CRM Parser Manual

Written by Ahmed Elsawalhy (Yagasoft.com)

V1.1

Contents

[2. Introduction 7](#_Toc80451557)

[3. Terms and Structures 8](#_Toc80451558)

[4. Algorithm 9](#_Toc80451559)

[5. Reserved 10](#_Toc80451560)

[6. Relationship Traversal 11](#_Toc80451561)

[7. Expressions 12](#_Toc80451562)

[8. Caching 13](#_Toc80451563)

[9. Constructs 14](#_Toc80451564)

[9.1. Complete Form 14](#_Toc80451565)

[9.2. Template 14](#_Toc80451566)

[Definition 14](#_Toc80451567)

[Example 14](#_Toc80451568)

[Parameters 14](#_Toc80451569)

[Logic 14](#_Toc80451570)

[Output 14](#_Toc80451571)

[9.3. Placeholder 14](#_Toc80451572)

[Definition 15](#_Toc80451573)

[Example 15](#_Toc80451574)

[Parameters 15](#_Toc80451575)

[Logic 15](#_Toc80451576)

[Output 15](#_Toc80451577)

[9.4. Reference 15](#_Toc80451578)

[Definition 15](#_Toc80451579)

[Example 15](#_Toc80451580)

[Parameters 16](#_Toc80451581)

[Logic 16](#_Toc80451582)

[Output 16](#_Toc80451583)

[9.5. Column 16](#_Toc80451584)

[Definition 17](#_Toc80451585)

[Example 17](#_Toc80451586)

[Parameters 17](#_Toc80451587)

[Logic 17](#_Toc80451588)

[Output 17](#_Toc80451589)

[9.6. Info 17](#_Toc80451590)

[Definition 17](#_Toc80451591)

[Example 18](#_Toc80451592)

[Logic 18](#_Toc80451593)

[Output 18](#_Toc80451594)

[9.7. Preload 18](#_Toc80451595)

[Definition 18](#_Toc80451596)

[Example 18](#_Toc80451597)

[Logic 18](#_Toc80451598)

[Output 18](#_Toc80451599)

[9.8. Discard 18](#_Toc80451600)

[Definition 18](#_Toc80451601)

[Example 18](#_Toc80451602)

[Logic 19](#_Toc80451603)

[Output 19](#_Toc80451604)

[9.9. Replace 19](#_Toc80451605)

[Definition 19](#_Toc80451606)

[Example 19](#_Toc80451607)

[Parameters 19](#_Toc80451608)

[Logic 19](#_Toc80451609)

[Output 19](#_Toc80451610)

[9.10. Dictionary 20](#_Toc80451611)

[Definition 20](#_Toc80451612)

[Example 20](#_Toc80451613)

[Logic 20](#_Toc80451614)

[Output 20](#_Toc80451615)

[9.11. Common Config 20](#_Toc80451616)

[Definition 20](#_Toc80451617)

[Example 20](#_Toc80451618)

[Logic 20](#_Toc80451619)

[Output 20](#_Toc80451620)

[9.12. Fetch 20](#_Toc80451621)

[Definition 21](#_Toc80451622)

[Example 21](#_Toc80451623)

[Parameters 21](#_Toc80451624)

[Logic 21](#_Toc80451625)

[Output 21](#_Toc80451626)

[9.13. Action 21](#_Toc80451627)

[Definition 21](#_Toc80451628)

[Example 21](#_Toc80451629)

[Parameters 21](#_Toc80451630)

[Logic 22](#_Toc80451631)

[Output 22](#_Toc80451632)

[9.14. Expression 22](#_Toc80451633)

[Definition 22](#_Toc80451634)

[Example 22](#_Toc80451635)

[Output 22](#_Toc80451636)

[10. Preprocessors 23](#_Toc80451637)

[10.1. Distinct 23](#_Toc80451638)

[Definition 23](#_Toc80451639)

[Example 23](#_Toc80451640)

[Parameters 23](#_Toc80451641)

[Logic 23](#_Toc80451642)

[10.2. Order 23](#_Toc80451643)

[Definition 23](#_Toc80451644)

[Example 23](#_Toc80451645)

[Parameters 23](#_Toc80451646)

[Logic 24](#_Toc80451647)

[10.3. Cache 24](#_Toc80451648)

[Definition 24](#_Toc80451649)

