Nicl Function Library

Sarah Elmendorf, Claire Lunch, and Corey Morgan

11/27/2018

Nicl is the NEON Ingest Conversion Language. It is the scripting language used to write validation and conversion rules for observational data to be ingested into the NEON database. The OS Parser is the software that ingests the data and stores it in the database; all user-modifiable instructions to the OS Parser are written in OS ingest workbooks. Within the workbooks, Nicl is used in the validationRulesParser and parserToCreate fields.

It is highly recommended to read this document while referencing at least one ingest workbook or validation file (an abbreviated version of an ingest workbook, provided along with data downloads from the NEON data portal). Seeing the functions as used in practice will be far clearer than reading this document alone.

This document is an index of available Nicl functions and practical considerations for their usage. For a more detailed description of the software underlying Nicl, see the Nicl Language document.

Table of Contents

1. General principles

For validation rules, a data upload that fails to meet the criteria of any one rule will result in failure to ingest the entire data upload. However, to the extent possible, the failure report will include all rules failed, to enable efficient resolution of failures. There is no warning capability; validation either passes or fails.

For creation rules, a data upload that fails to create a value will result in a blank field. If the field in question is required, either implicitly (date and location fields) or explicitly (using the validation function REQUIRE), the upload will fail.

Creation functions must be written in the parserToCreate column; validation functions must be written in the validationRulesParser column. Functions placed in the wrong column will not be carried out.

Validation and creation rules are written in square brackets. Multiple rules can be applied to the same field, in sequential sets of square brackets, e.g. [REQUIRE][GREATER_THAN(0)].

Inputs to functions can be numbers, strings, or fieldNames of other fields in the same workbook table. Strings must be in single quotes, e.g. [DEFAULT_TO('active')].

2. NEON functions

2.1. Samples

Sample data are central to almost all NEON observational data. Controlling the values of sample identifiers, connecting data collected on the same samples at different times, and maintaining hierarchies of samples and their subsamples, are all essential to the integrity of observational data. These functions are part of the system that instructs the OS Parser in how to handle sample-related data.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
EXISTS	Validation	NA	[EXISTS]	This sample must already be present in the NEON database.	See **Sample Existence Rules** below.
DOES_NOT_EXIST	Validation	NA	[DOES_NOT_EXIST]	This sample must not be present in the NEON database, nor elsewhere in the current upload.	See **Sample Existence Rules** below.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
DOES_NOT_EXIST_IN_DB	Validation	NA	[DOES_NOT_EXIST_IN_DB]	This sample must not be present in the NEON database, but may be present elsewhere in the current upload.	See **Sample Existence Rules** below.
MIGHT_EXIST	Validation	NA	[MIGHT_EXIST]	This sample may or may not be present in the NEON database.	Avoid using unless strictly necessary. See **Sample Existence Rules** below.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
DERIVE_FROM_SAMPLE_TREE	Creation	At least one valid sampleClass. Multiple sampleClasses are separated by OR.	[DERIVE_FROM_SAMPLE_TREE('swc' OR 'gsi')]	Allows date and location data to be inherited through a sample hierarchy. Populate by finding the values in data associated with samples of the input sampleClass, which must be either the class of the primary sample, or of one of its ancestors. The input class must exist in a table where the date and location were NOT derived. If there are multiple ancestor samples of the input class, the earliest start and latest end dates will be used, and the lowest location on the location hierarchy that encompasses all the locations appearing in the data.	Can only be applied to fields of fieldType activityStartDate, activityEndDate, and namedLocation.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
SPLIT_BY	Creation	The string to split by, generally a pipe.	[SPLIT_BY('|')]	Split a concatenated field into multiple elements.	Currently only valid for fields of fieldType childSampleTag, parentSampleTag. Allows for creating multiple child samples in a single field, or designating multiple parent samples for a mixture.
SAMPLE_CLASS	Creation	A sampleGroup from the same workbook.	[SAMPLE_CLASS('fieldSample')]	Populates with the class of the sample whose sampleGroup is input.	Only valid on samples included in the same data record.

Sample Existence Rules

There are 3 locations where a sample may occur prior to its upload in a given data table:

1. In the sample table ('in database')
2. In another data table in the same upload, with a lower rank number in the sequencing workbook ('in previous table')
3. Elsewhere in the same data table, in the same upload ('in current table')

EXISTS, DOES_NOT_EXIST, DOES_NOT_EXIST_IN_DB, and MIGHT_EXIST describe which of these occurrences should cause success or failure of a data upload.

