PLACEHOLDERS MANUAL

Written by Ahmed el-Sawalhy

V2.1

1. CONTENTS

2. FEATURES

- Formatting the value
- String modifiers
- Drill-through to related records
- Insert record URLs
- Conditions to choose alternate values (?: in C#)
- Nesting of conditions
- Guards

3. BASICS

The simplest placeholder form is "`{[field-name]}`".

E.g., "`{name}`" that results in "`Ahmed`" if the target record was an account and the account's name was Ahmed.

3.1. NESTING

If placeholders are nested, the system parses the outermost first, and then moves deeper. This ensures that when conditions are used, only the satisfied branches are parsed (better performance).

3.2. DRILL-THROUGH

You can traverse N-1 relationships (lookups) to retrieve a value from a related entity. Using the `.`-operator, chain field names until you reach the desire value; e.g. `{primarycontactid.gendercode@1025}`.

The system throws an error if it cannot find one of the nodes in the path defined.

4. FORMATTING

Placeholders can be modified to manipulate the resulting string.

4.1. FORMAT MODIFIERS

To format the value, add `@` after the field name and then the format.

E.g. `{casetypecode@1025}`, which is replaced by the Arabic label of the Case Type field value.

- `@[lang-id]`: replaces an option-set value with the localised label
- `@[date-format]`: replaces a date value with its custom formatted equivalent (e.g. `@dd-MM-yyyy`)
- `@name`: replaces the lookup GUID with its name, which is retrieved from the primary name field of the entity
- `@[precision]`: replaces a decimal value with another at the specified precision (e.g. `@2`)

4.2. STRING MODIFIERS

In addition, you can further modify the resulting string using the following modifiers after a '`$`' character:

- `upper`: converts the string to uppercase
- `lower`: converts the string to lowercase
- `title`: converts the string to title case
- `length`: returns the length of the string
- `trim`
    - Removes whitespace (default) from the start and end of the string
    - Accepts the character to remove as a parameter
    - For example, if the name is John
        - `{name$trim:J}` returns `ohn`
- `trimstart`: same as the above, but removes characters from the start of the string
- `trimend`: same as the above, but removes characters from the end of the string
- `sub`
    - Returns only part of the string
    - Accepts the starting index and the length (optional) as parameters
    - For example, if the name is Ahmed
        - `{name$sub:3}` returns `ed`
        - `{name$sub:2,3}` returns `med`
- `padleft`
    - Adds enough characters to the left of the string so that the total length of the string matches the given number
    - Accepts the total string length and the character to pad with as parameters
    - For example, if the account number is 123
        - `{accountnumber$padleft:5,0}` returns `00123`
- `padright`: same as the above, but adds characters to the right
- `replace`

    - Replaces a string with another in the given string
    - Accepts the original string to replace and the replacement string as parameters
    - The parameters must be surrounded by double quotes
    - Any double quotes in the parameters must be escaped using `\`
    - For example, if the name of the account is Link Development
        - `{name$replace:$replace:“Link”,“\“Link\””}` returns “Link” Development
- `regex`
    - Finds all groups in the string and joins them together
    - For example, if the name of the account is Ahmed Salem Ahmed
        - `{name$regex(-):(.*)\sSalem\s(.*)}` returns `Ahmed-Ahmed`
        - Steps:
            - Modify the string using a regular expression
            - Join all found groups (between `(` and `)`) with the character –
            - Find any group of characters, until you meet a space, then Salem, then another space, and then another group of characters
            - Return the two groups of characters

4.3. CHAINING

Multiple string modifiers can be chained.

E.g. given the account name Link Development, `{name$sub:0,4$upper}{name$sub:5}` results in `LINK Development`.

In addition, string modifiers can be chained after a single format modifier.

5. RECORD URL

Use `{recordurl}` with any target record to build a link to the record itself.

E.g. `{recordurl}` for an account with ID 12345 will return `http://localhost/DevEnv/main.aspx?etc=1&id=%12345%7d&pagetype=entityrecord`.

6. CONDITIONS

If you require even more control over this placeholder, you can specify nested conditions. A basic condition: `{{[field-name]}??[filled-value]::[empty-value]}`.

E.g. `{{address}??FILLED::EMPTY}`, which translates to “if address is filled, print `FILLED`, else print `EMPTY`”.

A more advanced condition: `{{casetypecode}==1??NUM-{customerid.accountnumber}::EMAIL-{customerid.emailaddress1}}` might result in `NUM-123` or `EMAIL-test@test.com`.

Supported operators: `==`, `>=`, `<=`, and `!=`.

7. GUARDS

You could label or value-guard placeholders for processing by custom code.

7.1. LABELS

Add `{![label]![field-name_or_condition]}` to only process that placeholder when the label is given to the parsing method (`ParseAttributeVariables`). This is useful for when you have groups of placeholders tied to different entities in the same text.

For example, in emails, you can send an email related to a case, but want to embed some information from the sender, which does not exist in the case record as a lookup.

```
Dear {customerid@},

Kindy note that your case has been closed.

Regards,

{!from!fullname}
```

In your code, pass the case without a specific label, and then call the method again passing the sending user reference with the `from` label.

7.2. METADATA GUARDS

Another type of guard is metadata checks. Only one metadata check is currently supported: `{!meta.logicalname==[value]![field-name_or_condition]}`. This forces the system to parse the given placeholder only if the target record is of the specified type.

This is similar to the case of the labels above, but in this case, you rely on the target record type instead of an explicit label.

If the customer is going to modify the placeholders, it is recommended to use well-defined labels instead of metadata guards.

8. RELATIONSHIPS

Drilling through lookups is straightforward and has been covered above. However, when it comes to 1-N and N-N, it requires a bit more work.

8.1. BASICS

The idea is that related entities are retrieved, and then the specified field value in all those records is concatenated one after the next.

The basic format for a relationship placeholder is `{>[relation-schema-name].[field-name]}`.

E.g. if we have 3 emails with sequential subject values, then `{>Incident_Emails.subject}` results in `Subject 1Subject 2Subject 3`.

To separate the subjects, add a separator modifier: `{>[relation-schema-name](|[separating-string]).[field-name]}`.

E.g. `{>Incident_Emails(|, ).subject}` results in `Subject 1, Subject 2, Subject 3`.

8.2. DRILL-THROUGH

You can access records at an even deeper level by using more `.`-operators to traverse lookups and relations.

E.g. `{>Incident_Emails.ownerid.>contact_owning_user(|, ).firstname}` resulting in all the names of the contacts that are owned by the same owner as the emails.

8.3. MODIFIERS

Additional operators can be added to manipulate the result.

DISTINCT

Add `*` to return unique results.

E.g. if there are 3 emails with identical names, then `{>Incident_Emails(*|, ).subject}` results in `Subject of Email`.

ORDERING

Add an ordering field `{>[relation-schema-name](^[ordering-field]).[field-name]}`, and the direction of ordering `{>[relation-schema-name](^[ordering-field]!).[field-name]}`.

E.g. sort email subjects decending `{>Incident_Emails(*^subject!|, ).subject}` results in `Subject 3, Subject 2, Subject 1`.

The ordering field can be different from the returned/accessed field.

8.4. CONDITIONS

Add a condition to filter the returned records: `{>[relation-schema-name]([modifiers])([field-name][optional-operator][optional-value]).[field-name]}`. If an operator and value are not specified, the default comparator is `NotNull`.

E.g. return only the account contacts' last name who are males
`{>contact_customer_accounts(*^firstname!|, )(gendercode==1).lastname}`

Supported operators: `==`, `>=`, `<=`, and `!=`.