



Data coercion

Data coercion

IMC-040B-UM.1

\$Date: 2025-04-24 16:07:39 +0200 \$

\$File: Coercion.xml \$

eRAD Inc

201 Brookfield Pkwy Suite 160
Greenville, SC 29607
www.erad.com

Copyright © 2010 eRAD Inc

eRAD Inc. copyrights this document and the associated software. Neither this document nor the associated software may be reproduced without the express written consent of eRAD Inc.

This document may include technical inaccuracies and/or typographical errors. The information herein changes periodically to address these issues and incorporates these changes into new editions of this document. eRAD Inc. may make improvements and/or changes to the product(s) and/or the program(s) described in this document at any time.

The following copyrights and trademarks are observed: PracticeBuilder 1-2-3, PracticeBuilder, ImageMedical.com, eRAD, eRAD PACS are registered trademarks of eRAD Inc.

Revision History

Revision 1.0	Nov 17, 2010	BM
Submitted		
Revision 1.0.1	Nov 18, 2010	BM
Biblio ID		
Revision 1.0.2	Dec 10, 2010	TN
Copyright information has been moved to individual documents		
Revision 1.1	Nov 19, 2015	BM
New functions, Viewer rules added		
Revision 1.2	Jun 11, 2018	BM
mul, div, mod functions added (medsrv-7.3.4 and medsrv-8.0.7)		
Revision 1.3	Jan 13, 2021	BM
toUpper and toLower functions added (medsrv-8.0.22)		
Revision 1.4	Oct 20, 2022	TN
Control variables (HPS-246)		
Revision 1.41	Dec 01, 2023	BM
Minor, 'scheduled for...' removed		
Revision 1.42	Aug 29, 2024	LGy
Added documentation about new control variable (@STUDYLEVELCOMMANDS)		
Revision 1.43	Oct 01, 2024	LGy
Added documentation about @STUDYRELEVANCY		
Revision 1.5	Apr 24, 2025	SZG (BM)
if/else/endif added (HPS-1517 and HPV-419)		

Table of Contents

1. Data coercion in general	3
2. Data coercion syntax	4
2.1. General Syntax	4
2.2. Data Types	4
2.3. Variables	4
2.4. lvalues	5
2.5. Values	5
2.6. Basic Functions	6
2.7. Logical Functions	6
2.8. String Functions	6
2.9. Date Functions	7
2.10. Arithmetic Functions	7
2.11. Encoding Functions	7
2.12. Misc Functions	8
2.13. Control Structures	8

1. Data coercion in general

Data coercion rules insert, remove and modify values in DICOM attributes when the system first acquires the object. Rules are assigned globally or to a specific registered device. Coercion works for DICOM alpha-numeric attributes only. Multiple rules can exist for a single device, and are applied in the defined sequence.

Three sets of global coercion rules exist. The first rule set is the Preceding Global Coercion Rules and is applied before any device-specific rules are applied. The second set of rules is the Trailing Global Coercion Rules, applied after the device-specific rules. Global coercion rules use the same commands as the device-specific rules. They are configured in the Preceding Global Coercion Rules and Trailing Global Coercion Rules sections of the Devices page. To assure administrators know these rules exist, they appear in uneditable tables before and after device-specific coercion rules on the Edit Device page.

The 3rd set is the Viewer Global Data Coercion Rules. These rules apply in the Viewer. That means they will not be saved to the study data, but modify the appearance of the study in the Viewer.

Device-specific coercion rules are available for registered DICOM devices. They get applied after preceding global coercion rules and before trailing global coercion rules.

To define coercion rules, perform the following steps.

1. From the Admin tab, click the Devices tab.
2. If defining global coercion rules,
 - 2.1. Scroll to the Preceding Global Coercion Rules section or the Trailing Global Rules section or the Viewer Global Data Coercion Rules.
 - 2.2. Define and save coercion rules as described below.
3. If defining device-specific coercion rules,
 - 3.1. Find the device on the Devices table and click on the edit button, , to the left of the name.
 - 3.2. Scroll to the Data Coercion Rules section.
 - 3.3. Define and save coercion rules as described below.

