SAFR - Scalable Architecture for Financial Reporting
PDF from SAFR Information Center 4.15.00

Logic Text Syntax
SAFR Workbench 4.15.000

SAFR - Scalable Architecture for Financial Reporting
PDF from SAFR Information Center 4.15.00

Logic Text Syntax
SAFR Workbench 4.15.000

Note

Before using this information and the product it supports, read the information in "Appendix: Notices" on page 147.

Fourth Edition

This edition applies to version 4, release 15, modification 000 of SAFR Workbench (part of SAFR product number 6949-17P) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Rules for all logic text

Rules **Notes**

Extra blanks between keywords and expressions have no effect.

For example, these IF statements are all the same:

```
IF ({field1}={field2}) THEN COLUMN=
   {field1} ENDIF

IF ({field1} = {field2}) THEN
    COLUMN = {field1} ENDIF

IF
  ({field1}
           =
            {field2})
                     THEN
                         COLUMN
                          =
                          {field1}
                              ENDIF

IF ({field1} = {field2})THEN
    COLUMN = {field1}
ENDIF

IF ({field1} = {field2})
   THEN COLUMN  =  {field1}
ENDIF
```

WARNING: Extra blanks change text strings, for example "ABC" and "A B C" are different strings.

Logic text can continue on the next line. A backslash (\) is optional at line end.

In previous versions of SAFR, a backslash (\) was required in order to continue a line of logic text on the next line. This backslash is no longer required. The backslash is still allowed for backwards compatibility. This means the following statements are the same:

```
 IF ({FIELD1} >= 2)\
    THEN COLUMN = {FIELD1} ENDIF

 IF ({FIELD1} >= 2)
    THEN COLUMN = {FIELD1} ENDIF
```

The **case of keywords** has no effect.

For example, these IF statements are all the same:

```
 IF ({FIELD1} >= COL.2)
    THEN COLUMN = {FIELD1}
    ELSE COLUMN = COL.2 ENDIF

if ({field1} >= col.2)
   then column = {field1}
   else column = col.2 endif

If ({Field1} >= Col.2)
    Then Column = {Field1}
    Else Column = Col.2 Endif
```

WARNING: Case changes text strings, for example "ABC" and "abc" are different strings.

Rules **Notes**

After a single quote everything on that line is a **comment**

Examples are:

```
 ' Make col the larger of field1 & col.2
IF ({FIELD1} >= COL.2)
   THEN COLUMN = {FIELD1} ENDIF
COLUMN = {field3}  ' Rest is comment (OK).
```

WARNING: comments to the left of keywords result in hiding the keywords, for example:

```
 IF ({field1} = {field2})THEN
    COLUMN = {field1}
' This comment hides keyword:     ENDIF
```

Use **curly brackets { }** to enclose **input fields** or names of metadata from the SAFR Workbench (such as a lookup path, logical file, physical file).

Examples are:

```
{shipping_date}
{product_category}
{CA_Sales_2009_Logical_File}
{Products_Logical_File}
{Products_USA_File}
{Lookup_Sales_to_Product_Category}
```

Keyboard shortcuts for all logic text screens

See topic "**What are the keyboard shortcuts?**"

To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Chapter 2. Logic text: syntax

Syntax: COLUMN and COL.nnn statements

Introduction

COLUMN is always a statement where you set the value of the current column. A COLUMN statement is only allowed in **Extract Column Assignment** and **Format Column Calculations.**

COL.nnn refers to a column value where nnn is the number of that column in that view (starting at one on the leftmost column).

You can **set the value** column in the view using a COL.nnn statement in **Extract Column Assignment**. This is the only logic text where COL.nnn is a statement.

COL.nnn can only appear in an **inquiry** (for example "IF (COL.nnn = ? ") in the format phase. In **Format Column Calculations**, the nnn must be a column number between 1 and the current column (inclusive). In **Format Record Filter**, nnn can be any column in the view.

Syntax: COLUMN and COL.nnn in Extract Column Assignment

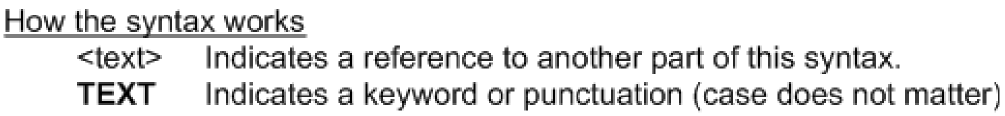
Introduction

COLUMN is always a statement where you set the value of the current column.

COL.nnn refers to a column value where nnn is the number of that column in that view (starting at one on the leftmost column).

