Regular Expression Processing in ABAP

Applies to:

This document applies to SAP ECC 6.0, SAP Netweaver 2004s. For more information, visit the ABAP homepage.

Summary

Regular expression is powerful tool for text processing. Regular Expression Processing can help you extend your ABAP capabilities. Many text search and replacement problems are difficult to handle without using regular expression pattern matching. This article provides introduction to regular expression technology in ABAP.

Author: Shaira Madhu

Company: Applexus Software Solutions (P) Ltd

Created on: 25 October 2010

Author Bio

Shaira Madhu is working as SAP Technology Consultant with Applexus Software Solutions (P) Ltd.

Table of Contents

What is Regular Expression?

Business applications are becoming more and more dependent on string processing and string generation because they are required for generating program code or data that is exchanged with markup languages like XML.

Regular expression is powerful tool for text processing. You may use regular expressions in order to validate a string input, extract a particular set of data from a text variable, or may even transform data to meet your business use. A particular data set is termed valid if, and only if, it matches against a certain Regular expression pattern. The most common use of Regular Expression is to search substrings within a data stream that corresponds to a particular Regular Expression pattern.

Pattern matching with regular expression can be applied to real world scenario, such as,

- Email id validation.
- Eliminating special characters from phone numbers.
- Parse data from xml format.
- Conversion of date from one format to other.
- Searching Error log file to get the execption message.

ABAP provide statements and classes for regular expression operation.

ABAP statement – REGEX

ABAP statements **FIND** and **REPLACE** support the use of regular expression with additional clause REGEX.

FIND statement searches the given text for a match with a regular expression, as shown below:

```
FIND REGEX 'A*B' IN 'ABAP'.

FIND ALL OCCURRENCES OF REGEX   pattern
                        IN   text
                   RESULTS   match_result.
```

All the maching expression will be stored in **match_result** variables with type **MATCH_RESULT_TAB.**

REPLACE statement will replace the given regular expression pattern form the given text.

```
REPLACE ALL OCCURRENCES OF REGEX    regpattern
                           IN    text
                         WITH   new
              REPLACEMENT COUNT   cnt.
```

ABAP Classes.

ABAP objects provides two classes for regular expression

- CL_ABAP_REGEX
- CL_ABAP_MATCHER

Regex class **CL_ABAP_REGEX** stores preprocessed RE pattern.This class generates an object-oriented representation from a regular expression in a character-like field. Class CL_ABAP_MATCHER applies a regular expression generated using CL_ABAP_REGEX to either a character string or an internal table.

```
DATA: regex     TYPE REF TO cl_abap_regex,
      matcher   TYPE REF TO cl_abap_matcher.

CREATE OBJECT regex   EXPORTING   pattern = 'ma*'
                                  ignore_case = abap_true.

CREATE OBJECT matcher EXPORTING   regex= regex
                                  text = ‘material’.
```

When creating an object of this class, the regular expression **REGEX** is linked with the text ‘**TEXT’** to be processed or with the table ‘**TABLE’** to be processed. An object of the class **CL_ABAP_REGEX** can be used with any number of objects of the class **CL_ABAP_MATCHER**. All the matching expression will be stored in variables with type **MATCH_RESULT_TAB.**

Find and replace operations are triggered by several methods whose results are stored in the internal state of the object. In addition, the success of an action is returned in the form of Boolean values. The current editing state can be queried using several **get** methods.

For all regex operations such as finding and replacing, the following methods are available:

- constructor(regex, [text], [table])
- find_next([success])
- find_all([matches])
- match(success)
- replace_next(newtext, success)
- replace_found(newtext, success)
- replace_all(newtext, success)

The state and result of the find or replace operation can be queried using the following methods:

- get_match(match)
- get_match(match)
- get_submatch(index, submatch)
- get_offset(index, offset)
- get_length([index], length)
- get_line([line])

In addition, to facilitate programming, methods are provided that can mainly be used in logical expressions:

- create(pattern, [text], [table], [ignore_case], [matcher])
- contains(pattern, text, [ignore_table], [success])
- matches(pattern, text, [success])

Exceptions

When an error occurs, an exception of error class CX_SY_MATCHER is triggered according to the relevant text parameter:

- CX_SY_MATCHER_INVALID_ARGUMENTS

  The specifications of the text to be scanned are invalid.