To define coercion rules, perform the following steps.

1. To see a list of available coercion commands, click the Help button.
2. Type the rules into the text box. Use the pulldown list and Add Tag button to select the attribute tag, if needed.
3. Click Save. If there is a detectable Syntax error in the created coercion rule, it will be displayed after pressing Save (but the coercion with the bad syntax will not be saved).

Note: Data coercion rules do not propagate between servers. If they are required on multiple eRAD PACS servers, you must make the change on each one.

2. Data coercion syntax

2.1. General Syntax

comment lines

Lines starting with '#' are treated as comments

<lvalue>=<expression>

Expression can be either a Value or a Function.

control structures

See Control Structures

2.2. Data Types

String

Group of ASCII-numeric and control characters. When non-contiguous, strings must be encapsulated within double quotes (""). Control characters must be preceded by a backslash, as in "\n", "\\" and "\".

Integer

Numeric value used and returned in some expression functions, including arithmetic functions.

gTag

The first part of a DICOM tag representing the attribute group. Expressed as four hex digits.

eTag

The second part of a DICOM tag representing the attribute element. Expressed as four hex digits.

2.3. Variables

Temporary variables

A variable used to temporarily hold an evaluated expression for reuse in later rules. Temporary values are not stored in the object. Variables can appear either as lvalues or in expressions.

Control variables

Variable names started by a '@' sign are control variables.

The software sets a default value to the control variables before invoking the coercion rules. The coercion rules can refer to or set control variables similarly to temporary variables.

After the evaluation completes, the software checks the result and acts accordingly.

Control variables:

@PROCESS

Controls whether processing the object continues or not. That is the object would be stored, forwarded, etc according to the context.

If the control variable evaluates to `NULL ( )`, then processing the object stops.

If the control variable evaluates to `true` (that is not `NULL ( )`), then the object will be processed (stored, forwarded, ...) as usual.

@STUDYLEVELCOMMANDS

This control variable can be used to execute macro commands in the viewer. Multiple macros can be specified, separated by semicolons.

Each macro is executed once per study, evaluated using the first DICOM object in each study. Execution happens after parsing all studies, but before showing any images.

In this example, we call the `SetDropXML` macro to turn off the "XML Interface Speech Recognition" engine.

```
$(@STUDYLEVELCOMMANDS)="SetDropXML(\"StudyOpen\", \"\", \"XML Interface Speech Recognition\");"
```

@STUDYRELEVANCY

The viewer sets the `@STUDYRELEVANCY` variable based on the relevancy information within the pbs. Possible values are made up of the relevancy level ("Current", "Prior", "Unrelated") and the position of the study within the level.

Examples:

```
"Current 0"
"Current 1"
"Prior 0"
"Prior 2"
"Unrelated 0"
```

2.4. Ivalues

(gTag,eTag)

DICOM attribute tag. The tag value for DICOM defined attributes and eRAD PACS's pre-defined user fields, User1-10. This lvalue is in the form "(gggg,eeee)".

SEQ(gTag1,eTag1, INo1,gTag2,eTag2 [,INo2,gTag3,eTag3...])

A target attribute specified by Tag2 within the INo item in the sequence specified by Tag1, where 0 represents the first item in the sequence. If the sequence Tag1 does not exist in the object, the entire rule is ignored.

USER(fieldName)

User defined field. A custom configured user field, specifically, one that do not exist in the DICOM library. This lvalue is in the form "USER(fieldName)", where fieldName is the Field Name defined in the custom user fields configuration file.

\$(varName)

Sets the value of the *varName* variable.

Note that in case of control variables the variable name starts with a '@' sign. (`$(@varName)`).

2.5. Values

(gggg,eeee)

Returns the value of a DICOM tag. If the DICOM tag does not exist, `NULL ( )` is returned.

SEQ(gTag1,eTag1, INo1,gTag2,eTag2 [,INo2,gTag3,eTag3...])

Returns the contents of the DICOM attribute specified by Tag2 within the sequence specified by Tag1. INo indicates the sequence item instance, where 0 represents the first item in the sequence. Returns `NULL ( )` if the attribute does not exist in the INo item of the sequence.