[Example 24](#_Toc80451650)

[Parameters 24](#_Toc80451651)

[Logic 24](#_Toc80451652)

[10.4. Store 24](#_Toc80451653)

[Definition 24](#_Toc80451654)

[Example 24](#_Toc80451655)

[Parameters 25](#_Toc80451656)

[Logic 25](#_Toc80451657)

[10.5. Read 25](#_Toc80451658)

[Definition 25](#_Toc80451659)

[Example 25](#_Toc80451660)

[Parameters 25](#_Toc80451661)

[Logic 25](#_Toc80451662)

[11. Post-processors 26](#_Toc80451663)

[11.1. Memory 26](#_Toc80451664)

[Store 26](#_Toc80451665)

[Read 26](#_Toc80451666)

[11.2. String 27](#_Toc80451667)

[Discard 27](#_Toc80451668)

[Substring 27](#_Toc80451669)

[Trim 28](#_Toc80451670)

[Pad 28](#_Toc80451671)

[Length 29](#_Toc80451672)

[Upper 29](#_Toc80451673)

[Lower 29](#_Toc80451674)

[Sentence 30](#_Toc80451675)

[Title 30](#_Toc80451676)

[Truncate 30](#_Toc80451677)

[Index 31](#_Toc80451678)

[Extract 31](#_Toc80451679)

[Replace 32](#_Toc80451680)

[Split 32](#_Toc80451681)

[11.3. Format 33](#_Toc80451682)

[Date 33](#_Toc80451683)

[Number 33](#_Toc80451684)

[11.4. Clear 34](#_Toc80451685)

[Definition 34](#_Toc80451686)

[Example 34](#_Toc80451687)

[11.5. Collection 34](#_Toc80451688)

[First 34](#_Toc80451689)

[Nth 34](#_Toc80451690)

[Last 35](#_Toc80451691)

[Count 35](#_Toc80451692)

[Top 35](#_Toc80451693)

[Distinct 36](#_Toc80451694)

[Order 36](#_Toc80451695)

[Where 37](#_Toc80451696)

[Filter 37](#_Toc80451697)

[11.6. Aggregate 37](#_Toc80451698)

[Join 37](#_Toc80451699)

[Minimum 38](#_Toc80451700)

[Maximum 38](#_Toc80451701)

[Average 38](#_Toc80451702)

[Sum 38](#_Toc80451703)

# Introduction

This solution tries to solve the issue of the rigidity of text in CRM.

For example, notifications are limited to including only columns at a single level or double level down. This can be limiting in cases where deep references, relationship parsing, or non-related references are required. In addition, there is no option to manipulate text retrieved from the fields.

For advanced scenarios, a custom solution is required; hence, creating the CRM Parser.

# Terms and Structures

The following terms are used throughout the manual.

|  |  |
| --- | --- |
| Term or form | Description |
| <> | Anything between < and > should be manually replaced with what it describes when used. E.g. Retrieve(“<entity-name>”, “<id>”), when used could be Retrieve(“account”, “000-0001”). |
| Row/record | A CRM record or row in a table or entity. |
| Context | The list of records the parser is currently using to execute constructs – a reference point. |
| Global state | An object that includes the context, CRM service, cache, and local memory for the parser |
| Processor | A preprocessor, construct, or post-processor. |
| Construct | The main structure or placeholder that is placed in the to-be-parsed text. Takes the form {\_`<description>`\_<key>(<parameters>)|%<preprocessors>%<block>@<post-processors>@|<key>}. |
| Block | What is going to be used to run the construct logic. E.g., the field name to retrieve, or the name of the action to execute. |
| Preprocessor | Logic applied before the main construct logic is executed using the block. Wrapped in %. |
| Post-processor | Logic applied after the main construct logic is executed using the block. Wrapped in @. |
| Parameters | A list of inputs to a construct or processor. Parameters listed are all required, in order, unless otherwise mentioned. |
| Replacement map | A simplified JSON of a capture group name and a replacement text.  The JSON should only contain a key and value; nested objects are not supported. |
| Regex groups | A feature where if the first parameter is defined as below, the regex will be executed and the result processed. If the regex contains captures, the processor will be applied to each and the result replacing the capture.  Form: $`<regex>`  E.g., given the text ‘this is a test’, if the upper post-processor is applied using the regex group $`\b([a-z])`, the processor will be applied to the first letter of each word only. Output: This Is A Test. |

# Installation

The YS Common solution is required for the configuration entities. It can be skipped if the dictionary or configuration constructs are not needed.

