



PUBLIC

How to Use Regular Expression in Format Definition

Applicable Releases:

SAP Business One 2007

SAP Business One 8.8

All Countries

English

July 2009

Table of Contents

Introduction.....	3
Regular Expression	3
Regular Expressions in Format Definition	3
Using Regular Expressions in Format Definition	5
Using Regular Expressions Encapsulated in the Locate by Functions	6
Composing Regular Expressions.....	7
Examples of Typical Regular Expressions	10
Example 1.....	10
Example 2.....	11
Example 3.....	11
Example 4.....	12
Example 5.....	12
Example 6.....	13
Example 7.....	13
Example 8.....	14
Example 9.....	15
Example 10.....	15
Example 11.....	16
Example 12.....	16
Example 13.....	17
Regular Expression Language Elements.....	19
Metacharacters.....	19
Escaped Character.....	20
Character Class.....	21
Look Around Assertion	21
Copyrights, Trademarks, and Disclaimers	23

Introduction

Regular Expression

Regular expression is a powerful computing language that lets you identify, analyze, and manipulate strings of text.

Regular expression is used in various applications such as word processing software, validation of user input to web pages, information retrieval, and so on. All the mainstream programming languages implement regular expression despite the differences in their function coverage and syntax.

If you have experience with wildcard characters such as “?” and “*” in the DOS operating system, you may notice the similarities between these wildcard characters and regular expressions. For example, the DOS command “dir *.txt” lists all files with the “txt” extension. The “*” (asterisk) stands for zero or more characters.

In general, the regular expression language defines the following two kinds of characters:

- Metacharacters: characters having special meaning in a program
- Literal: all other characters

For example, the DOS ‘?’ and ‘*’ characters are metacharacters.

A regular expression is composed of both character types in sequences that act as a pattern to search for all occurrences of conformant strings.

This document contains the following information:

- The use of regular expressions in the SAP Business One 2007 Format Definition add-on
- Examples of typical regular expressions
- Regular expression language elements

All the regular expressions in this document are highlighted in a bold blue font.

For example, the expression **\d{4}-\d{2}-\d{2}** matches all dates written in the YYYY-MM-DD format.

Regular Expressions in Format Definition

The SAP Business One 2007 Format Definition add-on uses the regular expression language to retrieve different business fields from bank statement text files.

Format Definition enables you to retrieve field data with the following methods:

- Regular expressions encapsulated in the following *Locate by* functions:
 - Sub-node
 - Keyword
 - Absolute position
 - Separator, used in CSV (Comma Separated Values) files
- Regular expressions you compose manually

Compose regular expressions only when the encapsulated regular expressions are not adequate for your needs.



Note

The regular expressions in Format Definition are based on the regular expressions in Microsoft.NET. The regular expression options are not set during runtime execution.

Using Regular Expressions in Format Definition

You can use the *Locate by* function to set a regular expression for a segment or a field.

Prerequisites

You have created a segment or a field.

Procedure

1. From the SAP Business One *Main Menu*, choose the segment or field for which you want to set a regular expression.
2. In the *Locate by* dropdown box on the *Attribute* tab, select one of the following:
 - Keyword
Enter data in the following fields:
 - Starting Keyword
 - Ending Keyword
 - Length
 - Prior Keyword
 - Following Keyword
 - Position
Enter data in the *Position* and *Length* fields.
 - Separator
 - Enter data in the *Separator* and *Index* fields.
 - Choose *Regular Expression in Separator* if necessary.
 - Sub-nodes
Choose one of the following:
 - Sub-Nodes are Sequential
 - Sub-Nodes are Alternative
 - Regular Expression
In the *Regular Expression* field, enter the regular expression you create to retrieve the information from the bank statement file.
3. Save your entries.

See also:

- For more information on how to use *Keyword*, *Position*, *Separator*, and *Sub-nodes*, see [Using Regular Expressions Encapsulated in the Locate by Functions](#).
- For more information on how to compose regular expressions, see [Composing Regular Expressions](#).