You can **set the value** column in the view using a COL.nnn statement in **Extract Column Assignment**. This is the only logic text where COL.nnn is a statement.

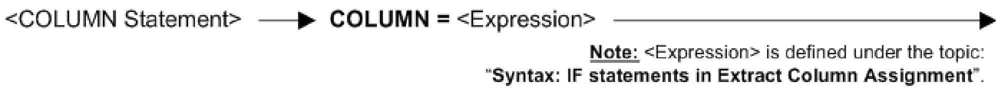
Neither COLUMN or COL.nnn can appear in a condition of an IF statement (or in SELECTIF or SKIPIF).



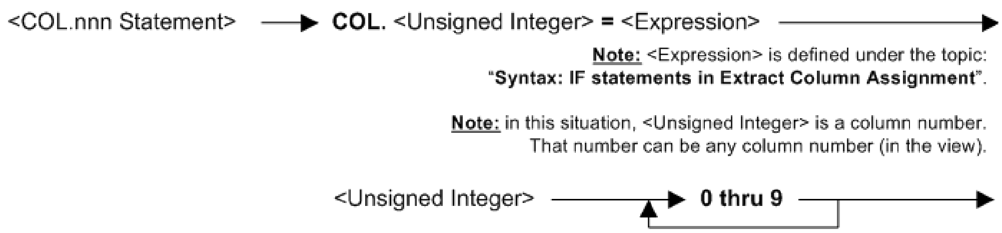
How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax for COLUMN and COL.nnn in Extract Column Assignment



Note: <Expression> is defined under the topic: "**Syntax: IF statements in Extract Column Assignment**".



Note: <Expression> is defined under the topic: "**Syntax: IF statements in Extract Column Assignment**".

Note: in this situation, <Unsigned Integer> is a column number. That number can be any column number (in the view).

Rules for the syntax

- You can set the value of any column in the view using COL.nnn statement. This statement can be placed in the Extract Column Assignment logic text for any column. This means that any column can set the value of any other column.
- You can only set the value of the current column using the COLUMN statement.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: COLUMN and COL.nnn in Extract Column Assignment

In all the following examples, **COLUMN can be replaced by COL.nnn**, for example COL.3. You can set the value of any COL.nnn from any other column. You can create multiple COL.nnn statements in Extract Column Assignment logic text.

Example logic text	Meaning

`COLUMN = ({field2}/{field1}) * 100`
`COL.27 = {field1} * {field26}`
`COL.28 = {field14} + {field1}`
`COL.29 = 0`
`COL.30 = "ABC"`
	Set the current column to field2 divided by field1 all multiplied by 100. Set column 27 to field1 times field26. Set column 28 to field14 plus field1. Set column 29 to zero. Set column 30 to "ABC".

`COLUMN = ALL("-")`	Set the current column to all dashes.
`COLUMN = REPEAT("-", 13)`	Set the current column to 13 dashes.
`COLUMN = "\xFF"`	Set the current column to hexadecimal FF.
`COLUMN = {Lookup1.Field3}`	Set the current column to Field3 found by lookup path Lookup1
`COLUMN = {Lookup1.Field3,field7}`	Set the current column to Field3 found by lookup path Lookup1 using effective date of field7.
`COLUMN = {Lookup1.Field3,;$SYM="A"}`	Set the current column to Field3 found by lookup path Lookup1 using symbol SYM set to "A".
`COLUMN = {Lookup1.Field3,field7;$SYM1=3,$SYM2=0}`	Set the current column to Field3 found by lookup path Lookup1 using effective date of field7 and symbols SYM1 set to 3 and SYM2 set to zero.
`COLUMN = DAYSBETWEEN({BUY_DATE},{SHIP_DATE})`	Set the current column to the days between the transaction date and the shipping date.

Examples: IF with COLUMN and COL.nnn in Extract Column Assignment

In all the following examples, **COLUMN can be replaced by COL.nnn**, for example COL.3. You can set the value of any COL.nnn from any other column. You can create multiple IF statements in Extract Column Assignment logic text. However, you cannot inquire on COL.nnn (for example, IF COL.4 = 0 is not allowed).

