduced in whole or in part or used for tendering or manufacturing purposes except under agreement or with the consent in writing of Glasswall Solutions Limited and then only on condition that this notice is included in any such reproduction. No information as to the contents or subject matter of this document or any part thereof arising directly or indirectly there from shall be given orally or in writing or communicated in any manner whatsoever to any third part being an individual firm or company or any employee thereof without the prior consent in writing of Glasswall Solutions Limited.

© Glasswall Solutions Limited 2016

If there are any questions related to this report, these should be addressed to:

Sam Hutton

Glasswall Solutions Limited

E-mail: [shutton@glasswallsolutions.com](mailto:shutton@glasswallsolutions.com)

**Index**

[1. Overview 5](#_Toc474337760)

[1.1 Syntax 5](#_Toc474337761)

[1.2 Values 5](#_Toc474337762)

[1.3 Validation 6](#_Toc474337763)

[2. Work Item Template Wording 7](#_Toc474337764)

[3. Substructures 8](#_Toc474337765)

[4. Syntax 9](#_Toc474337766)

[4.1 General Notes 9](#_Toc474337767)

[4.2 Comments 9](#_Toc474337768)

[5. Functions 10](#_Toc474337769)

[5.1 Office Word Functions 10](#_Toc474337770)

[5.1.1 VERSION(word-version) 10](#_Toc474337771)

[5.1.2 GETCPCHAR(CP) 10](#_Toc474337772)

[5.2 General Functions 10](#_Toc474337773)

[5.2.1 RANGE (field, min, max) 10](#_Toc474337774)

[5.2.2 ISIN (field, val1, val2,…) 11](#_Toc474337775)

[5.2.3 ISIN\_STR (field, val1, val2, ...) 11](#_Toc474337776)

[5.2.4 REGEX (field, regex) 11](#_Toc474337777)

[5.2.5 UNIQUE 11](#_Toc474337778)

[5.2.6 EXISTS (Not Yet Implemented as of version 1.3) 12](#_Toc474337779)

[6. Flags 13](#_Toc474337780)

[6.1 MUST\_IGNORE 13](#_Toc474337781)

[6.2 IS\_VALID 13](#_Toc474337782)

[6.3 EXISTS 13](#_Toc474337783)

[6.4 NOT\_EXISTS 13](#_Toc474337784)

[6.5 TBD 13](#_Toc474337785)

[6.6 NO\_REQUIREMENT 14](#_Toc474337786)

[7. Content Management Functions 15](#_Toc474337787)

[7.1 META (field, countofbytestoclear, blankvalue) 15](#_Toc474337788)

[7.2 HYPERLINK (Field, countofbytestoclear, blankvalue) 15](#_Toc474337789)

[7.3 REMEDYS 15](#_Toc474337790)

[7.3.1 REMEDY (field, countofbytestoclear, blankvalue) 15](#_Toc474337791)

[7.4 EMBEDDED (field, countofbytestoclear, blankvalue) 16](#_Toc474337792)

[7.5 MACRO (field, countof bytestoclear, blankvalue) 16](#_Toc474337793)

[7.6 REJECTED(field, contenttype) 16](#_Toc474337794)

[8. Conditionals 17](#_Toc474337795)

[9. DEFINE 18](#_Toc474337796)

[9.1 Plain DEFINE 18](#_Toc474337797)

[9.2 Macro DEFINE 18](#_Toc474337798)

[10. Array Slicing 19](#_Toc474337799)

[11. Special Tags 20](#_Toc474337800)

[11.1 %NOTE% 20](#_Toc474337801)

[12. BinPath Expression Language 21](#_Toc474337802)

[13. Including Files 22](#_Toc474337803)

[14. Tabs 23](#_Toc474337804)

[15. Ternaries 24](#_Toc474337805)

[16. Accessing Enumeration Members 25](#_Toc474337806)

[17. Using Batch Files 26](#_Toc474337807)

[18. REFERENCES 27](#_Toc474337808)

[19. APPENDIX 1. ‘HOW TO’ and Examples 28](#_Toc474337809)

# Overview

This guide is intended to provide advice on writing DVL and its intent is to produce DVL in a consistent format, and to provide the syntax of DVL.

## Syntax

The general syntax of a rule is

validation struct\_name {

field\_name := condition\_expression;

...

}

* field\_name is the name of the field being validated.
* The colon-equals (:=) symbol indicates what follows is the expression to be evaluated.
* condition\_expression is an expression that when it evaluates to true it means the field has valid content.
* condition\_expression is based on the Miraplacid BinPath Expression Language, with some additions detailed below.