Using Regular Expressions Encapsulated in the Locate by Functions

Format Definition provides some typical regular expressions encapsulated in the *Locate by* functions:

- Locate by keyword

Use this function when you match the text string as follows:

- Starting with a keyword which may or may not have a fixed length or a length range
When you match a line starting with a specified record identification, set the *Starting Keyword* and *Length* fields by setting "m,n" for a length range. "m,n" stands for "from m to n".
- Between two keywords (inclusive), you set the *Starting Keyword* and *Ending Keyword* fields.
- Between two keywords (exclusive), you set the *Prior Keyword* and *Following Keyword* fields.
- Combination of the 3 items above

- Locate by position

Use this function when you match the text string as follows:

- Starting at an absolute position and with a fixed length or length range, you set the *Position* and *Length* fields.
- Starting from an absolute position and continuing to the end of the text string, set the *Position* field and leave the *Length* field blank.

- Locate by sub-nodes

Use this function when you match a group of:

- Sequential records: you choose *Sub-notes are sequential*
- Substitution records: you choose *Sub-notes are alternative*



Note

The sub-nodes can be matched in an exact order to the format tree of Format Definition, just as alternation in regular expressions.

For more information on alternation, see the alternation expression section in [Composing Regular Expressions](#).

- Locate by separator

Use this function when you match a field separated by:

- A separator character: specifically in CSV format, you set the *Separator* and *Index* fields
- Possible separators matched by a specified regular expression: you set the *Separator* and *Index* fields and choose the *Regular expression in separator* checkbox.

You can use the Locate by function to set a regular expression for a segment or a field.

See also:

For more information on Format Definition, see the Format Definition 2007 Online Help.

Composing Regular Expressions

When the regular expressions encapsulated in the *Locate by* function are not sufficiently adequate for your needs, you need to manually compose regular expressions.

You can combine literal characters and metacharacters to create a new regular expression. You can compose a regular expression from a single character to a complex structure, depending on your needs.

Single Character Expression

A single character is the smallest regular expression unit. It can be:

- A letter or a digit
- An [escaped character](#) such as `\t` (tab)
- A [character class](#) (set) such as `.` (dot), `\s` (white-space), or `[0-5]` (digit from 0 to 5)

A character class can match all the single characters included in the set.

For example, for the text "2008", the regular expression `[0-5]` has three matches, which are "2", "0" and the second "0".

Sub-regular Expression

A sub-regular expression is composed by concatenating single characters. Concatenating characters together matches continuous characters in a text, in the exact order of characters in regular expressions.

For example, the regular expression `C\d\d\d\d` comprises one literal character `C` and four character classes `\d`. `C` matches the single letter "C". Each `\d` matches one digit and `\d\d\d\d` matches four digits in a sequence. When you use the `C\d\d\d\d` sequence to match the text "Payers: C1000, C1020, V1100, C1300", the matched texts are "C1000", "C1020" and "C1300".

Quantifier Expression

You can assign quantifiers to a regular expression. A quantifier expression applies to the single character or the sub-regular expression that immediately precedes it. Quantifiers enable regular expressions to match the text for the specified times defined by the assigned quantifiers.

Examples on Quantifier Expression

Quantifier	Quantifier Expression	Actions
{n}	<code>\d{4}</code>	In the text "2008", <code>\d{4}</code> matches four digits.
+	<code>[w\s]+</code>	Matches texts by the use of one or more occurrences of <code>[w\s]</code> , which is a character class of either alphanumeric or white-space <code>[w\s]+</code> matches texts such as "Business ", "Business One ", or "B1 2007".
*	<code>(C\d{4})*</code>	Matches texts by the use of zero or more occurrences of <code>C\d{4}</code> , which is grouped by parentheses and matches texts starting with "C" and followed by four digits In the text "Payers: C1000C1020, V1100, C1300", <code>(C\d{4})*</code> matches the expected texts, such as "C1000C1020", "C1300", as well as some empty strings

		before or after the expected two matches. * matches empty strings. To eliminate matches of empty strings, use +.
--	--	---

You can concatenate sub-regular expressions to create a more complicated regular expression.

Alternation Expression

You can separate different sub-regular expressions with a “|” (vertical bar) to create an alternation expression that matches texts with each sub-regular expression from left to right and stops wherever a match succeeds.

For example, **C|D** uses the first sub-regular expression **C** to match texts, but it uses the second sub-regular expression **D** only if **C** cannot match any texts.

By means of concatenation, alternation, groups, and quantifiers, you can compose almost all kinds of regular expressions. However, sometimes you might need to use *look around assertions* for more accurate matches.