Example logic text

Meaning

```
IF ({field1} > 0) THEN
   COLUMN = ({field2}/{field1}) * 100
   COL.27 = {field1} * {field26}
   COL.28 = {field14} + {field1}
ELSE
   COLUMN = 0
   COL.27 = 0
   COL.28 = 0
ENDIF
```

If field1 is greater than zero then set the current column to field2 divided by field1 all multiplied by 100, set column 27 to field1 times field26 and set column 28 to field 14 plus field1. If field1 is not greater than zero then set the current column and columns 27 and 28 to zero.

```
IF (CURRENT({field5}) <> PRIOR({field5}))
   THEN COLUMN = "PRODUCT: "
   ELSE COLUMN = " "
ENDIF
```

If the current record has a different value of field5 from the previous record, set the current column to "PRODUCT: " otherwise set the current column to blank. This assumes the input file is sorted into field5 order.

```
IF ({field5} = "Total")
   THEN COLUMN = ALL("-")
ENDIF
```

If field5 is "Total" then set the current column to all dashes.

```
IF {field6} = ALL("-")
   THEN COLUMN = {field2} + {field3}
ENDIF
```

If field6 is all dashes, then set the current column to a total of fields 2 and 3.

```
IF ({field5} = "Total")
   THEN COLUMN = REPEAT("-", 13)
ENDIF
```

If field5 is "Total" then set the current column to 13 dashes.

```
IF ({field6} = REPEAT("-", 13))
   THEN COLUMN = {field2} + {field3}
ENDIF
```

If field6 is 13 dashes, then set the current column to a total of fields 2 and 3.

```
IF ({field5} = "Total")
   THEN COLUMN = "\xFF"
ENDIF
```

If field5 is "Total" then set the current column to hexadecimal FF.

```
IF ({field6} = "\xFF")
   THEN COLUMN = {field2} + {field3}
ENDIF
```

If field6 is hexadecimal FF, then set the current column to a total of fields 2 and 3.

```
IF ISNOTSPACES({field1})
   THEN COLUMN = {field1}
   ELSE COLUMN = "DEFAULT"
ENDIF
```

If field1 is not spaces then set the current column to field1, otherwise set the current column to "DEFAULT".

<table>
<tr><th>Example logic text</th><th>Meaning</th></tr>
<tr><td><pre>IF ISFOUND({Lookup1})
   THEN COLUMN = {Lookup1}
   ELSE COLUMN = " "
ENDIF</pre></td><td>If the lookup path Lookup1 uses the current record to successfully find a target record, then set the current column to the lookup path field found, otherwise set the current column to blank.</td></tr>
<tr><td><pre> IF ISFOUND({Lookup2;$SYM="A"})
    THEN COLUMN = {Lookup2;$SYM="A"}
    ELSE COLUMN = 0
 ENDIF</pre></td><td>If the lookup path Lookup2 using symbol SYM set to "A", then set the current column to that lookup field, otherwise set the current column to zero.</td></tr>
<tr><td><pre>IF ISNULL({field4}
   THEN COLUMN = "EMPTY"
   ELSE COLUMN = {field4}
ENDIF</pre></td><td>If field4 for the current record contains null values, then set the current column to "EMPTY", otherwise set the current column to field4.</td></tr>
<tr><td><pre>IF ISNUMERIC({field4}
   THEN COLUMN = {field4} * 100
   ELSE COLUMN = 0
ENDIF</pre></td><td>If field4 for the current record is numeric, then set the current column to field4 times 100, otherwise set the current column to zero.</td></tr>
<tr><td><pre>IF (DAYSBETWEEN({BUY_DATE},{SHIP_DATE})
     > 10)
   THEN COLUMN = {SHIP_DATE}
   ELSE COLUMN = {BUY_DATE}
ENDIF</pre></td><td>If there are more than 10 days between the transaction date and the shipping date, then set the current column to the shipping date, otherwise set the current column to the transaction date.</td></tr>
<tr><td><pre>IF ({field1} BEGINS_WITH "BBB")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF</pre></td><td>If field1 begins with characters "BBB" then set the current column to field1, otherwise set the current column to blank.</td></tr>
<tr><td><pre>IF ({field2} CONTAINS "CCC")
   THEN COLUMN = {field2}
   ELSE COLUMN = " "
ENDIF</pre></td><td>If field2 contains characters "CCC" then set the current column to field2, otherwise set the current column to blank.</td></tr>
<tr><td><pre>IF ({field3} ENDS_WITH "EEE")
   THEN COLUMN = {field3}
   ELSE COLUMN = " "
ENDIF</pre></td><td>If field3 ends with characters "EEE" then set the current column to field3, otherwise set the current column to blank.</td></tr>
<tr><td><pre>IF ({field4} MATCHES "...")
   THEN COLUMN = {field4}
   ELSE COLUMN = " "
ENDIF</pre></td><td>If field4 matches characters "..." then set the current column to field4, otherwise set the current column to blank.</td></tr>
<tr><td><pre>IF ({field1} LIKE "MA...")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF</pre></td><td>Select input records where field1 is exactly 5 characters starting with "MA", and skip all other records.</td></tr>
</table>

Example logic text	Meaning

`IF ({field1} LIKE "..VA..")`
`THEN COLUMN = {field1}`
`ELSE COLUMN = " "`
`ENDIF`
	Select input records where field1 is exactly 6 characters with characters 3 and 4 as "VA", and skip all other records.