Install either Yagasoft.Libraries.Common (DLL installed) or Yagasoft.Libraries.Common.File (the parser class itself is embedded in the project itself) NuGet package, and then reference the CrmParser class.

<https://github.com/yagasoft/Dynamics365-YsCommonSolution>

<https://www.nuget.org/packages/Yagasoft.Libraries.Common/>

<https://www.nuget.org/packages/Yagasoft.Libraries.Common.File/>

<https://github.com/yagasoft/Dynamics365-CrmTextParser>

# Algorithm

1. Tokenisation
   1. The first step is for the parser to try to extract all constructs from the inside out (nested first).
   2. Constructs are replaced with a token.
   3. This is done to simplify the Regex and to allow for defining a context that is applied to all nested constructs if necessary.
2. Parser loop
   1. If no constructs are found in the text
      1. Tries to detokenize.
   2. Else
      1. Matches the construct key with its logic (decoration).
      2. Passes the parameters, preprocessors, block, and postprocessors.
      3. The global state is passed as well.
3. Construct execution
   1. Depends mainly on the construct.
   2. In general, the basic logic is as follows:
      1. The context is retrieved from the global state.
      2. Preprocessors are executed, in order, taking into account the block content.
      3. The result is a single text to replace the block in the construct.
      4. Construct logic is executed using the block as input.
      5. The result is a single text per record in the context.
      6. Post-processors are executed, in order, on each entry resulting from the construct logic.
      7. By default, all entries are merged into a single output string, unless an ‘aggregate’ post-processor is used.

# Reserved

The following patterns are reserved:

|  |  |
| --- | --- |
| Patterns | Description |
| ` | A tick. Used to escape text that contains reserved characters. |
| {<key>|<block>|<key>} | This is the basic form of a construct. The keys on the sides are a single-character marker, used for defining which construct to use.  This form is very unique and shouldn’t occur naturally in any text. |
| (<parameters>) | Parenthesis. Should be escaped. Used for parameters inside a construct. |
| % | Should be escaped. Used for defining preprocessors. |
| @ | Should be escaped. Used for defining post-processors. |
| | | A pipe. Should be escaped. Causes issues if not escaped. |

# Relationship Traversal

Some constructs support traversing relationships. A relation could be a lookup or grid.

A lookup can be referenced by using the dot operator; e.g., this.ownerid.managerid.

A grid can be referenced by using the bang operator; e.g. this.ownerid!ys\_servicerequests\_OwnerId. Use the schema name of the relation to traverse it. You can still traverse a lookup after the relation.

# Expressions

The parser supports evaluating expressions. Below are the rules in order of precedence. The higher entries in the table are evaluated first unless wrapped in parenthesis.

Expressions can be used in a block anywhere, except between ticks (`).

|  |  |
| --- | --- |
| Pattern | Description |
| \*,/ | Multiple and divide. |
| +,- | Add and subtract.  Can be a number or date.  If date, it must be in the following form: <date>+<value><unit>  The value is a number.  The unit is one of the following:   |  |  | | --- | --- | | Unit | Description | | m | Minute | | h | Hour | | d | Day | | M | Month | | y | Year | |
| <,>,<=,>= | Greater and less than. |
| ==,!= | Equality. |
| && | And. |
| || | Or. |
| ?? | If the left side is empty, take the right side. |
| ?: | Ternary conditional: <predicate>?<true-clause>:<false-clause>.  E.g., true?1:2, outputs 1. |

# Caching

By default, all calls to CRM’s web service are cached, except in the following cases:

# Constructs

This section lists all the constructs that can be used in the text.

## Complete Form

The complete form of a construct with all its optional parts:

{\_`<description>`\_<key>(<parameters>)|%<preprocessor>(<parameters>)%<preproc2>%<preproc3…>%<block>@<postprocessor>(<parameters>)@<postproc2>@<postproc3…>@|<key>}

## Expression

Does nothing except place its block in its place in the text. It is just like any other construct, except that it does nothing with the block.

Useful when you only want the effect of pre and post processors, or just evaluate an expression (check respective section).

If the block is not intended to be treated as an expression, escape it by wrapping in ticks `.