Look Around Assertion

[Look around assertions](#) address the matches that have prerequisites to texts on the left or right adjacent positions. They include *look ahead* and *look behind* assertions.

For example, in a transaction line of a bank statement you want to find an amount such as “:61:0711211121C216,08N051NONREF” or “:61:071121C216,08N051NONREF”, both of which comprise:

- A record identification “:61:”
- A date in the YYMMDD format (The first transaction also contains an optional date in the MMDD format.)
- One or two characters used to signify the direction of money flows
- A numerical amount value with two decimals and a comma as a separator
- Some other information

Since an optional four-digit date might exist in transaction lines, you cannot find the amount easily only by the absolute position. You can use look behind assertions in regular expressions.

The final regular expression is `(?<=:61:\d{6}(\d{4})?[a-zA-Z]{1,2})\d*,\d{2}`, in which the amount is designated by `\d*,\d{2}` and the look behind assertion is `(?<=:61:\d{6}(\d{4})?[a-zA-Z]{1,2})`.

The detailed explanation for the regular expression is displayed in the table below.

Target Text	Regular Expression
Record identification	<code>:61:</code>
Date YYMMDD	<code>\d{6}</code>
Date MMDD (Optional)	<code>(\d{4})?</code>
Direction of money flow	<code>[a-zA-Z]{1,2}</code>
Amount	<code>\d*,\d{2}</code>

It takes time to compose a complex regular expression. You can compose it step by step, starting from a single character and continuing to a complex structure. At each step, you can test the sub-regular expression in the SAP Business One Format Definition add-on.



Note

In the format tree of Format Definition, the current node matches the texts corresponding to the texts its parent node matches.

For example, the parent node “NoteToPayee” matches three texts and its child node “Ref” matches these three texts respectively.

See also:

For more information, see [Regular Expression Language Elements](#).

Examples of Typical Regular Expressions

All the examples below use the following template:

Description:

<Example description>

Regular Expression:

<The regular expression used to match the text>

Text:

<The text is displayed in gray. >

<All the matching texts are highlighted alternately in blue and red. >

Matches:

<The number of matches>

Extended Applications:

<The extended application of the regular expression used in the example>

Example 1

Description:

To match a string of alphanumeric characters with a fixed length, for example, a fixed length of 8.

Regular Expression:

`\w{8}`

Text:

:25:11457700

Matches:

1

Extended Applications:

- Match varied strings of alphanumeric characters:

String composed of	How to Modify	Modified Regular Expression
Characters with a variable range	Replace <code>{8}</code> with a range such as <code>{8,15}</code>	<code>\w{8,15}</code>
Zero or more characters	Replace <code>{8}</code> with <code>*</code> (star)	<code>\w*</code>
One or more characters	Replace <code>{8}</code> with <code>+</code> (plus)	<code>\w+</code>
Zero or one character	Replace <code>{8}</code> with <code>?</code> (question mark)	<code>\w?</code>
Pure numeric characters	Replace <code>\w</code> with <code>\d</code>	<code>\d{8}</code>
Pure letters	Replace <code>\w</code> with <code>[a-zA-Z]</code> (both lowercase and uppercase);	<code>[a-zA-Z]{8}</code>

A small range of characters For example: CDE	Replace \w with [CDE]	[CDE]{8} or [C-E]{8}
Anything	Replace \w with . (dot)	.{8}

- Match **\w** to only a single letter or digit, equivalent to **[0-9a-zA-Z]**.
- Perform a lazy match.

For example:

If you want to match as few characters as possible, add **?** after:

- **{8,15}** for a string of characters with the variable range of 8 to 15: **\w{8,15}?**
- ***** for a string of zero or more characters: **\w *?**
- **+** for a string of one or more character: **\w +?**

Example 2

Description:

To match a numeric value in the 999,999 format, in which the length of a decimal number is not fixed

Regular Expression:

(\d+(\,\d*)?)(\,\d+)

Text:

transaction 1: Amount is 100,600 EUR; Opposite account: 991234

transaction 2: Amount is ,80 EUR; Opposite account: 991235

transaction 3: Amount is 220 EUR; Opposite account: 991245

Matches:

3

Extended Applications:

- If the decimal point is not a “,” (comma), replace each “,” in the regular expression **(\d+(\,\d*)?)(\,\d+)** with the decimal point character.

