CRM PARSER MANUAL

Written by Ahmed Elsawalhy (Yagasoft.com)

V3.1

1 CONTENTS

2 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.

3 CODE AND CONTRIBUTION

Please do not hesitate to share your opinion and ideas on GitHub repo here.

Code can be found here.

4 INSTALLATION

The YS Common solution is required for the configuration entities. It can be skipped if the `configuration` expression is not needed.

Install either `Yagasoft.Libraries.Common` (DLL installed) or `Yagasoft.Libraries.Common.File` (the parser class file will be added to the project automatically) 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

5 TERMS

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 initial record given to the parser at the start of execution.
Global state	An object that includes the context, CRM service, cache, and local memory for the parser.
Expression	The main structure or placeholder that is placed in the to-be-parsed text. Placed between curly braces (`{` and `}`).
Block	Expressions grouped inside curly braces.
Parameters	A list of inputs to a function. Parameters listed are all required, in order, unless otherwise mentioned.

6 STRUCTURES

6.1 INPUT

The text provided to the `Parse` function. It contains text and expressions (defined later in this guide) that will be parsed, evaluated, and replaced with a value based on their logic.

6.2 EXPRESSION

Code wrapped in curly braces.

6.3 RESERVED

The following patterns are reserved, and must be escaped:

Patterns	Description
`	A tick. Escapes text that contains reserved characters.
****	Escapes the character that comes after.
{ **}**	Defines expression blocks.
(<parameters>)	Wrapped in parenthesis, similar to methods in code.
,	Separates parameters.
$	Defines function.
@	Defines objects.
! **+** **-** ***** **/** **?** **:** **<** **>** **||** **&&** **=**	Operators.

Outside a block, only curly braces are reserved.

7 ALGORITHM

7.1 INTERPRETATION

The parser starts by interpreting the input text given.

Text is converted into a list of “tokens”.

Tokens are then processed into an expression tree to determine execution order.

7.2 EVALUATION

The tree is traversed in-order.

Each node is evaluated (along with its children) and the result is passed on to the next node on the same level.

Some nodes do their work without output; they simply pass on their input value to the next node.

8 HOW TO USE

8.1 TRIGGER

To start parsing, supply a text value to the following method:

```
CrmParser.Interpreter
  .Parse(<text>, <custom-expression-types>)
  .Evaluate(<crm-service>, <entity-or-reference>, <context-object>);
```

The following table explains the function of each parameter:

Parameter	Description
input	Any text value that needs to be parsed for constructs.
expression types	The type (class) that contains custom constructs in your code. For example, if you create an 'indexing' construct (exists in the Auto-Numbering solution), you can define its class as follows: `public class CustomConstructs { [Construct("j", "index")] public class IndexConstruct : DefaultContextConstruct { <logic> } }` You should pass the `CustomConstructs` class to the parser.
context	Can be either an entity or entity reference.
service	The CRM org service. It should only be provided when expressions that require CRM access are used. For example, if `$retrieve` is used, you must pass a service to the parser.
context object	An object that might be required by a custom expression. If you create a custom expression, you can access this object by calling `State.ContextObject` from within the expression class. All the default expressions do not require any context object.

8.2 QUICK SAMPLE

```
The budget for this project's accounts is {@this#ys_accounts_ys_project_accountid
$distinct(.accountid)$map(.ys_budget)$sum$num(`$#.#`)}.
```

The code between the curly braces will be replaced with the sum of the budget amount, in its place in the text.

8.3 TESTING TOOL

All of the functionality of this parser can be tested using an XrmToolBox plugin:
https://www.xrmtoolbox.com/plugins/D365-CrmTextParser-Tester-Plugin

9 EXPRESSIONS

Think of code between curly braces as a pipeline: at one end, nothing is given; at the other end, a string output is placed at that position.

Each node in the pipeline performs some operation, or points to a certain value, and then outputs another value for the next node to use.

To access values earlier in the pipeline at a later stage, store them in memory (`~<store-name>`) and then retrieve them later (`@<store-name>`).

In the tables below, an "object" output indicates that the output depends on the input and operation of the respective expression.