>>>> Deprecated <<<<

In earlier versions of this guide this form of rules was permitted

A := 0;

or

A := 0 || 1 || 2;

Please **do not use** these forms anymore.

Instead, fully qualify them

A := current == 0;

Or

A := (current == 0) || (current == 1) || (current == 2);

NB: Always remember precedence of rules!! Add parentheses where clarification is needed.

## Values

Values can be expressed in decimal and hexadecimal using the standard C, C++, etc., format. Plain, unadorned decimal digits for decimal:

2345, 255, 78

And hexadecimal digits prefixed with 0x:

0x929, 0xFF, 0x4E

Uppercase A-F and lowercase a-f can be used for hexadecimal values.

## Validation

DVL is designed to validate the content of the fields. It does not concern itself with issues that an application (e.g., Office Word) is concerned with.

DVL is not concerned with validation that has been dealt with by the restrictions or AcceptRule expressions in its related BDDL file.

# Work Item Template Wording

When creating a Work Item, ensure you create it as a task and use this format for the Title:

Create DVL file for <structure-name>

This will make searches simpler.

As with the BDDL Work Items, populate the fields so:

Assigned To: Your name

Activity: Development

Area: Glasswall\BDDL Development\Office 1997-2007\MS-DOC

Iteration: 13nn [where nn is the current iteration number]

State: Active

Reason: New

Remember that the description field can be used to make more detailed comments about the structure you are developing. For example, you might want to mention some other structures that are included in your main structure. You can also attach files to the Work Item. Anything that helps other developers, and you, is worth adding to the Work Item.

# Substructures

If structure A contains structure B, then put B into a separate DVL file, but ensure you update the "Validation Files Log" file accordingly by completing the 'Substructure' column.

# Syntax

## General Notes

* If there is no MUST rule, use NO\_REQUIREMENT flag.
* Leave TBD as a flag to indicate further development is required (for example if the MUST rule is too complex).
* There is no need to append the structure name before each member:

Pms {

wpms := NO\_REQUIREMENT; # perfectly valid

...

}

Furthermore, the DVL checker raises an error if a dot is used before the ":="

## Comments

Comments can be inserted into DVL with a hash (#).

# This is a comment.

# Functions

Function names do not have an underscore between the words, and are in uppercase.

Functions can also be used as arguments to other functions, for example:

ipfnpmf := ISIN(current, 0xFF, RANGE(current, 0x00, 0x05));

## Office Word Functions

### VERSION(word-version)

RETURN TYPE: Boolean

DESCRIPTION: Returns true if the current document is the version of Office Word indicated by word-version.word-version can be one of:

Word1997, Word2000, Word2002, Word2003, Word2007

EXAMPLE: You want to test if the current document is version 2000:

VERSION(Word2000)

### GETCPCHAR(CP)

RETURN TYPE: Integer (width can vary)

DESCRIPTION: Retrieves the character specified by parameter CP and returns it.

EXAMPLE:

The requirement for field aCP in PlcfandRef as specified in [MS-DOC] §2.8.7 is:

"Each position in the main document specified by one of these CPs MUST be character 0x05...The last

CP MUST be ignored"

This would be specified in DVL as:

aCP[0..count(aCP) - 2] := GETCPCHAR(current) == 5;

## General Functions

### RANGE (field, min, max)

RETURN TYPE: Boolean

DESCRIPTION: If the content of 'field' has any of the values in the range min to max inclusive, the function returns TRUE, otherwise it returns FALSE. For example, the specification for Pmfs.ipfnpmf says:

An unsigned integer value that specifies the type of data source for the mail merge. This MUST be one of the following values: 0xFF, 0x00, 0x01, 0x02, 0x03, 0x04, and 0x05.

In DVL this can be represented as:

ipfnpmf := 0xFF || RANGE (current, 0x00, 0x05);

Or

ipfnpmf := ISIN (current, RANGE (current, 0x00, 0x05), 0xFF);

### ISIN (field, val1, val2,…)

RETURN TYPE: Boolean

DESCRIPTION: If the content of 'field' has any of the values listed, the function returns TRUE, otherwise it returns FALSE. For example, the specification for Wpms.wpmsType says:

An unsigned integer that specifies the document type of the mail merge. This value MUST be one of the following: 0x0, 0x1, 0x2, 0x4, and 0x8. In DVL this can be represented as:

wpmsType := ISIN (current, 0x0, 0x1, 0x2, 0x4, 0x8);