For example, if the decimal point is a “.” (dot), then the new regular expression is **(\d+(\.\d*)?)(\.\d+)**.

The “.” needs to be escaped as a “\.” because a “.” has a special meaning in regular expressions.

- If you have more than one numeric value in the text to match, you might get some annoying matches.

For example, for the text “:61:0711211121C216,08N051NONREF”, the four matches are

:61:0711211121C216,08N051NONREF.

To use look around assertions to match the accurate amounts, see [Example 6](#).

Example 3

Description:

To match a numeric value in the 999,99 format, in which 2 decimal numbers always exist after the “,”

Regular Expression:`\d*, \d{2}`**Text:**

Transaction 1: Amount is 100,60 EUR; Opposite account: 991234

Transaction 2: Amount is ,80 EUR; Opposite account: 991235

Matches:

2

Extended Applications:If the length of decimal numbers is not 2, you can replace `{2}` with another number.For example: `{4}`.

Example 4

Description:

To match the account number in 8 digits after the keyword “:25:” but before the other keyword “/”

Regular Expression:`(?<=:25:)\d{8}(?=/)`**Text:**

:25:74061101/4073193

Matches: 1**Extended Applications:**

- To match everything between two keywords, use the *Locate by keyword* function in Format Definition.
- To match different keywords, replace keywords such as “:25:” and “/” in look around assertions.
- To match from any position other than “:25:”, remove the look behind assertion `(?<=:25:)`.
- To match until any position other than “/”, remove the look ahead assertion `(?=/)`.
- Change the length “8” in the sub expression `\d{8}` to:
 - Another number such as “15”, to match 15 digits
 - A range such as “8,15”, to match the numeric string with 8 to 15 digits
 - A range such as “8, ”, to match the numeric string with 8 to numeric digits that are as large as possible

Example 5

Description:

To match alternative strings. For example, the record identification is either “:60F:” or “:60M:”.

Regular Expression:`:60(F|M):.*`**Text:**

:60F:D080516EUR000000208822,29

```
:some other text:
```

```
:60M:D080516EUR000000208822,29
```

Matches:

2

Extended Applications:

To extract the common part from all alternative strings:

1. Inside the parentheses, list the different parts and separate them with a “|” (vertical bar).
2. Add the other consequent common part, if any, to the parentheses.

Example 6

Description:

To match a field between the optional fields in the text

For example, “:61:0711211121C216,08N051NONREF” or “:61:071121C216,08N051NONREF”, comprises a record identification (“:61:”), a date in the YYMMDD format (“071121”), another optional date MMDD (“1121”), one or two characters to signify the direction of money flows (“C”), a numeric amount value (“216,08”), and some other information (“N051NONREF”). The numeric amount value (“216”) in the text is the target field to match.

Regular Expression:

```
(?<=:61:\d{6}(\d{4})?[a-zA-Z]{1,2})\d*, \d{2}
```

Text:

```
:61:0711211121C216,08N051NONREF
```

Matches:

1

Extended Applications:

All the fields before the target field are described in the look behind assertion embraced by “(?<=”and”)”. Especially, the optional field is embraced by parentheses and then a “?” (question mark). The sub-regular expression for the amount is the same as the one in example 3.

You can compose your own regular expressions for such cases in the form of

```
(?<=REGEX_FOR_FIELDS_IN_FRONT)(REGEX_FOR_TARGET_FIELD)
```

in which REGEX_FOR_FIELDS_IN_FRONT and REGEX_FOR_TARGET_FIELD are the regular expressions for the fields in front and the target field, respectively.



Always keep the parentheses therein.

Example 7

Description:

To search for all numbers, which are possibly document identifications, in free text descriptions, for example, to search for an invoice number

Regular Expression:

```
(?<=b)(?<!\.)\d+(?=b)(?!\.)
```

Text:

```
:86:GIRO 6890316
ENERGETICA NATURA BENELU
AFRIKAWEG 14
HULST
3187-A1176
TRANSACTIONEDATUM* 03-07-2007
```

Matches:

6

Extended Applications:

- The regular expression `(?<=\\b)(?!\\.)d+(?=\\b)(?!\\.)` used in the example finds all the digits between word boundaries except the digits with a dot after or before. The "." (dot) is escaped as `\\.`
- It may find some inaccurate matches, such as the dates in the text. If you want to exclude the inaccurate matches, you can exclude the "-" (hyphen) that is a prior or following character of the dates. You modify the expression in the same way as you do for the "." (dot), and the regular expression becomes `(?<=\\b)(?!\\.|\\-)(?!\\-\\)d+(?=\\b)(?!\\.|\\-)(?!\\-)`.