9.1 SCOPES

Form	Description	Output	Example
{ }	Triggers code execution for the code within.	String.	`{@this.fullname}`
( )	Groups code to be executed as a whole, and given higher priority than surrounding code. Groups function parameters.	Object.	`$map(.fullname)` `(1+2)*3`
[ ]	Forms a collection of objects.	Array.	`[@this.fullname,@this.statecode]`
,	Separates parameters and collection elements.	Object.	`$retrieve(`contact`,`1234…`)`

9.2 LITERALS

Form	Description	Output	Example
``	Escaped text.	String.	`` `This text will not be parsed.` ``

`/<regex>/<optional-flattening>`	Regex.
	When captures are used, the result is placed in a 3-leveled array.
	First level are the matches, second level are the captures, third level are the values within each capture.
	The flattening optional parameter triggers flattening of
		Array.	`` `/text1(.*?)text2/f2` ``
			`` `/match this text/` ``
			`` `/match a ([a-z]{3}) text/` ``

	the result if captures are used, the number of times specified; e.g., if 2 is given, flatten two levels.		
<number>	Integer or decimal.	Int or Double.	`5` `1.2`
null	Nothing.	Null.	`null`
<true|false>	True or false value.	Boolean.	`true` `false`
`<number><unit>`	A timespan. Described in the “operators” section in detail.	String.	`` `1d2h` ``

9.3 OBJECTS

Form	Description	Output	Example
@this	Loads the input entity/reference into this position in the pipeline.	Entity or entity reference.	`@this.fullname`
@value	Refers to the current value in the pipeline. Most useful as an operand for operators.	Object.	`2+@value`
@user	Refers to the CRM service authenticated user.	Entity.	`@user.lcid`
@<mem-store>	Loads a stored value from memory.	Object.	`@variable1`
.<property>	Tries to access the property in the current object loaded in the pipeline.	Object. Entities have the following additional properties: • `name`: primary name of the record • `logical`: logical name • `id`: GUID • `url`: URL of the record Entity reference properties: • `id` • `logical` • `name`	`@this.fullname` `@this.statecode.value`

		Option-set value properties: • `name` • `value`	

9.3.1 RELATIONSHIP

Entity and reference objects 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 sharp (`#`) operator; e.g., `@this.ownerid#ys_servicerequests_OwnerId`.

Use the *schema name* of the relation to traverse it.

Usually, you will need to loop (`$map`) over the result to get any meaningful result.

9.4 MEMORY

9.4.1 STORE

Stores the current pipeline value into memory.

Usage: `~<mem-store-name>`

9.5 OPERATORS

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 anywhere, except between ticks (`` ` ``).

Addition and subtraction have a special significance when used with a date as below.

9.5.1 PRECEDENCE

Use parentheses (`(` and `)`) to control/change precedence, just like a calculator.

From highest to lowest, with operators in the same row having the same precedence:

Patterns	Description
! **-**	Not Negative
***** **/**	Multiply Divide
+ **-**	Add Subtract
> **>=**	Greater than Greater than or equal

< **<=**	Less than Less than or equal
!= **==**	Not equal Equals
&&	Logical and
||	Logical or
??	Null-coalescing: if left side is null, take right side `<first-clause>??<second-clause>` E.g., `null??\`xyz\``, outputs 'xyz'.
? :	Conditional: if true, take first, else, take second `<predicate>?<true-clause>:<false-clause>` E.g., `false?1:2`, outputs '2'.

9.5.2 SHORT CIRCUIT

None of the operators short-circuit; all operands are fully evaluated before operations take place. It's planned for a future release.

9.5.3 UNITS

Usage: `<date>+<value1><unit1><value2><unit2>…`

The value and unit can be repeated as necessary to define different time granularities. The value is a number.

The unit can be one of the following:

Unit	Description
f	Millisecond
s	Second
m	Minute
h	Hour
d	Day
M	Month
y	Year

9.6 FUNCTIONS

Functions must be preceded with `$`. Some don't require parameters. Some functions repeat the expressions in its parameter definition.

Optional parameters are preceded with a `*`.

9.6.1 RANDOM GENERATOR

Generates a random string.