Or

wpmsType := ISIN ((current, 0, 1, 2, 4, 8);

Or

wpmsType := ISIN ((current, RANGE (current, 0, 2) 4, 8);

### ISIN\_STR (field, val1, val2, ...)

RETURN TYPE: Boolean

DESCRIPTION: If the content of string 'field' has any of the values listed, the function returns TRUE, otherwise it returns FALSE. Strings only in the test list

e.g.

xml\_tag1 := ISIN\_STR (current, "ALPHA", "BETA", "GAMMA");

### REGEX (field, regex)

RETURN TYPE: Boolean

DESCRIPTION: If the test string in 'regex' matches on the content of string 'field' the function returns TRUE, otherwise it returns FALSE.

e.g

xml\_tag2 := REGEX (current, "^\d\w+$");

### UNIQUE

RETURN TYPE: Boolean

DESCRIPTION: UNIQUE determines whether the value of a field is unique within all the structures specified.

EXAMPLE: Field ltag MUST be unique for all ATNBEs inside a given SttbfAtnBkmk. This will be written like this:

lTag := UNIQUE (ATNBE);

### EXISTS (Not Yet Implemented as of version 1.3)

RETURN TYPE: Boolean

DESCRIPTION: EXISTS determines whether the named node exists in the Binary DOM tree. It returns TRUE if it does exist, otherwise it returns FALSE.

EXAMPLE:

hsttbRfs := current == 0 : !EXISTS (root.tableStream.Pms.sttbfRfs) : EXISTS (root.tableStream.Pms.sttbfRfs);

# Flags

Flag names have an underscore between the words.

## MUST\_IGNORE

Specifies that the value must be ignored.

For example:

validation ... {

...

unused1 := MUST\_IGNORE;

...

}

MUST\_IGNORE is like a switch that allows to ignore some conditions under certain circumstances.

var1 := condition1 && MUST\_IGNORE;

Will apply the condition test. But we may optionally suppress the report of an issue at the request of the client if these generate too many false positives. In addition, if the condition is a test against a fixed value (e.g var1 := current == 5) then we will ‘Remedy’ the file by making that change.

var2 := condition2 ? MUST\_IGNORE : condition3;

In this case, we simply apply ‘condition3’ to ‘var2’ if ‘condition2’ is false, Must\_Ignore has no effect here.

## IS\_VALID

Specifies that the value is valid. It is equivalent to NO\_REQUIREMENT.

## EXISTS

Specifies that the field exists. For example, the specification for the tbcd field in the TBC structure says:

A structure of type TBCData, as specified in [MS-OSHARED] that contains toolbar control data. This MUST exist if tbch.tct is not equal to 0x16. This MUST NOT exist if tbch.tct is equal to 0x016 In DVL this can be represented as:

tbcd := tbch.tct != 0x16 ? EXISTS : NOT\_EXISTS;

## NOT\_EXISTS

Specifies that the value does not exist.

See the example from EXISTS.

## TBD

Specifies that the rule for the member has not been defined yet. If there is no MUST rule for the member, replace TBD with NO\_REQUIREMENT, otherwise replace with the appropriate rule. Fields that have the TBD flag will report an error.

## NO\_REQUIREMENT

Specifies that there is no MUST rule that needs to be tested. Safe to use with built in types (uint8, int8, ...) and user defined types (Wpms, Pmfs, ....).

# Content Management Functions

## META (field, countofbytestoclear, blankvalue)

USAGE:

field := META(current, 10, 0x00);

Specifies that the marked object contains Metadata which can be blanked.

Should only be used as specified on a line of its own.

EXAMPLE:

/\* fcGrpXstAtnOwners specifies an offset in the Table Stream. An array of XSTs begins at this offset.

This array contains the names of authors of comments in the document. \*/

validation GrpXstAtnOwners {

Xst[0 .. count(Xst)] := META(current, (current.cch \* 2), 0x00);

}

## HYPERLINK (Field, countofbytestoclear, blankvalue)

USAGE:

field := META(current, 10, 0x00);

Specifies that the marked object contains Metadata which can be blanked.

Should only be used as specified on a line of its own.

EXAMPLE:

/\* fcGrpXstAtnOwners specifies an offset in the Table Stream. An array of XSTs begins at this offset. This array contains the names of authors of comments in the document. \*/

validation GrpXstAtnOwners {

Xst[0 .. count(Xst)] := META(current, (current.cch \* 2), 0x00);

}

## REMEDYS

### REMEDY (field, countofbytestoclear, blankvalue)

USAGE:

field := REMEDY(current, 10, 0x00);

Specifies that the marked object contains a data which can be blanked.