`IF ({field1} LIKE ".....NA")`
`THEN COLUMN = {field1}`
`ELSE COLUMN = " "`
`ENDIF`
	Select input records where field1 is exactly 6 characters ending in "NA", and skip all other records.

`IF ({field1} LIKE "^BBB*")`
`THEN COLUMN = {field1}`
`ELSE COLUMN = " "`
`ENDIF`
	If field1 begins with characters "BBB" then set the current column to field1, otherwise set the current column to blank.

`IF ({field1} LIKE "*CCC*")`
`THEN COLUMN = {field1}`
`ELSE COLUMN = " "`
`ENDIF`
	If field1 contains characters "CCC" then set the current column to field1, otherwise set the current column to blank.

`IF ({field1} LIKE "*EEE$")`
`THEN COLUMN = {field1}`
`ELSE COLUMN = " "`
`ENDIF`
	If field1 ends with characters "EEE" then set the current column to field1, otherwise set the current column to blank.

`IF ({field1} LIKE "^B*C*E$")`
`THEN COLUMN = {field1}`
`ELSE COLUMN = " "`
`ENDIF`
	If field1 begins with "B", contains "C" and ends with "E" then set the current column to field1, otherwise set the current column to blank.

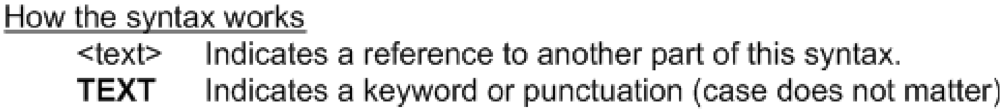
Syntax: COLUMN and COL.nnn in Format Column Calculations

Introduction

COLUMN is always a statement where you set the value of the current column.

COL.nnn refers to a column value where nnn is the number of that column in that view (starting at one on the leftmost column).

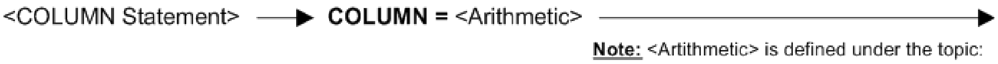
COL.nnn can only appear in an **inquiry** (for example "IF (COL.nnn = ? ") in the format phase. In **Format Column Calculations**, the nnn must be a column number between 1 and the current column (inclusive).



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax for COLUMN in Format Column Calculations



Note: <Artithmetic> is defined under the topic: "**Syntax: IF statements in Format Column Calculations**".

Syntax for inquiry on COL.nnn in Format Column Calculations

See topic "**Syntax: IF statements in Format Column Calculations**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Rules for the syntax

- You can only set the value of the current column using the COLUMN statement.
- COL.nnn can only appear in an **inquiry** (for example "IF (COL.nnn = ? "). The nnn must be a column number between 1 and the current column (inclusive).

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: COLUMN statement in Format Column Calculation

You must set a value of COLUMN.

Example logic text	Meaning
`COLUMN = Col.3 * Col.4`	Set the current column to column 3 times column 4. The current column must be to the right of Column 4.
`COLUMN = "TOTAL"`	Set the current column to "TOTAL".
`COLUMN = ALL("-")`	Set the current column to all dashes.
`COLUMN = REPEAT("-", 13)`	Set the current column to 13 dashes.
`COLUMN = "\xFF"`	Set the current column to hexadecimal FF.

Examples: IF with COLUMN in Format Column Calculation

You must set a value of COLUMN.

Example logic text	Meaning

`IF (Col.7 = 999)`
`   THEN COLUMN = "TOTAL"`
`ENDIF`
	If column 7 is 999 then set the current column to "TOTAL". The current column must be to the right of Column 7.

`IF (Col.7 = 999)`
`   THEN COLUMN = Col.3 * Col.4`
`ENDIF`
	If column 7 is 999 then set the current column to column 3 times column 4. The current column must be to the right of Column 7.

`IF (Col.2 = Col.3)`
`   THEN COLUMN = ALL("-")`
`ENDIF`
	If column 2 equals column 3 then set the current column to all dashes. The current column must be to the right of Column 3.

`IF (Col.2 = Col.3)`
`   THEN COLUMN =`
`        REPEAT("-", 13)`
`ENDIF`
	If column 2 equals column 3 then set the current column to 13 dashes. The current column must be to the right of Column 3.

Example logic text	Meaning

`IF (Col.4 = "14733")`
`   THEN COLUMN = "\xFF"`
`ENDIF`
	If column 4 is "14733" then set the current column to hexadecimal FF. The current column must be to the right of Column 4.