Form:
`$rand(<length>, <either>[`<char1>`, `char2>`]<or>`<u-l-n>`, <is-letter-start-?>, <number-letter-ratio>)`

Parameters:

1. Length (number): length of the output string.
2. Pool (array or “uln”): either an array of strings to choose from, or a combination of the letters u, l, and n.
    a. u: upper-case letters.
    b. l: lower-case letters.
    c. n: numbers.
3. * Is letter start? (boolean): flag to indicate the output must start with a letter.
4. * Number-letter ratio: a full-form (e.g., 40) percentage indicating the amount of number to letter characters in the output.

Output: string.

Examples:

```
$rand(2,[`a`,`j`,5,2],false,50)

$rand(5,`un`,true,20)
```

9.6.2 DEBUG

Placing this function anywhere switches on the “debug” mode.

From this point on, the parser outputs evaluation steps in detail.

Placing the function again, turns off this mode.

Form: `$debug`

9.6.3 MEMORY-ONLY OPERATIONS

Evaluates the expressions in the parameter definition, and then discards the output.

Form: `$mem(<expressions>)`

Example: `$mem(1+2)`

9.6.4 LOCALISE

Placing this function anywhere switches the language indicator in memory.

From this point on, option-set values (and any supported expressions) will be localised, if possible.

Form: `$loc(<lcid>)`

Example: `$loc(1025)`

9.6.5 TAG

Wraps object in a tag to be consumed by a concerned expression. (not implemented yet)

Form: `$tag(`<name>`)`

Example: `$tag(`tagged`)`

9.6.6 CRM

RETRIEVE

Retrieves a record from CRM.

Form: `$retrieve(`<logical-name>`, `<id>`, [`<optional-attr1>`, `<optional-attr2>`])`

Example: `$retrieve(`contact`, `123…`, [`fullname`, `statuscode`])`

RETRIEVE BY ATTRIBUTE

Retrieves a list of records whose attribute values match the ones given.

Form: `$retrbyattr(`<logical-name>`, `{\"<attr-name1>\":\"<attr-value1\", \"<attr-name2>\":\"<attr-value2\"}`, [`<optional-attr1>`, `<optional-attr2>`])`

Example: `$retrbyattr(`contact`, `{"statecode":"0", "industrycode":"775640"}`, [`fullname`])`

FETCH

Retrieves a list of records using FetchXML query.

Form: `$fetch(`<fetch-xml>`)`

Example: `$fetch(`

```
<fetch mapping='logical'>
   <entity name='account'>
      <attribute name='accountid'/>
      <attribute name='name'/>
   </entity>
</fetch>`)
```

ACTION

Retrieves a list of records whose attribute values match the ones given.

Form: `$action(`<action-name>`, `{"<param-name1>":"<param-value1", "<param-name2>":"<param-value2"}`, `<optional-target-logical-name>`, `<optional-target-id>`)`

Example: `$action(`ys_qualifylead`, `{"reason":"qualified"}`, `lead`, `123…`)`

CONFIGURATION

Retrieves configuration from CRM. (not implemented yet)

9.6.7 DATE

Form	Description	Input/output	Example
$now	Outputs the current local date.	Output: DateTime.	`$now`
$utcnow	Outputs the current UTC date.	Output: DateTime.	`$utcnow`
$utc	Converts the input date to UTC time zone.	Input: DateTime. Output: DateTime.	`$utc`
$local	Converts the input date to the local time zone.	Input: DateTime. Output: DateTime.	`$local`
$offset(`<time-zone>`)	Converts the input date to the given time zone by name. Time zone names can be extracted from Windows using this code: `TimeZoneInfo.GetSystemTimeZones()`	Input: DateTime. Output: DateTime.	`$offset(`Arabian Standard Time`)`
$exact(`<format>`)	Converts a text date to a DateTime object using this exact format.	Input: string. Output: DateTime.	`` `2023-04`$exact(`yyyy-MM`) ``
$date(`<format>`)	Converts a DateTime object to a string using this exact format. Formats can be found at Microsoft.	Input: DateTime. Output: string.	`$now$date(`F`)`

9.6.8 STRING

In most functions, a regex (unless otherwise specified) is used to extract strings from the input text, and then perform the respective function on those strings. The output in this case is an array.