Should only be used as specified on a line of its own.

Used like HYPERLINK and META but applied to padding areas at the end of structures

or strings to clear data that is suggested to be undefined and ignorable.

## EMBEDDED (field, countofbytestoclear, blankvalue)

USAGE:

field := EMBEDDED(current, 10, 0x00);

Specifies that the marked object contains embedded data which can be blanked. Should only be used as specified on a line of its own. Used like HYPERLINK and META but applied to padding areas at the end of structures or strings to clear data that is suggested to be undefined and ignorable.

## MACRO (field, countof bytestoclear, blankvalue)

USAGE:

field := MACRO(current, 10, 0x00);

Specifies that the marked object contains macro data which can be blanked. Should only be used as specified on a line of its own. Used like all other Mark and Convert Functions but applied to data structures, fields, records that either contain Macro content, or flags/values that imply Macros are present, which may trigger a Product to start looking for actual Macros that we have removed and warning user of their presence erroneously.

## REJECTED(field, contenttype)

USAGE:

field := REJECTED(current, MC\_EMBEDDEDFILE);

Specifies that the marked object contains data which can be removed from the output file. Should only be used as specified on a line of its own. Used like all other Mark and Convert Functions but applied to data structures, fields, records that either contain and manageable content. It works by setting the 'ucpObjectProcessStatus' in the current structure to 'eUcpObjectStatus\_CheckedAndRejected' therefore when the file is regenerated, this object and any of it's children will NOT be written to the output file.

This is intended to be used with text based file formats like XML or RTF so that we can cleanly remove "block start - data - block end" entities

The type of the content is marked using the contenttype parameter which MUST be one of the following.

MC\_EMBEDDED\_FILE

MC\_METADATA

MC\_HYPERLINK

MC\_FORMDATA

MC\_MACRO

IMPORTANT NOTE

--------------

Although you attach this rule to a field, that field must be a structure object and not a scalar object, because we need to set the 'ucpObjectProcessStatus' field in the base of that object.

# Conditionals

Conditionals can be created with the following syntax:

<member> := <condition> ? <expression 1> : <expression 2>;

This is equivalent to:

if (<condition>)

<member> := <expression 1>;

else

<member> := <expression 2>;

Note that if ... else ... is not available in DVL and is used for illustration only.

For example:

sttbfRfs := rfs.hsttbRfs == 0 ? MUST\_IGNORE : IS\_VALID;

In this example, Pms.sttbfRfs is ignored when parent.rfs.hsttbRfs is equal to 0, otherwise it is valid;

Conditionals can be nested to create more complex rules. The following format should be used stack them:

field1 := condition1 ? expression1

: condition2 ? expression2

;

Equivalent to:

if (condition1)

field := expression1;

else if (condition2)

field := expression2;

Note that if ... else ... is not available in DVL and is used for illustration only.

See Section 14 for further explanation of how to indent Ternary Operators.

# DEFINE

## Plain DEFINE

Some expressions will be repeated, and it makes sense to define them once and use that definition. The syntax is

DEFINE name = expression;

For example:

DEFINE ktCid = 0;

DEFINE ktChar = 1;

DEFINE ktMask = 3;

Then later on:

validation Kme {

...

param := ISIN(kt, ktCid, ktChar, ktMask);

...

}

This is only useful if you're likely to need ktCid, ktChar, or ktMask more than once in the same DVL file.

## Macro DEFINE

DVL allows "templates" to be created using place holders.

The format is:

DEFINE <functionname>(%1, %2 ....) = {BDDL\_expression};

For example:

DEFINE CALCNUMPLCS(%1, %2) = {(%1 - 4) / (4 + %2)};

Which can be used like this:

CALCNUMPLCS(field, 8);

Which will be converted to:

{(field - 4) / (4 + 8)};

This is a more realistic example:

CALCNUMPLCS(root.wordDocStream.fib.fibRgFcLcbBlob.fibRgFcLcb97.lcbPlcfBtePapx, 2);

The value returned will be the number of data elements in that PLC (assuming the size of a single data element in this example is 2).

# Array Slicing

In DVL it is possible to reference several elements of an array simultaneously using the ‘..’ (double-dot) notation. For example

aCP[0 .. 4]

is the same as

aCP[0]

aCP[1]

aCP[2]

aCP[3]

aCP[4].