The matches are:

```
:86:GIRO 6890316
ENERGETICA NATURA BENELU
AFRIKAWEG 14
HULST
3187-A1176
TRANSACTIONEDATUM* 03-07-2007
```

However, you may lose some digits that need to be matched, such as "3187" before the "-".

Example 8

Description:

To find a BP account number in 9 digits with a prior "P" or "0" at the beginning of free text descriptions

Regular Expression:

```
(?<=^(P|0))d{9}
```

Text:

```
0000006681 FORTIS ASR BETALINGSCENTRUM BV
```

Matches:

1

Extended Applications:

- Use the positive look behind assertion `(?<=PRIOR_KEYWORD)` to represent the prior keyword.
- The "A" indicates that matches start from the beginning of the text. If the text includes the record identification, you may also use it to replace the "A" in the look behind assertion.

For example:

:86:000006681 FORTIS ASR BETALINGSCENTRUM BV

The regular expression becomes `(?<=:86:(P|0))\d{9}`.

Example 9

Description:

Continues example 8 and searches for the relevant BP name after the BP account number. The BP name is composed of a letter, dot, or space.

Regular Expression:

`(?<=^(P|0)\d{9})[a-zA-Z.]*`

Text:

0000006681 FORTIS ASR BETALINGSCENTRUM BV

Matches:

1

Extended Applications:

Put the BP account number in the look behind assertion of the regular expression.

Example 10

Description:

To search all possible document identifications in a sub-record of a record with the documentation identification of “:86:”. Examples of a sub-record are “?00”, “?10”, and so on.

A sub-record, containing possible document identifications, is composed of the following parts:

- Keywords such as “RE”, “RG”, “R”, “INV”, “NR”, “NO”, “RECHN” or “RECHNUNG”
- An optional group comprised of all the following:
 - A dot, hyphen, or slash separator
 - An optional space
 - An optional string starting with the keyword “NR” or “NO” followed by a dot, hyphen, or slash separator
 - An optional space
- Document identification in digits

Regular Expression:

`(?<=?\d{2})(RE|RG|R|INV|NR|NO|RECHN|RECHNUNG)(\.-|/)\s*((NR|NO)(\.-|/))?\s?)?\d+`

Text:

:86:004?00LASTSCHRIFT.O.E.?10931?20R8182019,R8182029?3020070000

?31273353300?32PILOT.COMPUTERHANDELS.GMBH?34000

Matches:

1

Extended Applications:

- Put all possible strings before the target document identifications of a sub-record in the look behind assertion as `\?\d{2}(RE|RG|R|INV|NR|NO|RECHN|RECHNUNG)(\.-|/)\s*((NR|NO)(\.-|/))?\s?)?`.

- Put the optional part of the sub-record inside the parentheses and add a question mark after the closing parenthesis.



More than one optional string exists in the text above. One example is `((NR|NO)(\.-|/))?`.

Example 11

Description:

To search for one type of record among all the lines in a CSV file containing two types of records, each with a different number of fields. For example, to search for the records containing 35 fields.

Regular Expression:

```
(?<=^\r\n)([^\r\n;]*){34}[^;\r\n]*?(\\r\\n|$)
```

Text:

```
25070070;021983200;66;04.04.08;EUR;0.00;4933.83;4933.83;0.00;Freestander.Hildesheim;CDS.Hildesheim;;;;;;;;;9
```

```
25070070;021983200;67;07.04.08;EUR;0.00;26890.74;26890.74;0.00;Freestander.Hildesheim;CD S.Hildesheim;;;;;;;;;9
```

```
25070070;021983200;66;07.04.08;9700/702;65570874.02.04.2008.04.12;UEBERWEISUNG;;TRF;65570874.02.04.2;45.10;;;04.04.08;;;;;;;;;;;;;ICP.GMBH;;30050000;0001458710;051;
```

```
25070070;021983200;66;07.04.08;9700/702;ABR...158321697.V...02.04.08;UEBERWEISUNG;;TRF;NON REF;1524.12;;;04.04.08;;;DI.....32,86.MWS.....6,42;;;;;;;;;;;;;CONCARDIS.GMBH;;50040000;0600200001;051;
```