- CX_SY_MATCHER_NO_CURRENT_MATCH

  A text replacement or result queries were executed without a valid match.

- CX_SY_MATCHER_INVALID_SUBMATCH

  An access was executed to a subgroup that does not exist.

- CX_SY_MATCHER_INVALID_MATCH_OPERATION

  You tried to use the match method for an internal table.

If the exceptions are not caught, the runtime error INVALID_REGEX is triggered.

All of the methods in CL_ABAP_MATCHER are described below.

CONSTRUCTOR - Assignment of a regular expression to a character string

When creating an object of this class, the regular expression **regex** is linked with the text **text** to be processed or with the table **table** to be processed. An object of the class **CL_ABAP_REGEX** can be used with any number of objects of the class **CL_ABAP_MATCHER**.

If neither of the two parameters **text** or **table** is specified, the exception assigned to the class **CX_SY_MATCHER** is raised. If the regular expression **regex** was created in an environment other than the current text environment, the catchable exception CX_SY_REGEX_LOCALE_MISMATCH is triggered.

CREATE - Function for Creating a Matchers

The class method **create (pattern, {text | table}, [ignore_case], [no_submatches}, [simple_regex], matcher)** returns a matcher object for the specified pattern **pattern** and the specified text **text**. The options **ignore_case**, **no_submatches**, and **simple_regex** affect the semantics of the regular expression analogous to the constructor of the class **CL_ABAP_REGEX**.In the parameter **matcher**, a comparison object of the type **CL_ABAP_MATCHER** is returned

In the case of an invalid regular exception, the catchable exception **CX_SY_INVALID_REGEX** is raised. If neither of the parameters **text** or **table** is passed, the catchable exception **CX_SY_INVALID_REGEX_OPERATION** is raised.

FIND_NEXT – Search for Next Match

This method searches for the next match with the regular expression and stores the result internally. The Boolean return value specifies whether another match was found.If the method is successful, you can use the **get** methods to obtain information about the found location or replace it with **replace_found (new)**. If the method is not successful, the last match found is reset.Once the text is fully searched, repeated calls of **find_next(success)** will always return the value **success = ‘ ’.**

```
DATA v_sucess TYPE abap_bool.
CALL METHOD lr_matcher->find_next
                                  RECEIVING
                                  success = v_sucess
      .
```

FIND_ALL – Search for all Subsequent Matches

In the parameter **matches**, this method returns all remaining, non-overlappingmatches of the expression in an internal table of the type **match_result_tab**. If the pattern is not found, an empty table is returned.Calling **find_all (matches)** resets the last match found. Calling a **get** method after **find_all(matches)** raises a catchable exception.After calling **find_all(matches)**, the entire character sequence or table is regarded as processed. Repeated calls of the method will not return any more matches.

```
DATA: lt_result TYPE match_result_tab.

It_result = matcher->find_all ( ).

IF lines (lt_result) GT 0.

v_exist    = ‘Y’.

ENDIF.
```

MATCH – Use Regular Expression for Reamaning Character set

This method checks whether the expression matches the whole, not yet searched section of the character string and returns the result in the Boolean parameter **success**.If the method is successful; you can use the **get** methods to obtain information about the match, analogous to **find_next (success)**. If the method is not successful, the current match is reset.

Once the method has been called, the entire character sequence is regarded as processed. Repeated calls of **match (success)** will not return and further matches.

```
**MATCH - Email id validation.


Data v_pattern = '^([0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*@([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-
zA-Z]{2,9})$'.

lr_matcher = cl_abap_matcher=> create (pattern = v_pattern
                                       Text = ’test@gmail.com’).

CALL METHOD lr_matcher->match
                          RECEIVING
                          success = v_sucess.

      IF v_sucess = abap_false.
            Message 'Invalid email id' TYPE 'I'.
      ENDIF.
```

This method cannot be called when processing internal tables; any attempt to do so will raise the exception CX_SY_INVALID_REGEX_OPERATION..

REPLACE_NEXT - Replacement of the Next Match

This method searches for the next match of the regular expression and replaces it with the character string defined by the replacement string **newtext**.

The Boolean return value **success** contains the value **'X'** if a match is found and replaced.

The stored current match is reset. Calling a **get** method after **replace_next (newtext)** raises a catchable exception.

Calling the method **replace_next (newtext)** has the same effect as the call sequence **find_next ( )** followed by **replace_found (newtext)**.