Array slicing makes sense when comparing elements of two arrays, or the same array, simultaneously, for example:

arr[0 .. max-1] := current < arr[1 .. max];

This will test that

arr[0] < arr[1];

arr[1] < arr[2];

...

arr[max-1] < arr[max];

In other words, it tests that the array contents are in ascending order.

The same validation can be achieved by this expression:

arr := arr[0..max-1] < arr[1..max];

As another example, if you want to check each element is greater than 0:

arr[0 .. MAX\_INDEX] := current > 0;

The important thing to remember is that for an expression, each array slice generates a loop, so, consider this rather contrived example:

arr[0..max-2] := arr[1..max-1] < arr[2..max];

Will generate

arr[0] := arr[1] < arr[2];

arr[1] := arr[2] < arr[3];

...

arr[max-2] := arr[max-1] < arr[max];

# Special Tags

## %NOTE%

In order to facilitate the parsing of any issues that may need further consideration, prefix any specification with %NOTE% as the first part of a comment.

When using the %NOTE% option, leave the field the note refers to as TBD and DO NOT change it to NO\_REQUIREMENT which is incorrect in that circumstance.

Example:

In Ttmbd.dvl the field fcSubset can be any value, but the specification says, "This value MUST NOT exceed the total number of fonts used in the document." This may need to be validated later, so this is how it is specified in the DVL:

# %NOTE% This value MUST NOT exceed the total number of

# fonts used in the document.

fcSubset := TBD;

# BinPath Expression Language

The facilities of Miraplacid's BinPath Expression Language are available to use in DVL. For example, if you require a count of elements in an array, you can use the 'count' function:

# validate that aCP is in ascending order

# NOTE: count is number of elements, so index ranges from

# 0 to count - 1.

aCP[0 .. count(aCP) - 1] := aCP[0 .. count(aCP) - 2] < aCP[1 .. count(aCP) - 1];

Refer to the "Binary DOM Library" document on the Wiki for a list of all the functions and details of how to use the BinPath Expression Language.

# Including Files

In order to group commonly used DEFINEs you can use the ‘include’ construct to tell the DVL parser to search for matches to defines in the named files. The syntax is:

include “<filepath>”

Example:

include "office\_dvl/common/PlcDefines.dvl"

This assumes a softlink has been created. See "Howto\_softlink\_office\_bddl\_incldues\_dir" in Glasswall > BDDL Development > Howtos, but change the name of the soft link you create to 'office\_dvl' and the target directory to <Your workspace directory path>\glasswall.classic\data.definitions.and.rules\MSOffice\validation\.

# Tabs

Tabs do not appear the same when printed on different output devices, or pasted into a word-processor document, or even just viewed in someone else's differently tab spaced editor. So do not use tab characters in DVL files.

Most editors allow the insertion of spaces when the tab keys is pressed and you should set your editor to do this. Four space indentation should be used, so set the editor to insert four spaces.

# Ternaries

If you are creating long expressions using the ternary operator, use this style of indentation:

# An example of cascaded ternary operator, using field in Selsf

blktblSel := ( L == 0 && N == 0) ? MUST\_IGNORE

: ( L == 0 && N == 1) ? NO\_REQUIREMENT

: ( L == 1 && N == 0) ? NO\_REQUIREMENT

: ( L == 1 && N == 1) ? NO\_REQUIREMENT

;

Note how the terminating semicolon is lined up separately to make it clear the statement has finished.

If you have a final result (which has no preceding question mark) indent it so that it too lines up in the results column.

# An example of cascaded ternary operator, using field in Selsf

fInsEnd := current == 1 ? P == 1

: I == 1 ? current == MUST\_IGNORE

: 0

;

You can use the tabular layout even if you have only a single ternary:

A := current == 0 ? $customer{name}

: 'Sir or Madam'

;

Such formatting makes it even more important that Guideline 13 is followed.

# Accessing Enumeration Members

When an item is defined as an enum in BDD, the matching DVL must indicate which type the rule is being applied to by adding a means of selecting the type using a notation that provides the type.

For Example:

items.child$type

Example:

BDD

---

struct Cid{

(CidAllocated | CidMacro | CidFci | uint32) cid;

}

struct TBDelta{

..

Cid cid;

..

}

DVL

---

validation TBDelta {

..

cid := ISIN(cid.cid$uint32, 1, 3);

..

cid := sizeof(cid.cid$CidAllocated) < 200; /\*example for demo only, not part of the real dvl in this case\*/

..