Matches:

2

Extended Applications:

- The regular expression `(?<=^\r\n).*\\r\\n|$)` matches every line in the text:
 - a. From the beginning of the text or after a line break (`\r\n`)
 - b. Until another line break or the end of the text (`$`)

The regular expression used in this example further limits the 35 fields separated by a “;” (semicolon). Its sub expression `([^\r\n;]*){34}[^;\r\n]*?` limits the pattern of content in every line. The first 34 fields are made up of zero to many non line-break characters followed by a semicolon. And the last field ends with an optional semicolon.

- Replace the “;” with other separators, in your case, for example, a “,” (comma).

Example 12

Description:

Continuing example 11, with the only difference being that the record in this example has either 35 fields or 37 fields.

Regular Expression:

```
(?<=^\r\n)([^\r\n;]*){34}(((^\r\n;)*?(\\r\\n|$))|(((^\r\n;)*){2}[^;\r\n]*?(\\r\\n|$)))
```

Text:


```
25070070;021983200;66;04.04.08;EUR;0.00;4933.83;4933.83;0.00;Freestander.Hildesheim;CDS.
Hildesheim;;;;;;;;;9
```

```
25070070;021983200;67;07.04.08;EUR;0.00;26890.74;26890.74;0.00;Freestander.Hildesheim;CD
S.Hildesheim;;;;;;;;;9
```

```
25070070;021983200;66;07.04.08;9700/702;65570874.02.04.2008.04.12;UEBERWEISUNG;;TRF;6557
0874.02.04.2;45.10;;;04.04.08;;;;;;;;;;;;;ICP.GMBH;;30050000;0001458710;051;
```

```
25070070;021983200;66;07.04.08;9700/702;ABR..158321697.V..02.04.08;UEBERWEISUNG;;TRF;NON
REF;1524.12;;;04.04.08;;;DI.....32,86.MWS.....6,42;;;;;;;;;;;;;CONCARDIS.GMBH;;500400
0;0600200001;051;
```

Matches:

2

Extended Applications:

Use alternation in cases such as example 12:

1. Extract the common part, for example, the first 34 fields: `(?<=^\r\n)([^\r\n;]*){34}`.
2. Put each of the different parts of both the 35 and 37 field record types in the inner parentheses.
3. Separate the different parts with a “|” (vertical bar).
4. Put the different parts inside a pair of outer parentheses.

Different part for case with 35 fields: `[^\r\n]*;?(r\n|$)`

Different part for case with 37 fields: `([^\r\n;]*){2}[^\r\n]*;?(r\n|$)`

Example 13**Description:**

To search one bank statement line in the relevant records of MT940 files. Typically the MT940 file consists of one line with the record identification “:61:” followed by an optional record with the identification “:86:” that has a maximum of six lines.

Regular Expression:

```
:61:.*\n(:86:.*\n(.*\n){0,5})?(?=(?:61:)|(:62(F|M)))|)$
```

Text:

```
:20:STARTUMS..
```

```
:25:74061101/4073193
```

```
:28C:0
```

```
:60F:C0711221EUR37981,28
```

```
:61:0711221122C334,62N051NONREF
```

```
:86:051?00GUTSCHRIFT?10931?20BUCH.ST..021075200.20112007
```

```
?21RENR..89098.V.20.10.?22KASS.Z..KD-NR..13894?3074051230
```

```
?31190000059?32STADT.GRAFENAU?34000
```

```
:61:0711221122C773,28N051NONREF
```

```
:86:051?00GUTSCHRIFT?10931?20BUCH.ST..121209350.20112007
```

```
?21RENR..88828.V.13.10.?22KASS.Z..KD-NR..12862?3074051230
```

```
?31190000059?32STADT.GRAFENAU?34000
:61:0711231122C19,90N051NONREF
:86:051?00GUTSCHRIFT?10931?20EINGANGSVERR..0000002
?2121.11.07./.68120124.071120?22TRANSACT...REFNR.0124100000
?3074061101?314073193?32KOMMUNIKATION.WEN?34000
:61:0711221122C629,58N051NONREF
:86:051?00GUTSCHRIFT?10931?20TID.68120124.VOM.19.11.2007
?21EC.KARTENUMSAETZE?3037010050?31555633508?32TRANSACT.GMBH?34000
:62F:C071122EUR45321,58
```