DOES_NOT_EXIST – 1, 2, or 3 all fail

Sample should not be present in the database, in a previous table in the same upload, or anywhere else in the current table. Sample presence in the

database or a previous table means upload fails; more than one instance of sample in the current table means upload fails.

DOES_NOT_EXIST is almost always the validation used when a sample is created - when it is first collected in the field, or first subsampled from another sample. This prevents duplicate samples from being entered into the database.

DOES_NOT_EXIST_IN_DB – 1 or 2 fail, 3 passes

Sample should not be present in the database or in a previous table in the same upload. Multiple instances of sample in the current table are allowed.

DOES_NOT_EXIST_IN_DB is used when samples are first created, but they are created in multiples. For instance, in the mosquito identification data table, samples are pooled to create a child sample, with a data record created for each parent sample, so the child sample appears in as many records as it has parents. DOES_NOT_EXIST_IN_DB ensures that all parents of a given child sample are uploaded together, and child sample identifiers can't be duplicated between different data uploads.

EXISTS – 1 or 2 passes, 3 is not checked

Sample should be present either in the database or in a previous table in the same upload. If sample is not present in one of these two places, upload fails. Multiple instances in the current table are irrelevant to pass or fail.

EXISTS is used when samples are referenced again after being created, and ensures sample identifiers match. For example, EXISTS is used to ensure data returned by analytical facilities matches sample identifiers collected by NEON.

MIGHT_EXIST - 1, 2, or 3 pass

Sample may or may not exist in the database, in a previous table in the same upload, or in multiple instances in the current table. All scenarios pass.

MIGHT_EXIST is used when a sample may be created, or may already be present in the database. For example, in the woody vegetation structure mapping and tagging data table, a given tree may have already been identified and tagged in a previous year, or it may have grown between sampling years, in which case it will be a newly recorded sample.

Syntax

Sample existence rules are placed in the workbook in the sample tag field, but they apply to the sample as a whole; i.e., if sample data are uploaded with only a barcode, and no tag, the existence rules will be applied as usual.

Matching for sample tags is case-insensitive, but all letters are converted to upper case on storage in the database.

2.2 Taxonomy

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
ELEMENT_OF	Validation	Taxonomy type, taxon value (taxonID or scientificName), filter local (T or F)	[ELEMENT_OF('MOSQUITO', 'scientificName',F)]	Value in field must match a taxonID or scientificName in the taxonomy table indicated. If filterLocal=T, the taxon must be present at the site in question.	All inputs are required. filterLocal=F can cause problems in the transition from L0 to L1 data, so use filterLocal=T unless there is a compelling reason not to.

2.3 File paths

Select types of data are stored in cloud storage buckets, instead of being parsed and stored in the database tables. For example, the digital hemispherical photos can't be parsed into database tables, so they are stored as files. These functions ensure that data are present in the appropriate buckets, and provide the URLs to end users to access the data.

Note that because data are placed in cloud storage directly, they bypass many validation steps. The validations described here only confirm that the files exist and are of the correct type. Because of this serious limitation, this option is used only when the file size and/or data structure are not compatible with the OS data model.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
URI	Creation	Concatenated string to create file path	[URI('https://s3.data.neonscience.org/neon-dhp-images') + '/' + fulcrumFilePath]	Create URL where data can be found.	This function only makes the URL string, it doesn't put any data there.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
VERIFY_URL	Validation	NA	[VERIFY_URL]	Check that the URL works.	Used in combination with MIME_TYPE, see next line.
MIME_TYPE	Validation	Valid mime types	[MIME_TYPE('image/nef' OR 'image/jpg' OR 'image/jpeg')]	Check that the data found at the URL match one of the mime types.	
S3_DATA_FRAME_URL	Creation	NA	[S3_DATA_FRAME_URL + dnaSampleID + '_16S_' + [NOW] + '.csv']	Create base URL and concatenate with other strings to generate full URL where data are to be put.	The data frame process is complicated, to use this workflow consult the data frame instructions document.
GROUP_BY_SAMPLE	Creation	A sampleGroup	[GROUP_BY_SAMPLE('sample')]	Define the grouping variable for data frame creation.	The data frame process is complicated, to use this workflow consult the data frame instructions document.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
TO_DATA_FRAME_COLUMN	Creation	The field to group by and the name to give it in the data frame	[TO_DATA_FRAME_COLUMN(sampleID, 'sampleID')]	Write the grouping variable to the data frame.	The data frame process is complicated, to use this workflow consult the data frame instructions document.