}

This means that the rule is applied to the cid.cid field and treated as the uint32 type. It is the responsibly of the writer to ensure that the rule is only applied in a context where the field is of the type being tested.

# Using Batch Files

Using batch files in <tfs>\Features\OFFICE\<branch>\glasswall.classic\data.definitions.and.rules\BDD\_DVL\_to\_code\_scripts

There are two fundamental batch files in this support directory;

generate\_<filetype>\_cpp\_from\_bdd.bat and generate\_<filetype>\_cpp\_from\_dvl.bat

1. generate\_<filetype>\_cpp\_from\_bdd.bat - These batch files use the <tfs>\inhouse.tools\GWPerlScripts\BDD\_to\_Cpp\_Convertor\BDD\_To\_Cpp.pl Perl script to read the \*.BDD files and create the matching \*.CPP and \*.H files for each one, plus an \*\_if.CPP/H file and support files for the root BDD.
2. generate\_<filetype>\_cpp\_from\_dvl.bat - These batch files use the <tfs>\inhouse.tools\GWBinaryAnalysisTools\UCP\_Struct\_Rule\_Converters\dvl\_tools\dvl\_checker\Releases\Latest\_Version\dvl\_checker.exe to read the \*.BDD files and \*.DVL and create the validation \*.CPP file contains all of the validation methods for each structure in the BDD files.

NB These batch files default to using the dvl\_checker.exe defined in environment 'PATH\_TO\_DVLCHECKER' which they set to the one mentioned in this paragraph if 'PATH\_TO\_DVLCHECKER' is unset. (For testing use\_ucp\_dvlchecker.bat pre-sets 'PATH\_TO\_DVLCHECKER' to a different dvl\_checker version so you can call that before running the DVL processing batch file you are working on)

# REFERENCES

DVL Guide [this guide]

Glasswall > BDDL Development > Validation > DVL Guide

Validation Files Log

Glasswall > BDDL Development > Validation > Validation Files Log

Binary DOM Library

Glasswall > BDDL Development

# APPENDIX 1. ‘HOW TO’ and Examples

Example 1 - What information in a specification to place in a DVL Rule

----------------------------------------------------------------------

The specification:

2.9.343 VertMergeOperand

cb (1 byte): An integer value that specifies the byte count of the

remainder of this structure. This value MUST be 2.

itc (1 byte): An integer that specifies the index of a cell in the

table row. The first cell has an index of zero. All cells in the row

are counted, even if they are vertically merged with cells above or

below them. This value MUST be a valid index of a cell in the table

row.

vertMergeFlags (1 byte): A value from the VerticalMergeFlag

enumeration that specifies whether this cell is vertically merged

with the cells above or below it.

The DVL

validation VertMergeOperand {

cb := 2;

# %NOTE% itc An integer that specifies the index of a cell in

# the table row.

itc := NO\_REQUIREMENT;

vertMergeFlags := ISIN(current, 0, 1, 3);

Explanation:

It is important to note validation statements in the DVL even if you

cannot see how they might be implemented.

Example 2 - Avoiding out of bounds on arrays

--------------------------------------------

This rule suffers from a problem if the ‘my\_array\_name’ is zero in quantity

rulename := my\_array\_name[count(my\_array\_name)-1] != 0;

Please ensure that you prefix the test with a test for count(my\_array\_name) > 0 before trying to do the test.

This version checks there is at least one item in the array before trying to find the last one.

Otherwise simply return true regardless.

rulename := count(my\_array\_name) > 0 ? my\_array\_name[count(my\_array\_name)-1] != 0 : 1 ;

Example 3 - Testing presence of scalars

--------------------------------------------

If an item is a single byte field, the EXISTS/NOT\_EXISTS test does not apply as scalar fields are inserted

as permanent members of classes. So the DVL test on it below is of no use.

e.g.

This structure has a single byte 'padding' field

struct LPUpxChpx {

AcceptRule (parent.parent.parent.stdf.stdfBase.cupx\_istdNext & 0x000F) > 0;

uint16 cbUpx;

UpxChpx CHPX optional;

uint8 padding optional restrictions {parent.cbUpx % 2 !=0};

}

So a 'existance test' will not work and is redundant

e.g.

validation LPUpxChpx {

cbUpx := NO\_REQUIREMENT;

CHPX := NO\_REQUIREMENT;

>>>> padding := cbUpx % 2 !=0 ? EXISTS : NOT\_EXISTS; <<<<<<<<<<<< This Test made no sense and was removed.

}