If the replacement pattern is invalid, the catchable exception **CX_SY_INVALID_REGEX_FORMAT** is raised.

REPLACE_FOUND - Replacement of the Last Match Found

This method replaces the last found match with the character string defined by the replacement pattern **newtext**.

The Boolean return value **success** contains the value **'X'** if the current match was stored and replaced.

The stored current match is reset. Calling a **GET** method after **replace_found(newtext)** triggers a catchable exception.

In the case of **replace_found(success)**, the special character **$`** always refers to the end of the last match or text replacement and not to the start of the line.

If no current match exists, the catchable exception **CX_SY_NO_CURRENT_MATCH** is raised.

If the replacement pattern is invalid, the catchable exception **CX_SY_INVALID_REGEX_FORMAT** is raised.

REPLACE_ALL – Replacement of allSubsequent Matches

This method replaces any current matches as well as all remaining, non-overlapping matches of the regular expression with the character string defined by the replacement pattern **newtext**.

The **count** parameter contains the number of replaced matches. If no matches are replaced, **count** has the value **0**.

The stored current match is reset. Calling a **get** method after **replace_all(matches)** raises a catchable exception.

```
*Eliminating special characters from phone numbers.
DATA: lr_matcher  TYPE REF TO cl_abap_matcher,
       v_cnt       TYPE  I.
TRY.
lr_matcher = cl_abap_matcher=>create( pattern = '\+|\s|\(|\)|-'
                                      text = '+ 1 (123) 567-8910' ).
CATCH cx_sy_matcher .
ENDTRY.

TRY.
CALL METHOD lr_matcher->replace_all
  EXPORTING
    newtext = ''
  receiving
    count   = v_cnt
    .
 CATCH cx_sy_matcher .
ENDTRY.
*We will get the result from lr_matcher->text .The value will be '12345678910'.
```

After **replace_all(count)** has been called, the entire character string or table is regarded as processed. Repeated calls of the method will not make any further replacements.

The contents of special characters like **$1** are recalculated for each match. However, the specialcharacter **$`** always refers to the end of the last match or text replacement **at the time the method was called**.

If the replacement pattern is invalid, the catchable exception **CX_SY_INVALID_REGEX_FORMAT** is raised.

GET_MATCH – Query of the Current State

In the **match** parameter, this method returns the current match of the regular expression of structure of the type **match_result**.

If no current match is stored, the exception **CX_SY_NO_CURRENT_MATCH** is raised.

GET_SUBMATCH – Subgroup Query

This method returns the character string of the subgroup with the number **index** of the current match in the parameter **submatch**. The subgroup **0** represents the complete match.

```
Data v_smatch type string.

Data v_sucess type abap_bool.

CALL METHOD lr_matcher->find_next
                          RECEIVING
                          success = v_sucess

CALL METHOD lr_matcher->get_submatch
                    EXPORTING
                          index     = 0
                    RECEIVING
                          submatch = v_smatch.
```

If no current match is stored, the exception **CX_SY_NO_CURRENT_MATCH** is raised. If the subgroup with the number **index** does not exist, the exception **CX_SY_INVALID_SUBMATCH** is raised.

GET_OFFSET – Query of the Replacement of Last Match Found

This method returns the offset of the current match in the parameter **offset**.If you specify a numeric value **index**, the system returns the offset of the subgroupwith the number **index**. The subgroup **0** represents the complete match.

If no current match is stored, the exception **CX_SY_NO_CURRENT_MATCH** is raised.If the subgroup with the number **index** does not exist, the exception **CX_SY_INVALID_SUBMATCH** is raised.

GET_LENGTH –Query of the Length of the Last Match Found

This method returns the length of the current match in the parameter **length ,**If a numeric value **index** is specified, the system returns the length of thesubgroup with the number **index**. The subgroup **0** represents the complete match.

If no current match is specified, the exception **CX_SY_NO_CURRENT_MATCH** is raised.If the subgroup with the number **index** does not exist, the exception **CX_SY_INVALID_SUBMATCH** is raised.

GET_LINE – Query of the row of the Last Match Found

This method returns the line of the current match in the parameter **line**.

If no current match is stored, the exception **CX_SY_NO_CURRENT_MATCH** is raised.