Matches:

3

Extended Applications:

The regular expression `:61:.*\n(:86:.*\n(.*){0,5}?)?(?=(?:61:)|(?:62(F|M)))|)$` matches each possible text until it meets the next line starting with “:61:”, “:61F:”, or “:62M”, or the end of the text (\$). The “:86:” record block includes the first line starting exactly with “:86:” and followed by up to five lines. It uses a lazy match by adding a “?” (question mark) to the end of `(.*){0,5}` to avoid matches across the following bank statement line record.


Otherwise, the matches are demonstrated as follows:


```
:20:STARTUMS..
:25:74061101/4073193
:28C:0
:60F:C071121EUR37981,28
:61:0711221122C334,62N051NONREF
:86:051?00GUTSCHRIFT?10931?20BUCH.ST..021075200.20112007
?21REN..89098.V.20.10.?22KASS.Z..KD-NR..13894?3074051230
?31190000059?32STADT.GRAFENAU?34000
:61:0711221122C773,28N051NONREF
:86:051?00GUTSCHRIFT?10931?20BUCH.ST..121209350.20112007
?21REN..88828.V.13.10.?22KASS.Z..KD-NR..12862?3074051230
?31190000059?32STADT.GRAFENAU?34000
:61:0711231122C19,90N051NONREF
:86:051?00GUTSCHRIFT?10931?20EINGANGSVERR..0000002
?2121.11.07./.68120124.071120?22TRANSACT...REFNR.0124100000
?3074061101?314073193?32KOMMUNIKATION.WEN?34000
:61:0711221122C629,58N051NONREF
:86:051?00GUTSCHRIFT?10931?20TID.68120124.VOM.19.11.2007
?21EC.KARTENUMSAETZE?3037010050?31555633508?32TRANSACT.GMBH?34000
```

Regular Expression Language Elements

Metacharacters

All the characters that are not listed in the following metacharacter table are literal characters. You can use them directly to match themselves.

Metacharacter	Description
.	Matches any single character except \n (line feed)
*	<p>A quantifier specifying zero or more occurrences</p> <p>For example: <code>.*</code> matches a string until \n, while the match can be an empty string.</p>  <p><code>*</code> is always for a greedy match, because it matches as many characters as possible.</p> <p>For example, to match the text "regex is powerful" with <code>.*s</code>, the match is "regex is " other than "regex ". If you want to match the latter, use <code>.*?s</code>. (The lazy version of <code>*</code> is <code>*?</code>, and this rule also applies to <code>+</code>, <code>?</code>, <code>{n,m}</code>.)</p>
+	<p>A quantifier specifying one or more occurrences</p> <p>For example, <code>.+</code> matches a string until \n, while the match must contain at least one character, so it cannot match an empty string.</p>
?	<p>A quantifier specifying zero or one occurrence</p> <p>For example, <code>?.</code> matches an optional character that contains either exactly one character or no character (empty string).</p>
{ }	<p>A quantifier specifying a range of occurrences</p> <p>For example, <code>{2}</code> matches a string with exactly two characters; <code>{2,5}</code> matches a string with at least two but no more than five characters; and <code>{2,}</code> matches a string with at least two characters.</p>
^	<p>Designates the beginning of a text except for the characters used between [and]</p> <p>For example, <code>^{4}</code> matches the first four characters in a text.</p>
\$	<p>Designates the end of a text</p> <p>For example, <code>^.*\$</code> matches the whole text only if the text is a single line; <code>^(.*\n)*.*\n?\$</code> can match the whole text, either of a single line or multiple lines.</p>
-	<p>Becomes a metacharacter only when used between [and]</p> <p>Specify a continuous range of characters in character classes.</p> <p>For example, <code>[0-9]</code> matches a digit, equivalent to <code>\d</code>; <code>[a-z]</code> matches a letter in lowercase.</p>
[]	<p>Matches a single character included in specified character sets</p> <p>For example, <code>[aeiou]</code> matches any letter in a group of a, e, i, o, u; <code>[^aeiou]</code></p>