2.4 Other

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
DEFAULT_TO	Creation	Value to default to - can be a string, number, date, or fieldName	[DEFAULT_TO('active')]	Populates with the value indicated, if the field is blank when uploaded.	Does not change the value of a field that is already populated.
"	Creation	Value to populate with - can be a string, number, date, or fieldName	['active']	Populates with the value indicated.	Overwrites value if the field is already populated.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
DEFAULT_TO_LAB_LOGGED_IN	Creation	NA	[DEFAULT_TO_LAB_LOGGED_IN]	Populates with the name of the lab uploading data - selected from dropdown menu in upload UI.	Does not change the value of a field that is already populated. Do not have the lab pre-populate this field, lab names need to match exact strings in the NEON database.
CREATE_UID	Creation	NA	[CREATE_UID]	Populates with a randomly generated unique identifier, used to create the unique identifier for the record of data.	Currently only used for fieldName=uid.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
NAMED_LOCATION_TYPE	Validation	One or more NEON named location types	[NAMED_LOCATION_TYPE('OS Plot - mam' OR 'AOS buoy named location type')]	Checks that the value is a valid named location of one of the indicated types.	This validation should be applied to ALL named location fields, including named location fields populated via DERIVE_FROM_SAMPLE_TREE. Can be applied to named location fields even if they are not fieldType=namedLocation.
REQUIRE	Validation	NA	[REQUIRE]	Field must be populated.	Populating with an empty string or NA fails as if the field were blank.
REQUIRE_NULL	Validation	NA	[REQUIRE_NULL]	Field must be blank.	Typically used as the outcome of a conditional, e.g. [IF(fieldName='active'), REQUIRE_NULL]. See guidelines for conditionals in Section 7.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
IS_BLANK	Validation	Optional: either no input, or a fieldName.	[IS_BLANK] or [IS_BLANK(fieldName)]	Checks whether a field is populated.	Typically used as input to a conditional, e.g. [IF(IS_BLANK(fieldName)), REQUIRE]. When used without a fieldName, the current field is evaluated. Do not use with the current fieldName as input; function will not work.
IS_NOT_BLANK	Validation	Optional: either no input, or a fieldName	[IS_NOT_BLANK] or [IS_NOT_BLANK(fieldName)]	Checks whether a field is populated.	Typically used as input to a conditional, e.g. [IF(IS_NOT_BLANK(fieldName)), fieldName + otherField-Name]. When used without a fieldName, the current field is evaluated. Do not use with the current fieldName as input; function will not work.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
LOV	Validation	NA	[LOV]	Field must match an element in a pre-determined list of values.	The validation takes no input, but the name of a list of values is required in the lovName column of the workbook, to identify the list of values to validate against. Match is case-insensitive.

3. Logical functions

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
=	Validation	Two values to be compared	[IF(fieldName=0), REQUIRE]	Check for equality of two values.	Typically used as input to a conditional. See guidelines for conditionals in Section 7.
NOT	NA	Logical value to be reversed	[IF(NOT(fieldName=0)), REQUIRE]	Convert TRUE to FALSE and vice versa.	Typically used as input to a conditional.
AND	Validation	Two values to be compared	[IF(fieldName=0 AND otherFieldName='Yes'), REQUIRE]	Compare two logical values and return TRUE if both are TRUE	Typically used as input to a conditional.
OR	Validation	Two values to be compared	[IF(fieldName=0 OR otherFieldName='Yes'), REQUIRE]	Compare two logical (TRUE/FALSE) values and return TRUE if either is TRUE	Typically used as input to a conditional.

4. Numeric functions

In addition to validating numeric values, Nicl can perform basic arithmetic and a few simple functions.

Negative numbers are indicated by a - sign following, rather than preceding, the number. See details under GREATER_THAN/LESS_THAN.

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
GREATER_THAN; GREATER_THAN_OR_EQUAL_TO	Validation	Value to compare to, can be a number or a fieldName	[GREATER_THAN(5)]	Check if value exceeds a minimum threshold.	Syntax for comparison to a negative number: [GREATER_THAN(5-)]
LESS_THAN; LESS_THAN_OR_EQUAL_TO	Validation	Value to compare to, can be a number or a fieldName	[LESS_THAN(5)]	Check if value falls below a maximum threshold.	Syntax for comparison to a negative number: [LESS_THAN(5-)]
+-*/^	Creation	Numbers and/or fieldNames	[((fieldName + 5)*otherFieldName)^2]	Standard arithmetic functions.	Note that blanks behave differently in these functions than in SUM, MEAN, etc. See null handling guidelines in Section 8.
COUNT	Creation	FieldName(s)	[COUNT(fieldName, otherFieldName)]	Counts the number of non-blank values in the fields indicated.	
SUM	Creation	FieldName(s)	[SUM(fieldName, otherFieldName)]	Sums the values in the fields indicated.	
MEAN	Creation	FieldName(s)	[MEAN(fieldName, otherFieldName)]	Calculates the arithmetic mean of the values in the fields indicated.	