CONTAINS –Search for Regular Expression

This class static method checks whether the regular expression **pattern** is contained in the search text **text** or in the internal table **table**. A Boolean value is returned as the result.

The semantics of the expression can be influenced by any options supported by the constructor of the class **CL_ABAP_REGEX**.After calling the **contains( )** method, you can use **get_object(matcher)** to create an instance of the class **CL_ABAP_MATCHER**, the status of which corresponds to the result of **contains( ) .**

MATCHES –Test for Match with Regular Expression

This class static method checks whether the regular expression **pattern** is contained in the search text **text** or in the internal table **table**. A Boolean value is returned as the result.

The semantics of the expression can be influenced by any options supported by the constructor of the class **CL_ABAP_REGEX**.After calling the **contains( )** method, you can use **get_object(matcher)** to create an instance of the class **CL_ABAP_MATCHER**, the status of which corresponds to the result of **contains( ) .**

Regular Expression Patterns

Regular Expressions are composed of symbols and characters (literals). I will try to cover some of the commonly-used symbols

Escape character

Special character	Meaning
****	Escape character for special characters

Special Characters for single character string

Special character	Meaning
.	Placeholder for any single character
\C	Placeholder for any single character
\d	Placeholder for any single digit
\D	Placeholder for any character other than a digit
\l	Placeholder for any lower-case letter
\L	Placeholder for any character other than a lower-case letter
\s	Placeholder for a blank character
\S	Placeholder for any character other than a blank character
\u	Placeholder for any upper-case letter
\U	Placeholder for any character other than an upper-case letter
\w	Placeholder for any alphanumeric character including _
\W	Placeholder for any non-alphanumeric character except for _

[ ]	Definition of a value set for single characters
[^]	Negation of a value set for single characters
[-]	Definition of a range in a value set for single characters
[[:alnum:]]	Description of all alphanumeric characters in a value set
[[:alpha:]]	Description of all letters in a value set
[[:blank:]]	Description for blank characters and horizontal tabulators in a value set
[[:cntrl:]]	Description of all control characters in a value set
[[:digit:]]	Description of all digits in a value set
[[:graph:]]	Description of all graphic special characters in a value set
[[:lower:]]	Description of all lower-case letters in a value set
[[:print:]]	Description of all displayable characters in a value set
[[:punct:]]	Description of all punctuation characters in a value set
[[:space:]]	Description of all blank characters, tabulators, and carriage feeds in a value set
[[:unicode:]]	Description of all Unicode characters in a value set with a code larger than 255
[[:upper:]]	Description of all upper-case letters in a value set
[[:word:]]	Description of all alphanumeric characters in a value set, including _
[[:xdigit:]]	Description of all hexadecimal digits in a value set
\a \f \n \r \t \v	Diverse platform-specific control characters
[..]	Reserved for later enhancements
[==]	Reserved for later enhancements

Special Characters for character string patterns

Special character	**Meaning**
{n}	Concatenation of **n** single characters
{n,m}	Concatenation of at least **n** and a maximum of **m** single characters
{n,m}?	Reserved for later enhancements
?	One or no single characters
*****	Concatenation of any number of single characters including 'no characters'
***?**	Reserved for later enhancements
+	Concatenation of any number of single characters excluding 'no characters'
+?	Reserved for later enhancements

|	Linking of two alternative expressions
( )	Definition of subgroups with registration
(?:)	Definition of subgroups without registration
\1, **\2**, **\3** ...	Placeholder for the register of subgroups
\Q ... **\E**	Definition of a string of literal characters
(? ... **)**	Reserved for later enhancements

Special Characters for search string

Special character	Meaning
^	Anchor character for the start of a line
\A	Anchor character for the start of a character string
$	Anchor character for the end of a line
\Z	Anchor character for the end of a character string
\<	Start of a word
\>	End of a word
\b	Start or end of a word
\B	Space between characters within a word
(?=)	Preview condition
(?!)	Negated preview condition

Special Characters for replacement texts

Special character	Meaning
$0, **$&**	Placeholder for the whole found location
$1, **$2**, **$3**...	Placeholder for the register of subgroups
$`	Placeholder for the text before the found location
$'	Placeholder for the text after the found location

Related Content

Special Characters in Regular Expressions

Regular Expressions for Information Processing in ABAP

Regular Expression Processing using ABAP

For more information, visit the ABAP homepage.

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.