### Definition

Key: e

### Example

{e|`This is a repeatable text.`@store(repeatableText)@|e}

### Output

The block after expression evaluation and applying the pre and post processors.

## Column

The most important of all constructs.

Returns a column value. Supports traversing using dot and bang operators.

### Definition

Key: c

### Example

{c(name)|ownerid|c}

In this example, we are getting the record of the owner and then his name.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Transform | √ | Transform the result using one of the following switches:   |  |  | | --- | --- | | Switch | Description | | raw | Output the raw value based on its type as follows:   1. Date: sortable ISO 2. Option-set: numeric value 3. Lookup: <logical-name>:<ID> | | name | Output the default formatted representation provided by CRM’s web service. Falls back to raw if missing. | | log | Output the logical name of the lookup. Defaults to empty if wrong type. | | id | Output the ID of the lookup. Defaults to empty if wrong type. | | url | Output a hyperlink of the lookup. Defaults to empty if wrong type. |   The default behaviour is to try to represent result in as user-friendly pattern as possible. |

### Logic

1. Traverse the block.
2. If one of the nodes is empty
   1. Output nothing.
3. Apply the parameter that is given in the table above.

### Output

The column value after the traversal.

## Preload

Ensures that the specified list of columns is loaded from CRM in one go for the current record in the context. It improves performance when multiple columns are accessed in the text.

### Definition

Key: <

### Example

{<|ownerid,createdon|<}

### Logic

1. Cache the column values in the current record in the context by retrieving the list of columns given from CRM.

### Output

Nothing.

## Reference

The most interesting of all constructs.

Either reference a query, action, or a relation. The reference is then set as the context for all nested constructs. The block is executed for each record in the context.

### Definition

Key: .

### Example

{.(this.owner)|Owner’s name: {c|fullname|c}|.}

In this example, we are setting the context of execution to the owner record set in the current record. Effectively, we traversed into the owner lookup. The full name will be retrieved from the owner record. The output could be Owner’s name: Test User.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Scope |  | Can be a query, action, or relation. The first two are actual constructs defined elsewhere in the parsed text before the reference. The latter will be defined below.  Use this to reference the current context instead of a query or action.  A relation could be a lookup or grid. A lookup can be referenced by using the dot operator.  A grid can be referenced by using the bang operator. |
| Context name | √ | Stores the resulting context in memory for later reference. |
| Local variable name | √ | Store each record in the context in this variable in memory to be referenced in the nested constructs. Similar to a foreach in C#. |
| Global | √ | The text global could be added as a parameter to indicate that the context should not be reset after execution the reference. |

### Logic

1. Parse the parameters.
2. Extract the scope traversal from the parameters.
3. If not ‘this’
   1. Try to find the stored context by name defined in the scope parameter
   2. Set as initial context
4. Traverse the context (this or otherwise) using the remaining nodes in the scope.
5. Set the resulting records as the new context.
6. If column name is given
   1. Store the whole context by that name.
7. If local variable is given
   1. Store the current record by that name.
8. Execute the block for each record in the new context.
9. If ‘global’
   1. Reset the context to how it was before this construct.

### Output

The result of executing the block for each record in the context.

## Fetch

Stores the FetchXML given in memory for later execution when referenced (reference-construct).

### Definition

Key: f

### Example

{f(anyAccount)|`<fetch no-lock='true' top='1'>  
 <entity name='account' >  
 <attribute name='name' />  
 </entity>  
</fetch>`|f}

In this example, we are defining a query that will retrieve any account from CRM upon execution.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Store name |  | The name under which the query will be stored. |

### Logic

1. Store the query in an executable wrapper in the state’s memory under the given name.

### Output

Nothing.

## Action

Stores a reference to the given action in memory for later execution when referenced (reference-construct).

### Definition

Key: a

### Example

{a(getLiveExamResult,`{examId:'{c(id)|ys\_examid|c}'}`)|ys\_getexamresultfromwebservice|a}

In this example, we are defining an action reference that retrieves an exam result from an external web service upon execution, given the retrieved exam ID in the input parameters defined.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Store name |  | The name under which the query will be stored. |
| Input params | √ | A simplified JSON of input parameters. |
| Global | √ | Accepts: either specify the value global or omit the parameter.  Default: false.  Defines the action as a global action, which does not pass the context as a Target. |

### Logic

1. Store the action in an executable wrapper in the state’s memory under the given name.
2. Include in the definition
   1. The input params.
   2. The global flag.