Examples: IF with COL.nnn in Format Column Calculation

In this logic text, you cannot set the value of COL.nnn - you can only enquire on the value of a column between 1 and the current column (inclusive).

Example logic text	Meaning

`IF (Col.7 = "X")`
`   THEN COLUMN = "TOTAL"`
`ENDIF`
	If column 7 is "X" then set the current column to "TOTAL". The current column must be to the right of Column 7.

`IF (Col.4 = "14733")`
`   THEN COLUMN = Col.2`
`ENDIF`
	If column 4 is "14733" then set the current column to column 2. The current column must be to the right of Column 4.

`IF ((Col.4 + Col.5)* COl.6`
`      > 1000)`
`   THEN COLUMN = "\xFF"`
`ENDIF`
	If column 4 and column 5 are added and then multiplied by column 6 and the result is greater than 1000 then set the current column to hexadecimal FF. The current column must be to the right of Column 6.

Syntax: COL.nnn in Format Record Filter

Introduction

COL.nnn refers to a column value where nnn is the number of that column in that view (starting at one on the leftmost column).

COL.nnn can only appear in an **inquiry** (for example "IF (COL.nnn = ? "). In **Format Record Filter**, nnn can be any column in the view.

Syntax for inquiry on COL.nnn in Format Record Filter

See topic "**Syntax: IF statements in Format Record Filter**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: SELECTIF in Format Record Filter

Example logic text	Meaning

`SELECTIF(COL.3 > 1000)`	Select for output only those records with column 3 greater than 1000. Skip all other records. The code at left is a shorthand for:
	`IF (COL.3 > 1000)`
	`   THEN SELECT`
	`ENDIF`

`SELECTIF(COL.2 = "ABC")`	Select for output only those records with column 2 equal to "ABC". Skip all other records.

Example logic text	Meaning
`SELECTIF(NOT COL.2 = "ABC")`	Select those output records with field column 2 not equal to "ABC". Skip all other records. This example gives the same result as: `SKIPIF(COL.2 = "ABC")`

`SELECTIF(COL.2 = "A" OR`
`        COL.2 = "D")`
	Select for output only those records with column 2 equal to "A" or "D". Skip all other records.

`SELECTIF(COL.2 = "A" AND`
`        COL.3 > 10)`
	Select for output only those records with column 2 equal to "A" and column 3 greater than 10. Skip all other records.

`SELECTIF(COL.3 + Col.4 > Col.5)`	Select for output only those records with column 3 plus column 4 is greater than column 5. Skip all other records.
`SELECTIF(NOT COL.6 = ALL("-"))`	Select for output those records with column 6 is not equal to all dashes. Skip all other records. This example gives the same result as: `SKIPIF(COL.6 = ALL("-"))`

`SELECTIF(NOT COL.6 =`
`        REPEAT("-", 13))`
	Select for output those records with column 6 is not equal to 13 dashes. Skip all other records. This example gives the same result as:
	`SKIPIF(COL.6 = REPEAT("-", 13))`

`SELECTIF(NOT COL.6 = "\xFF")`	Select for output those records with column 6 is not equal to hexadecimal FF. Skip all other records. This example gives the same result as: `SKIPIF(COL.6 = "\xFF")`

Examples: SKIPIF in Format Record Filter

Example logic text	Meaning

`SKIPIF(COL.3 > 1000)`	Skip for output those records with column 3 greater than 1000. Select all other records. The code at left is a shorthand for:
	`IF (COL.3 > 1000)`
	`   THEN SKIP`
	`ENDIF`

`SKIPIF(COL.2 = "ABC")`	Skip for output those records with column 2 equal to "ABC". Select all other records.
`SKIPIF(NOT COL.2 = "ABC")`	Skip those output records with field column 2 not equal to "ABC". Select all other records. This example gives the same result as: `SELECTIF(COL.2 = "ABC")`

`SKIPIF(COL.2 = "A" OR`
`       COL.2 = "D")`
	Skip for output those records with column 2 equal to "A" or "D". Select all other records.

Example logic text **Meaning**

```
SKIPIF(COL.2 = "A" AND
       COL.3 > 10)
```

Skip for output those records with column 2 equal to "A" and column 3 greater than 10. Select all other records.

```
SKIPIF(COL.3 + Col.4 > Col.5)
```

Skip for output those records with column 3 plus column 4 is greater than column 5. Select all other records.

```
SKIPIF(COL.6 = ALL("-"))
```

Skip for output those records with column 6 is equal to all dashes. Select all other records.

```
SKIPIF(COL.6 = REPEAT("-", 13))
```

Skip for output those records with column 6 is equal to 13 dashes. Select all other records.

```
SKIPIF(COL.6 = "\xFF")
```

Skip for output those records with column 6 is equal to hexadecimal FF. Select all other records.