	<p>matches any character not in a group of a, e, i, o, u.</p> <p>[0-9a-z] matches any single digit or single letter in lowercase.</p>
	<p>Matches an alternation of sub-regular expressions separated by a “ ” (vertical bar); the leftmost successful match wins.</p> <p>For example, C D matches either C or D; IN PU matches either IN or PU.</p> <p></p> <p>If you concatenate alternation sub-regular expressions with other expressions, put them inside parentheses. For example, to match the document IDs of all A/R and A/P invoices, use (IN PU)d+; otherwise (IN PU)\d+ becomes the alternation of IN and PU\d+.</p>
\	<p>A special character used to escape some other metacharacters from their special meanings</p> <p>For example, * matches literal “*” (asterisk);</p> <p>\\ matches literal “\” (backslash).</p>
()	<p>Groups substrings in a match</p> <p>For example, col(o ou)r matches either the word “color” or “colour”;</p> <p>in contrast, colo our matches either the word “colo” or “our”.</p>

Escaped Character

The escaped character “\” (a single backslash) signals to a regular expression parser that a character following the backslash is not an operator.

You can create some useful escaped characters by combining the escape character “\” with some other characters.

Escaped character	Description
\r	Matches a carriage return
\n	Matches a new line feed In Microsoft Windows, a line break for text files is always the combination of a carriage return and a line feed. For example: \r\n
\t	Matches a tab character
\b	Matches a word boundary (between \w and \W characters) For example, \b[a-z]*\b matches a word in lowercase.
\u<Unicode>	Matches a Unicode character <Unicode> is the Unicode value of a character in hexadecimal form (four digits). For example, \u0020 matches a space.
\x<ASCII>	Matches an ASCII character

	<p><ASCII> is the ASCII value of a character in hexadecimal form (two digits). For example, <code>\x20</code> matches a space.</p>
--	--

Character Class

You can create a character class by combining the escape character “\”(backslash) and some special characters.

You can regard character class as a shortcut for the patterns inside [and] in the table below.

Character class	Description
<code>\s</code>	Matches any white-space character, equivalent to <code>[\r\n\t\x20]</code>
<code>\S</code>	Matches any non white-space character, equivalent to <code>[^\r\n\t\x20]</code>
<code>\d</code>	Matches any decimal digit, equivalent to <code>[0-9]</code>
<code>\D</code>	Matches any non-decimal digit, equivalent to <code>[^0-9]</code>
<code>\w</code>	Matches any alphanumeric character, equivalent to <code>[0-9a-zA-Z]</code>
<code>\W</code>	Matches any non alphanumeric character, equivalent to <code>[^0-9a-zA-Z]</code>

Look Around Assertion

Look around assertions address the matches that have prerequisites to texts on the left or right adjacent position.

Look around	Description
<code>(?<=<condition>)</code>	<p>Positive look behind assertion</p> <p>Continue matching only if the regular expression <condition> is matched successfully at the position on the left.</p> <p>For example, <code>(?<=(IN PU))\d+</code> matches digits after either keyword IN or PU.</p>
<code>(?!<condition>)</code>	<p>Negative look behind assertion</p> <p>Continue matching only if the regular expression <condition> is not matched at the position on the left.</p> <p>For example, <code>(?!19)\d{2}</code> matches two digits without a prior “19”.</p>
<code>(?=<condition>)</code>	<p>Positive look ahead assertion</p> <p>Continue matching only if the regular expression <condition> is matched successfully at the position on the right.</p> <p>For example, <code>(?<=:25:)\d{8}(?!/)</code> matches the eight digits after “:25:” but before a slash.</p>
<code>(?!<condition>)</code>	<p>Negative look ahead assertion</p> <p>Continue matching only if the regular expression <condition> is not matched at the position on the right.</p> <p>For example, <code>\d{4}(?!USD)</code> matches all the four digits except those followed by “USD”.</p>

For more information on the regular expression language in Microsoft.Net, see the MSDN [http://msdn.microsoft.com/en-us/library/aa719739\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/aa719739(VS.71).aspx).

Copyrights, Trademarks, and Disclaimers

© Copyright 2009 SAP AG. All rights reserved.

The current version of the copyrights, trademarks, and disclaimers at <http://service.sap.com/smb/sbocustomer/documentation> is valid for this document.