### Output

Nothing.

## Info

Returns some info about the current record in the context.

### Definition

Key: i

### Example

{i|name|i}

In this example, we are getting the record of the owner and then his name.

### Logic

1. Treat the current record in the context as a lookup.
2. Apply the same logic as in the Column construct.

### Output

The info value of the current record.

## Template

Define any text, including constructs, given a name that can be referenced later. Think of it like a ‘constant’ defined in code.

### Definition

Key: t

### Example

{t(salutation-template)|[salutation] {c|ys\_customername|c}|t}

In this example, we marked the ‘salutation’ to be replaced later. This marking is nothing to do with the parser logic. It’s a user-defined way to mark text for replacement; do your thing here. The ‘customer name’ is the name of the field to retrieve from CRM; it’s a construct defined below.

The template is stored under the name: ‘salutation-template’.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Name |  | The name to store the template. Can be used later to reference the template. |

### Logic

1. Detokenise the block completely.
2. Store the result in the memory under the name given.

### Output

Nothing.

## Placeholder

Reference a template by name to insert into this position, optionally, replace certain text in the template.

### Definition

Key: p

### Example

{p(`[salutation]`,`Dear`)|salutation-template|p}

In this example, we are retrieving the template called ‘salutation-template’. The text ‘[salutation]’ in the template will be replace with the word ‘Dear’.

### Parameters

Inputs, defined in pairs (e.g., `text1`,`replacement1`,`text2`,`replacement2`):

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Text | √ | Text to find in the template. |
| Replacement | √ | Text to replace it with. |

### Logic

1. Retrieve the template by the name defined in the block.
2. Parse the parameters.
3. If replacement pairs are defined
   1. Take each pair.
   2. Replace first item in the pair with the second item.
4. Place the result in place of the construct.

### Output

Template block, with replaced text.

## Discard

Executes the block, but outputs nothing in the end. Can be used to do some operations in memory in preparation for later actions.

### Definition

Key: \_

### Example

{\_|{c|ys\_name@store(accountName)@|c}|\_}

In this example, retrieve the name column, store it as accountName in memory, and then output nothing.

### Logic

1. Parse the block and execute.
2. Discard the result.

### Output

Nothing.

## Replace

Returns the block with the matching pattern replaced. It supports capture groups (explained in the logic).

### Definition

Key: r

### Example

{r($`\s\d\s`,`\_`)|Test 2 programs|r}

In this example, we are replacing all spaces followed by a digit and a space by an underscore. Output: Test\_programs.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex group |  | The pattern to use to match the text to replace. |
| Replacement |  | Either a text or a replacement map.  If a replacement map (check terms table) is given, replace the capture group by name with the value given in the map; otherwise, the text will be inserted in place of the match. |

### Logic

1. If replacement map given
   1. Search for capture groups.
   2. Match the capture group name or index with one in the map.
   3. If found
      1. Replace with corresponding value
2. Else
   1. Replace all the matches (regardless of captures) with the given text.

### Output

The text with the matches replaced.

## Dictionary

Returns the text value stored in the ys\_keyvalue table in CRM.

### Definition

Key: v

### Example

{v|informationurl|v}

In this example, we are retrieving a value from the dictionary table with key: informationurl.

### Logic

1. Retrieve from the table ys\_keyvalue a record where key is the block content.

### Output

The value retrieved.

## Common Config

Returns the value stored in the Common Configuration table in CRM.

### Definition

Key: g

### Example

{g|ys\_informationurl|g}

In this example, we are retrieving the value of the column ys\_informationurl from the Common Config table.

### Logic

1. Retrieve from the table ldv\_genericconfiguration the value of the column specified in the block content.
2. Output the value in the most user-friendly format possible.

### Output

The value retrieved.

# Preprocessors

This section lists all the preprocessors that can be applied on blocks.

## Store

Stores the value so far in memory. If preprocessors are chained, the block value processed by preceding processors up to this point is stored.

### Definition

Key: store

### Example

{c|%store(x)%ys\_name|c}

In this example, we are storing the value ys\_name in memory under the name x.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Store name |  | The name under which the value is stored in memory. |

### Logic

1. Stores the value.

## Read

Read the value stored from memory. Overwrites the value of preceding processors.