Examples: IF with SELECT in Format Record Filter

Example logic text **Meaning**

```
IF (COL.3 > 1000)
   THEN SELECT
ENDIF
```

Select for output only those records with column 3 greater than 1000. Skip all other records. The code at left can also be written as:

```
SELECTIF(COL.3 > 1000)
```

```
IF (COL.2 = "ABC")
   THEN SELECT
ENDIF
```

Select for output only those records with column 2 equal to "ABC". Skip all other records.

```
IF NOT (COL.2 = "ABC")
   THEN SELECT
ENDIF
```

Select those output records with field column 2 not equal to "ABC". Skip all other records. The code at left gives the same result as:

```
SKIPIF(COL.2 = "ABC")
```

```
IF (COL.2 = "A") OR
   (COL.2 = "D")
   THEN SELECT
ENDIF
```

Select for output only those records with column 2 equal to "A" or "D". Skip all other records.

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
   THEN SELECT
ENDIF
```

Select for output only those records with column 2 equal to "A" and column 3 greater than 10. Skip all other records.

```
IF (COL.3 + Col.4 > Col.5 * 100)
   THEN SELECT
ENDIF
```

Select for output only those records with column 3 plus column 4 is greater than column 5 times 100. Skip all other records.

```
IF NOT (COL.6 = ALL("-"))
   THEN SELECT
ENDIF
```

Select for output those records with column 6 is not equal to all dashes. Skip all other records. This example gives the same result as:

```
SKIPIF(COL.6 = ALL("-"))
```

Example logic text **Meaning**

```
IF (COL.6 = REPEAT("-", 13))
   THEN SELECT
ENDIF
```

Select for output those records with column 6 is equal to 13 dashes. Skip all other records.

```
IF (COL.6 = "\xFF")
   THEN SELECT
ENDIF
```

Select for output those records with column 6 is equal to hexadecimal FF. Skip all other records.

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN SELECT
ELSE IF (COL.2 "D")
     THEN SELECT
     ELSE IF (COL.2 "G")
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Select for output those records with column 2 equal to "A" and column 3 greater than 10.

Also select for output those records with column 2 equal to "D".

Also select for output those records with column 2 equal to "G".

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) OR
   (COL.2 = "D") OR
   (COL.2 = "G")
   THEN SELECT
ENDIF
```

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN IF (COL.4 + COL.5
        > COL.6)
     THEN SELECT
     ELSE IF (COL.7 = 0)
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Consider those records with column 2 equal to "A" and column 3 greater than 10.

Of the considered records, select for output those records with column 4 plus column 5 is greater then column 6.

Of the considered records not yet selected, select also for output those records with column 7 equal to zero.

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) AND
   ((COL.4 + COL.5 > COL.6) OR
   (COL.7 = 0))
   THEN SELECT
ENDIF
```

Examples: IF with SKIP in Format Record Filter

Example logic text **Meaning**

```
IF (COL.3 > 1000)
   THEN SKIP
ENDIF
```

Skip for output those records with column 3 greater than 1000. Select all other records. The code at left can also be written as:

```
SKIPIF(COL.3 > 1000)
```

```
IF (COL.2 = "ABC")
   THEN SKIP
ENDIF
```

Skip for output those records with column 2 equal to "ABC". Select all other records.

```
IF NOT (COL.2 = "ABC")
   THEN SKIP
ENDIF
```

Skip those output records with field column 2 not equal to "ABC". Select all other records. The code at left gives the same result as:

```
SELECTIF(COL.2 = "ABC")
```

```
IF (COL.2 = "A") OR
   (COL.2 = "D")
   THEN SKIP
ENDIF
```

Skip for output those records with column 2 equal to "A" or "D". Select all other records.

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
   THEN SKIP
ENDIF
```

Skip for output those records with column 2 equal to "A" and column 3 greater than 10. Select all other records.

```
IF (COL.3 + Col.4
    > Col.5 * 100)
   THEN SKIP
ENDIF
```

Skip for output those records with column 3 plus column 4 is greater than column 5 times 100. Select all other records.

```
IF (COL.6 = ALL("-"))
   THEN SKIP
ENDIF
```

Skip for output those records with column 6 is equal to all dashes. Select all other records. This example gives the same result as:

```
SKIPIF(COL.6 = ALL("-"))
```

```
IF (COL.6 = REPEAT("-", 13))
   THEN SKIP
ENDIF
```