Form	Description	Example
$length(`<regex>`)	Length of the input text. Regex is optional.	
$index(`<sub-string>`)	Index of a sub-string. Regex can be used in place of the "sub-string" parameter.	
$sub(<start-index>, <length>)	Part of a string by index.	
$trim(`<what-to-trim>`, <is-trim-start-only-?>, <is-trim-end-only-?>, `<regex>`)	Trim the characters from the string edges. Regex is optional.	
$pad(`<pad-string>`, <total-length>, <is-right-side-only-?>, `<regex>`)	Pad the string with the given string. Regex is optional.	

$trunc(<max-length>, `<fill-string>`, `<regex>`)	Truncate the string to the given length and replace with another filler text. Regex is optional.	
$upper(`<regex>`)	Convert to upper case. Regex is optional.	
$lower(`<regex>`)	Convert to lower case. Regex is optional.	
$sentence(`<regex>`)	Convert to sentence case. Regex is optional.	
$title(`<regex>`)	Convert to title case. Regex is optional.	
$extract(`<regex>`)	Extract a string from the text.	
$split(`<separator>`)	Split the text into sub-strings using a separator. Regex can be used in place of the "separator" parameter. The text is then split at the captures in the regex.	
$replace(`<replacement-text>`, `<regex>`)	Replace text with another.	
$enchtml(`<regex>`)	Encode text into HTML. Regex is optional.	
$dechtml(`<regex>`)	Decode text from HTML. Regex is optional.	
$num(`<format>`, `<regex>`)	Format a number using this exact format. Regex is optional.	

9.6.9 COLLECTION

Form	Description	Input/output	Example
$map(<code>)	Applies the expressions in the parameter definition to each element of the collection given.	Input: array. Output: array.	`[`x`,`yy`,`zzz`]` `$map($length)`
$for(<loop-variable>, <start>, <finish>, <scope>)	Not implemented yet.		
$get(<start-index-or-n>, <end-index-or-n>)	Gets the elements specified in the range given. Start index can be a number of the character `n`, which means "last element".	Input: array. Output: array.	`$get(1,`n`)` `$get(`n`)`

	End index is optional. If omitted, it is set to the last element by default.		
$break(<count>)	Not implemented yet.		

9.6.10 AGGREGATE

Form	Description	Input/output	Example
$count	Count the number of elements in an array.	Input: array.	`$count`
$distinct(<code>)	Keeps only unique values evaluated using the expression in the parameter definition.	Input: array.	`$distinct(.fullname)`
$order	Not implemented yet.	Input: array.	
$clear	Sets the current pipeline value to null.	Input: array.	`$clear`
$where(<code>)	Filters the elements using the expression in the parameter definition.	Input: array.	`$where(.statecode .value == 0)`
$filter(<code>)	Removes the elements using the expression in the parameter definition.	Input: array.	`$filter(.statecode .value == 1)`
$join(`<separator>`)	Joins the elements using the given separator.	Input: array.	`$join(`,`)`
$min	Gets the minimum value in the array.	Input: array.	`$min`
$max	Gets the maximum value in the array.	Input: array.	`$max`
$avg	Gets the average value of the array.	Input: array.	`$avg`
$sum	Sums the values in the array.	Input: array.	`$sum`
$flat(<levels>)	Flattens the array this number of times.	Input: array.	`$flat(2)`

10 CACHING

Planned for a future release.

11 EXTENDING THE PARSER

If you want to create your own expressions, follow the example below.

For example, if you create an 'indexing' function (exists in the Auto-Numbering solution), you can define its class as follows:

```
[Expression]
public class IndexExpression(ParserContext context) : FunctionExpression(context)
{

	protected override string FinalForm => "^[$]sequence$";

	protected override string RecognisePattern =>
"^[$](?:s(?:e(?:q(?:u(?:e(?:n(?:c(?:e)?)?)?)?)?)?)?)?$";

	protected override object FunctionEvaluate(object baseValue = null)
	{
		if (globalState.ContextObject is not AutoNumberingEngine contextObject)
		{
			return null;
		}

		Parameters = EvaluateParameters(baseValue);

		var keyName = GetParam<string>("Key", 0);

		var value = GetParam<object>("Value", 1);

		return contextObject.ProcessIndices($"{keyName}:{value}");
	}
}
```

You should pass the `IndexExpression` class to the parser as defined in the 'trigger' section of this guide.

Refer to the existing code in the repository on GitHub for more advanced scenarios.