### Definition

Key: read

### Example

{c|%read(y)%ys\_name|c}

In this example, we are reading the value stored in y. If y had value ownerid then the column-construct will retrieve the owner of the record instead of the name.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Store name |  | The name under which the value is stored in memory. |

### Logic

1. Read the value from memory.
2. If the read value is a collection (from a post-processor store for example)
   1. Concatenate the elements of the collection into a single value.
3. Overwrite preceding processors value.

## Cache

Sets the cache mode of the construct. Not inherited by nested constructs.

### Definition

Key: cache

### Example

{.(this!departments)|%cache(true,global)%{c|ys\_name|c}|.}

In this example, we are retrieving related departments, and then caching the result of this query in CRM’s memory – not the parser.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Cache |  | Flag to enable or disable caching.  Default: true. |
| Global | √ | If the text global is passed here, the caching action happens on the executing process memory – not the parser state memory; so, it persists after the parser has finished. |

### Logic

1. Set the ‘cache’ parameter on the construct object.

## Distinct

Used with the reference-construct to retrieve only unique records.

### Definition

Key: distinct

### Example

{.(this!accounts)|%distinct(accounttype,industry)%{c|ys\_taxpercent|c}|.}

In this example, we are retrieving related accounts and then making them distinct over the account type and industry (combined), then selecting the tax percentage value.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Columns |  | The list of columns to use, comma-separated. |

### Logic

1. Set the distinct parameter on the construct object to be used after running the query.

## Order

Used with the reference-construct to sort records.

### Definition

Key: order

### Example

{.(this!accounts)|%order(accounttype,!industry)%{c|ys\_taxpercent|c}|.}

In this example, we are retrieving related accounts, ordering them by account type first, and then by industry in descending order (notice the bang sign ! before industry), then selecting the tax percentage value.

### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Columns |  | The list of columns to use, comma-separated. Optionally, prefix a bang ! for descending order. |

### Logic

1. Set the order parameter on the construct object to be used after running the query.

# Post-processors

This section lists all the post-processors that can be applied on the construct result.

## Memory

### Store

Stores the value so far in memory. If post-processors are chained, the result processed by preceding processors up to this point is stored.

#### Definition

Key: store

#### Example

{c|ys\_name@store(x)@|c}

In this example, we are storing the value stored in ys\_name in CRM in memory under the name x.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Store name |  | The name under which the value is stored in memory. |

#### Logic

1. Stores the value.

### Read

Read the value stored from memory. Overwrites the value of preceding processors.

Useful in combination with the store post-processor. It can be used to store a construct result, do some post-processing, store again under a different name, read the previously stored result, do more processing, store again, and so on. Effectively, using the same value to transform it into different stored values that can be referenced later.

#### Definition

Key: read

#### Example

{c|ys\_name@store(y)@trim( )@store(trimmed)@read(y)@|c} cleaned = {e|%read(trimmed)%|e}

In this example, …

1. Store the value retrieved by the column construct in y
   1. E.g., John Doe .
2. Remove spaces around the name.
3. Store the trimmed text in trimmed.
4. Restore the original text in the pipeline from memory (y).
5. Output the read value.
6. Append cleaned = to the output.
7. Parse the expression-construct.
8. Read trimmed from memory.
9. Append the read value to the output.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Store name |  | The name under which the value is stored in memory. |

#### Logic

1. Read the value from memory.
2. Overwrite preceding processors value.

## Discard

Ignore the result of the construct pipeline so far.

### Definition

Key: discard

### Example

{c|ys\_name@discard@|c}

In this example, we output nothing.

### Logic

1. Ignore the value returned by the construct after this point.

## String

### Length

Outputs the length of the text. Supports regex groups (check terms section), but only 1 capture is recommended.

#### Definition

Key: length

#### Example

{c|ys\_count@length@|c}

In this example, the output is the length of the name.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |

### Index

Returns the indices of the matches in the output. Supports regex groups (check terms section).

If a ‘capture group’ is detected in the pattern, all capture group values indexes are returned as comma-separated values (example below).

#### Definition

Key: index

#### Example

{c|ys\_description@index($`\d`,last)@join(`|`)@|c}

In this example, given description as test 1 and 234, test 5 and 678, the output will be 11,12,13|27,28,29.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex group |  | The pattern to find the text whose index is required. |
| Last | √ | If last is given as a parameter, retrieve the index of the last match only, and last capture as well if multiple. |
| Single | √ | If single is given as a parameter, only output one capture per match in the text (discards 2, 3, 6, and 7 indices in the example above). |

### Substring

Takes part of the string specified by the index and length. Supports regex groups (check terms section).