Skip for output those records with column 6 is equal to 13 dashes. Select all other records. This example gives the same result as:

```
SKIPIF(COL.6 = REPEAT("-", 13))
```

```
IF (COL.6 = "\xFF")
   THEN SKIP
ENDIF
```

Skip for output those records with column 6 is equal to hexadecimal FF. Select all other records. This example gives the same result as:

```
SKIPIF(COL.6 = "\xFF")
```

Example logic text **Meaning**

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN SKIP
ELSE IF (COL.2 = "D")
     THEN SKIP
     ELSE IF (COL.2 = "G")
          THEN SKIP
          ENDIF
     ENDIF
ENDIF
```

Skip for output those records with column 2 equal to "A" and column 3 greater than 10.

In addition, skip for output those records with column 2 equal to "D".

In addition, skip for output those records with column 2 equal to "G".

Select all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) OR
   (COL.2 = "D") OR
   (COL.2 = "G")
   THEN SKIP
ENDIF
```

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN IF (COL.4 + COL.5
         > COL.6)
     THEN SKIP
     ELSE IF (COL.7 = 0)
          THEN SKIP
          ENDIF
     ENDIF
ENDIF
```

Consider those records with column 2 equal to "A" and column 3 greater than 10.

Of the considered records, skip for output those records with column 4 plus column 5 is greater then column 6.

Of the considered records not yet skipped, skip also for output those records with column 7 equal to zero.

Select all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) AND
   ((COL.4 + COL.5 > COL.6) OR
    (COL.7 = 0))
   THEN SKIP
ENDIF
```

Syntax: IF statements

Syntax: IF statements in Extract Record Filter

Introduction

IF statements can be part of any logic text. An IF statement allows a condition to control if one or more statements are executed.

IF statements are allowed in all logic text, although the statements that can be called from an IF statement depend on the particular logic text.

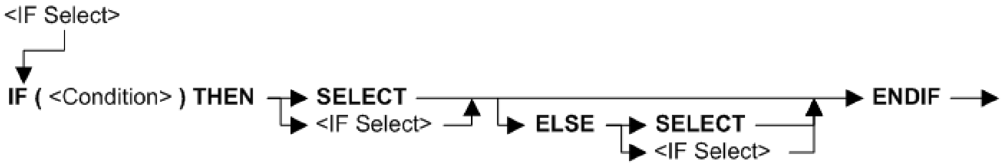
An IF statement can call another IF statement - this is called "nesting" of IF statements, and is allowed in all logic text.

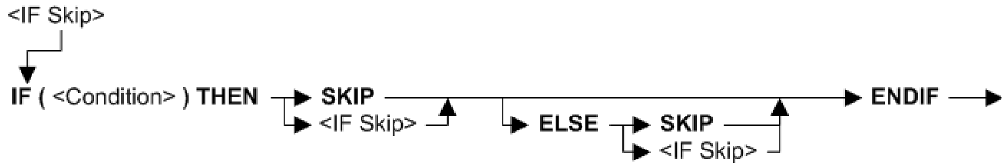
The details of what conditions and what statements are allowed in an IF statement in Extract Record Filter are shown below.

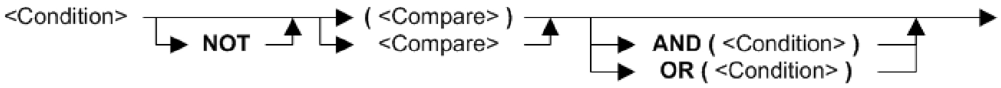
How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