USER(fieldName)

Returns the value of a user defined field. If the field does not exist, `NULL ( )` is returned.

\$(varName)

Returns the value of the *varName* variable. If the variable is not set, `NULL ( )` is returned.

Quoted string

This is in the form, "string". Returns the literal string *string*. The string can contain escaped characters, including "\n", "\" and \"\".

Unquoted string

Non-quoted strings can be used when they contain only contiguous, alphanumeric characters.

(gggg,eeee),"d",n (Retired)

This returns the nth field in the DICOM tag (gggg, eeee) value as separated by the delimiter d.

2.6. Basic Functions

NULL()

Deletes the target lvalue.

empty string

The empty string, "".

2.7. Logical Functions

and(a,b)

Returns true if both a and b are non-NULL.

equals(a,b)

Returns true if a and b are equal, NULL if they are not equal.

if(cond,a,b)

Returns a if cond is not NULL, b if cond is NULL.

not(a)

Returns true if a is NULL, NULL if a is not NULL.

or(a,b[,c...])

Returns the first non-NULL value.

2.8. String Functions

concat(a,b[,c...])

Concatenates the values a and b (and c, etc.) NULL values are treated as an empty string, "".

contains(a,b)

Returns b if b is found in a, NULL otherwise.

indexof(a, ss)

Returns the starting position of ss in a, or -1 if not contained.

split(a,d,n)

Returns the *n*th field in *a* using *d* as the field delimiter.

strlen(s)

Returns the length of *s*, or NULL if *s* is NULL.

substr(s, p [,n])

Returns the given section of *s* starting from the position *p* (0 based, 0 means the beginning of the string), in the length specified by *n*.

translate(a,d,i1,o1 [,i2,o2...])

Returns the output value, *oN*, if the input string, *iN*, matches the source string, *a*. If the source string matches no input string, the function returns the default value, *d*. Source string, *a*, and input strings, *iN*, must be String types or NULL(). The default value, *d*, and output values, *oN*, can be any type but must all be of the same type.

toUpper(s)

Returns the upper-case version of *s*.

toLower(s)

Returns the lower-case version of *s*.

2.9. Date Functions

dicomAge(d1, d2)

Returns the value *d1* - *d2* in calendar years, months or days in DICOM-compliant format: *nnnY*, *nnnM*, or *nnnD*. If either *d1* or *d2* is an invalid date value, or *d2* predates *d1*, NULL is returned.

2.10. Arithmetic Functions

add(n,m[,o...])

Returns the sum of the parameter integer strings.

sub(n,m)

Returns *n* - *m*, where *n* and *m* must be integer strings.

between(v,n,m)

Returns true if *v* is greater than or equal to *n* and *v* is less than *m*, NULL otherwise. *v*, *n* and *m* are integer strings.

mul(n1,n2[,n3...])

Multiply all operands. Operands must be Number types.

div(n1,n2)

Divide the operands using integer division, *n1/n2*. Operands must be Number types. Division by zero is checked, but zero-value denominators will return an error.

mod(n1,n2)

Return the remainder of the integer division, *n1/n2*. Operands must be Number types. Division by zero is checked, but zero-value denominators will return an error.

2.11. Encoding Functions

codenumber(n)

n must be a *>= 0* integer string, returns a coded *>= 0* integer string with the same number of digits.

codestring(s[,exc])

s is a string, returns a coded string with the same length. If *exc* is used, the result will not include these characters.

2.12. Misc Functions

rnd(n,seed)

Returns a random number (integer string) between 0 and n-1. If **seed** is used, the random number will be calculated from it.

2.13. Control Structures

if(<expression1>) <lvalue>=<expression> [<lvalue>=<expression>, ...] [else <lvalue>=<expression> [<lvalue>=<expression>, ...]] endif

Conditional statements. If expression1 is not null, the statement(s) until 'else' or 'endif' are executed. If expression1 is null and there is 'else' part, the statement(s) between 'else' and 'endif' are executed.

Example:

```
if((0008,0050))
  (0008,0050)=concat("PFX",(0008,0050))
else
  (0008,0050)="new"
endif
```