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
STDDEV	Creation	FieldName(s)	[STDDEV(fieldName, otherFieldName, thirdFieldName)]	Calculates the standard deviation of the values in the fields indicated, using n-1 in the denominator.	
PSTDDEV	Creation	FieldName(s)	[STDDEV(fieldName, otherFieldName, thirdFieldName)]	Calculates the 'population' standard deviation of the values in the fields indicated, using n in the denominator.	

5. String functions

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
+	Creation	Two or more strings and/or fieldNames	[fieldName + otherFieldName]	Concatenate two values.	
MATCH_REGULAR_ EXPRESSION	Validation	A regular expression to be validated against	[MATCH_REGULAR_ EXPRESSION('WDP.[A-Z]{4}.20[0-9]{2}(0[1-9]|1[012])(0[1-9]|1[0-9]|2[0-9]|3[01]).(0[0-9]|1[0-9]|2[0-4])[0-5][0-9]')]	Validate that the value in the field matches the regular expression.	

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
NOT_MATCH_REGULAR_EXPRESSION	Validation	A regular expression to be validated against	NOT_MATCH_REGULAR_EXPRESSION('N/A|NA')]	Validate that the value in the field does not match the regular expression.	
STARTS_WITH	Validation	A string or a fieldName	[STARTS_WITH(fieldName)]	Validate that a string value starts with the exact string in another field.	
ENDS_WITH	Validation	A string or a fieldName	[ENDS_WITH(fieldName)]	Validate that a string value ends with the exact string in another field.	
ASCII	Validation	NA	[ASCII]	Validate that the field contains only ASCII characters.	It's generally useful to apply this validation to every free text field.

6. Date functions

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
NOW	Creation	NA	[NOW]	The current date and time, to the millisecond, in UTC.	

Function	Creation or validation function?	Inputs	Example	Description	Special considerations
UPLOAD_DATE	Creation	NA	[UPLOAD_DATE]	The date and time of the data upload, to the millisecond, in UTC.	Typically used to set the activity start and end dates for lab-specific files.
CONVERT_TO_UTC	Creation	Either 'namedLocation' or a fieldName that contains a named location	[CONVERT_TO_UTC(namedLocation)] or [CONVERT_TO_UTC(laboratoryName)]	Converts the input date, in local time, to UTC. Uses the named location to determine the time zone of local time.	
DAYS; HOURS; MINUTES; SECONDS	Creation	NA	[ovenEndDate - collectDate][DAYS]	Indicate the units for a calculated time interval	

6.1. Date formats

Date formats in input data can be in the following formats, and will be interpreted as indicated.

The first column is a date or date/time example as would be expected to be seen within a csv cell. The second column is the canonical interpretation of the date/time (that is, what the OS Parser sees). The third column tells whether the parser recognizes that there is a time component in this date, and the fourth column indicates whether the parser recognizes the time zone.

Input date Time in CSV Cell	Parser Interpretation	Has Time?	Has Time zone?
`2016-05-05T22:59:38Z`	2016-05-05T22:59:38Z	Yes	Yes
`2016-10-12 09:08MST`	2016-10-12T09:08-06:00[America/Denver]	Yes	Yes
`2009-11-15T`	2009-11-15	No	No
`2009-11-15 3:42 am`	2009-11-15T03:42	Yes	No
`2009-11-15 03:42:45.55234 PM`	2009-11-15T15:42:45.552340	Yes	No
`2009-11-15 03:42:45.55234P`	2009-11-15T15:42:45.552340	Yes	No
`2009-11-15`	2009-11-15	No	No
`2009-11-15 03:42:45 AM`	2009-11-15T03:42:45	Yes	No