#### Definition

Key: sub

#### Example

{c|ys\_name@sub(0,3)@|c}...

In this example, output the first 3 characters of the name, followed by 3 dots.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |
| Start |  | The starting index. Zero-based. |
| Length | √ | The length of the string starting at the start index. |

### Trim

Removes the specified characters from the edges of the output. Supports regex groups (check terms section).

#### Definition

Key: trim

#### Example

{c|ys\_name@trim( )@|c}

In this example, spaces are removed from around the name.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |
| Characters |  | Characters to remove. |
| Start | √ | If start is given as a parameter, only trim the output start.  Default: start and end, both enabled |
| End | √ | If end is given as a parameter, only trim the output end. |

### Pad

Fills the text with the given character up to the specified length. Supports regex groups (check terms section).

#### Definition

Key: pad

#### Example

{c|ys\_count@pad(0,4)@|c}

In this example, the output is filled with zeros on the left until it fits the length specified. If it exceeds the length, nothing happens.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |
| Character |  | Character to fill with. |
| Length |  | Length of the output. |
| Right | √ | If right is given as a parameter, pad the right side of the output. |

### Truncate

Shortens the output to the specified length, and then append the given replacement. Supports regex groups (check terms section).

#### Definition

Key: truncate

#### Example

{c|ys\_description@truncate(10,`...`)@|c}

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |
| Length |  | Length of the new output. |
| Replacement | √ | Text to append to the new output (e.g., ‘…’). |

### Upper

Converts the text to upper case. Supports regex groups (check terms section).

#### Definition

Key: upper

#### Example

{c|ys\_description@upper@|c}

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |

### Lower

Converts the text to lower case. Supports regex groups (check terms section).

#### Definition

Key: lower

#### Example

{c|ys\_description@lower@|c}

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |

### Sentence

Converts the text to sentence case. Supports regex groups (check terms section).

#### Definition

Key: sentence

#### Example

{c|ys\_description@sentence@|c}

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |

### Title

Converts the text to title case. Supports regex groups (check terms section).

#### Definition

Key: title

#### Example

{c|ys\_description@title@|c}

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |

### Extract

Returns the matches in the output. Supports regex groups (check terms section).

If a ‘capture group’ is detected in the pattern, all capture group values are returned as comma-separated values (example below).

#### Definition

Key: extract

#### Example

{c|ys\_description@extract($`\d`,last)@join(`|`)@|c}

In this example, given description as test 1 and 234, test 5 and 678, the output will be 2,3,4|6,7,8.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex group |  | The pattern to find the text whose match value is required. |
| Last | √ | If last is given as a parameter, retrieve the last match only. |
| Single | √ | If single is given as a parameter, only output one capture per match in the text (discards 2, 3, 6, and 7 in the example above). |

### Replace

Replaces text matching the pattern given. Supports regex groups (check terms section).

#### Definition

Key: replace

#### Example

{c|ys\_description@replace($`\d`,\_)@|c}

In this example, given description as test 1 and 234, test 5 and 678, the output will be test \_ and \_\_\_, test \_ and \_\_\_.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex group |  | The pattern to use to match the text to replace. |
| Replacement |  | Either a text or a replacement map.  If a replacement map (check terms table) is given, replace the capture group by name with the value given in the map; otherwise, the text will be inserted in place of the match. |

### Split

Finds the matches in the text, and then splits them by the given text. Supports regex groups (check terms section).

All splits are blown and returned as separate items. E.g., if the output from the previous processor returned 2 items and the split caused each item to expand by 3, then the output of this processor is 6 items.

#### Definition

Key: split

#### Example

{c|ys\_description@split($`(?:(.\*?)(?: and )?)\*`,`,`)@join(`|`)@|c}

In this example, given description as test,1 and test,2, the output will be test|1|test|2.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex group |  | The pattern to find the text whose match value is required. |
| Last | √ | If last is given as a parameter, retrieve the last match only, and last capture as well if multiple. |
| Single | √ | If single is given as a parameter, only output one capture per match in the text (discards 2, 3, 6, and 7 in the example above). |

## Format

### Date

Formats output as a date. Supports regex groups (check terms section).