Input date Time in CSV Cell	Parser Interpretation	Has Time?	Has Time zone?
2009-11-15 03:42	2009-11-15T03:42	Yes	No
2009-11-15 1:42	2009-11-15T01:42	Yes	No
2009-11-15 15:42	2009-11-15T15:42	Yes	No
2009-11-15T14:12:12	2009-11-15T14:12:12	Yes	No
2009-11-15T14:12:12.3449	2009-11-15T14:12:12.344900	Yes	No
2009-11-15T14:12:12Z	2009-11-15T14:12:12Z	Yes	Yes
2009-11-15T14:12:12UTC	2009-11-15T14:12:12Z[UTC]	Yes	Yes
2009-11-15 14:12:12 EDT	2009-11-15T14:12:12-05:00[America/New_York]	Yes	Yes
2009-11-15T14:12:12+01:00	2009-11-15T14:12:12+01:00	Yes	Yes
2009-11-15T14:12:12-07:30	2009-11-15T14:12:12-07:30	Yes	Yes
20091115	2009-11-15	No	No
20091115T	2009-11-15	No	No
20091115 3:42 am	2009-11-15T03:42	Yes	No
20091115 03:42:45.55234 PM	2009-11-15T15:42:45.552340	Yes	No
20091115 03:42:45.55234P	2009-11-15T15:42:45.552340	Yes	No
20091115 03:42:45 AM	2009-11-15T03:42:45	Yes	No
20091115 03:42	2009-11-15T03:42	Yes	No
20091115 1:42	2009-11-15T01:42	Yes	No
20091101 15:42	2009-11-01T15:42	Yes	No
20091115T14:12:12	2009-11-15T14:12:12	Yes	No
20091115T14:12:12Z	2009-11-15T14:12:12Z	Yes	Yes
20091115T14:12:12UTC	2009-11-15T14:12:12Z[UTC]	Yes	Yes
20091115 14:12:12 EDT	2009-11-15T14:12:12-05:00[America/New_York]	Yes	Yes
20161012 09:08MST	2016-10-12T09:08-06:00[America/Denver]	Yes	Yes
20091115T14:12:12+01:00	2009-11-15T14:12:12+01:00	Yes	Yes
20091115T14:12:12-07:30	2009-11-15T14:12:12-07:30	Yes	Yes
20091115 14:42am	2009-11-15T14:42	Yes	No

This table lists some values that may look like valid dates but actually are not

Incorrect input date Time	What's the matter?
2016-05-05T22:59:38ZQAA	Trash at the end of the date
16-10-05T22:59	Two-digit years are so last millennnium
2009-9-31	30 days hath September
2009-11-15 T	Floating "T"

Incorrect input date Time	What's the matter?
`2009-11-15 14:42 AM`	14:42 is not in the AM
`2009-11-15 10:42 am`	Lowercase am
`2016-13-15 10:42 AM`	Thirteen months in a year may *sound* like a good idea, but no parsing for you.
`06/29/2016`	Sorry...
`29/06/2016`	...I don't even...
`14-May-2016`	...nope...
`2016-JUN-30`	...also nope...
`2016JUN30`	... nor this. All dates and date times *must* start with the "yyyy-MM-dd" or "yyyyMMdd" format.

7. IF statements

Any Nicl function with a logical output can be used in an IF statement. But, many Nicl functions behave slightly differently in an IF statement than outside of one. In an IF statement, a function's output will be interpreted as a logical value, whereas outside an IF statement, a TRUE silently passes the field's original value, and a FALSE fails data ingest.

IF can be used in both creation and validation functions. In validations, it is most frequently used to make certain fields required or un-required, depending on the contents of other fields.

The table below contains examples; see Null handling in Section 8 below for additional considerations.

Function	Description
[GREATER_THAN(5)]	Validation fails and data are not ingested unless input value is greater than 5
[IF(LESS_THAN(5)), 0]	If the input value is less than 5, it is overwritten by 0
[IF(fieldName < 5), 'belowThreshold']	If the value in fieldName is less than 5, this field is populated by the string 'belowThreshold'
[IF(fieldName = 'Yes'), REQUIRE]	If the value in fieldName is Yes, this field must be populated
[IF(IS_BLANK(fieldName)), REQUIRE]	Either this field or fieldName must be populated

8. Null handling

A few Nicl functions (REQUIRE, REQUIRE_NULL, IS_BLANK, IS_NOT_BLANK) are explicitly about whether a field is populated or not. Other Nicl functions handle null values according to the following rules:

Validation functions carry an implicit IF(IS_NOT_BLANK), e.g., LESS_THAN(fieldName) will pass validation if either fieldName or the field being compared to it is blank.

Inside an IF statement, a blank returns FALSE, i.e., the -then part of the if-then will not be carried out. When combining logical functions, e.g. IF(fieldName=0 AND otherFieldName=10), a blank is still FALSE.

In parser math, functions like SUM, MEAN, etc will ignore blank fields and carry out their calculations on however many populated fields are available from their inputs. In contrast, arithmetic expressions like + - * / will output a blank if there is a blank in any of their inputs.