If regex groups are used, the captures are the ones to be formatted, with the rest of the text intact.

#### Definition

Key: date

#### Example

{c|ys\_description@date(`yyyy`)@|c}

In this example, given description as 2021-01-11, output: 2021.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |
| Output format |  | The date format to output the date. |
| Kind | √ | Either utc or local to define the kind of input date. |
| Input format | √ | The date format to process the input date. Useful in assisting the processor in processing the input text incorrectly. |

### Number

Formats output as a number. Supports regex groups (check terms section).

If regex groups are used, the captures are the ones to be formatted, with the rest of the text intact.

#### Definition

Key: number

#### Example

{c|ys\_description@number(`Cost: (\d\*)`,`$#.000`)@|c}

In this example, given description as Cost: 123.1, output: Cost: $123.100.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |
| Output format |  | The date format to output the date. |

## Clear

Removes all empty text from the output.

### Definition

Key: clear

### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@clear@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, empty tax, and 20; the output will be 15,20.

## Collection

### Count

Returns the count of items in the output.

#### Definition

Key: count

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@count@|.}

In this example, if the system has 3 accounts, the output will be 3.

### First

Returns only the first item in the output.

#### Definition

Key: first

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@first@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 15.

### Nth

Returns only the last item in the output.

#### Definition

Key: nth

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@nth(2)@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 10.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Index |  | The index of the item to return. Not zero-based. |

### Last

Returns only the last item in the output.

#### Definition

Key: last

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@last@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 20.

### Top

Returns the first x items in the output.

#### Definition

Key: top

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@top(2)@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 15,10.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Count |  | The number of items to return. |

### Distinct

Return unique values from the output. Supports regex groups (check terms section).

#### Definition

Key: distinct

#### Example

{.(this!accounts)|{c|ys\_name|c}@distinct($`(?:.\*?)(\d\*)`)@|.}

In this example, if the system has 3 accounts stored with the categories: Company1-Site1, Company2-Site1, and Company1-Site2; the output will be Company1-Site1,Company2-Site1.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |

### Order

Return unique values from the output. Supports regex groups (check terms section).

#### Definition

Key: order

#### Example

{.(this!accounts)|{c|ys\_name|c}@order($`(?:.\*?)(\d\*)`)@|.}

In this example, if the system has 3 accounts stored with the categories: Company1-Site1, Company2-Site1, and Company1-Site2; the output will be Company1-Site1,Company1-Site2,Company2-Site1.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups | √ | Refer to terms section. |
| Descending | √ | If true is given, the order will be reversed. |

### Where

Return values that pass the test from the output. Supports regex groups (check terms section).

#### Definition

Key: where

#### Example

{.(this!accounts)|{c|ys\_name|c}@where($`Site\d+`)@|.}

In this example, if the system has 3 accounts stored with the categories: Company1-Site1, Company2-Site1, and Company1-Site2; the output will be Company1-Site1,Company2-Site1.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups |  | Refer to terms section. |

### Filter

Remove values that pass the test from the output. Supports regex groups (check terms section).

#### Definition

Key: filter

#### Example

{.(this!accounts)|{c|ys\_name|c}@filter($`Site\d+`)@|.}

In this example, if the system has 3 accounts stored with the categories: Company1-Site1, Company2-Site1, and Company1; the output will be Company1.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Regex groups |  | Refer to terms section. |

## Aggregate

### Join

Returns only the last item in the output.

#### Definition

Key: join

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@join(`|`)@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 15|10|20.

#### Parameters

Inputs:

|  |  |  |
| --- | --- | --- |
| Parameter | Optional | Description |
| Join pattern |  | The text to insert between the items when joining. |

### Minimum

Returns the minimum value in the output.

#### Definition

Key: min

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@min@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 10.

### Maximum

Returns the maximum value in the output.

#### Definition

Key: max

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@max@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 20.

### Average

Returns the average value of all items in the output.

#### Definition

Key: avg

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@avg@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 15.

### Sum

Returns the sum of all items in the output.

#### Definition

Key: sum

#### Example

{.(this!accounts)|{c|ys\_taxpercent|c}@sum@|.}

In this example, if the system has 3 accounts stored with the tax percentages: 15, 10, and 20; the output will be 45.