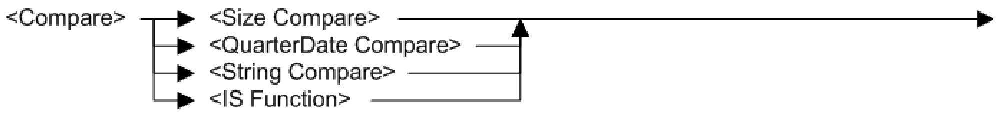
Syntax

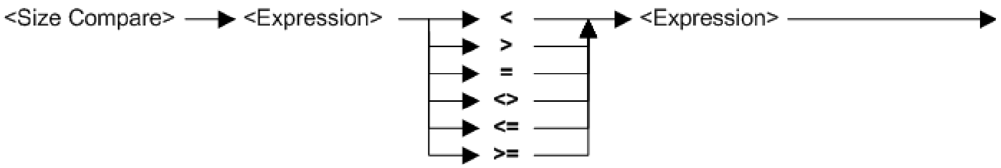


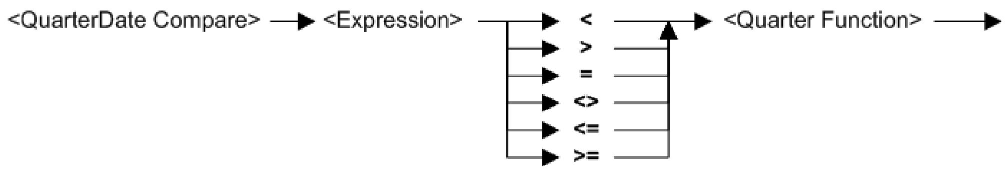


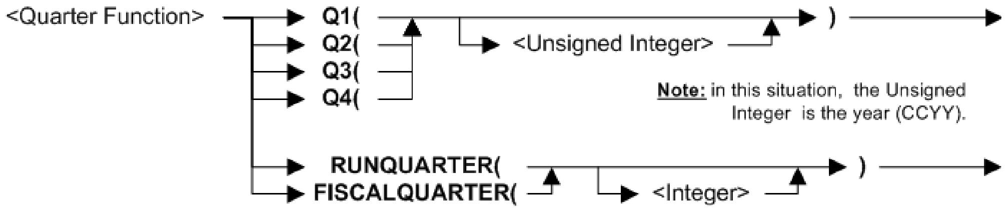


Note: it is recommended to put parentheses "(" ")" around <Compare> text, as long as this does not duplicate parentheses around conditions.



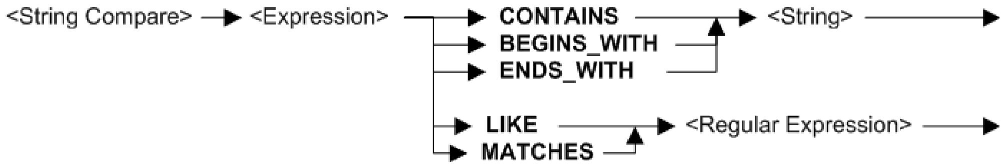


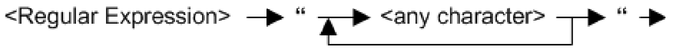




Note: in this situation, the Unsigned Integer is the year (CCYY).

Note: the Integer is the number of quarters to add or subtract from the current value. The default is the current value.





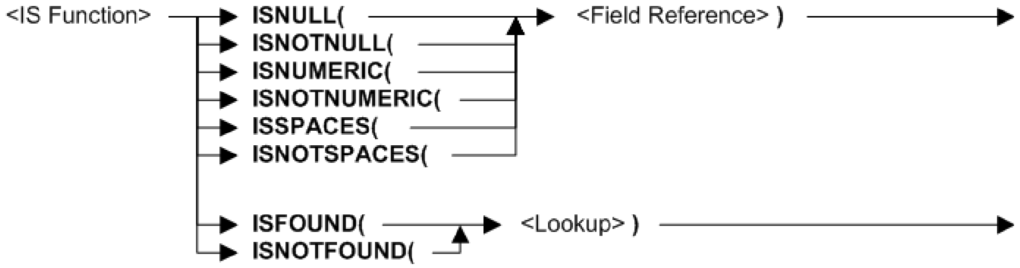
Note: <Regular Expression> uses these meanings:

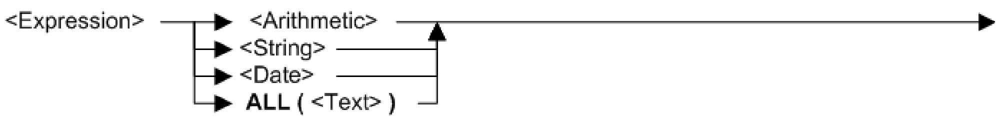
^ means the start of a string

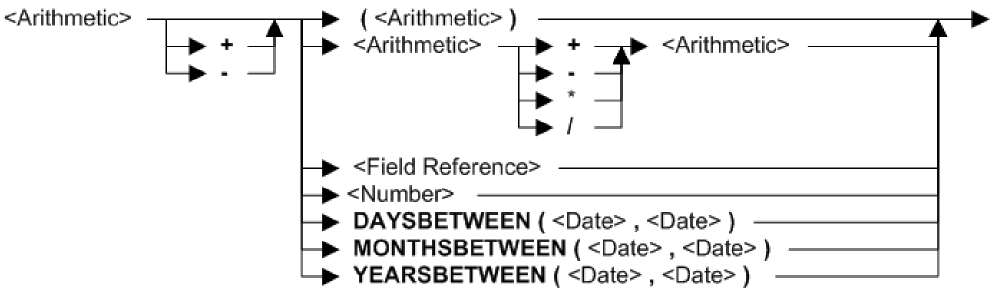
$ means the end of a string

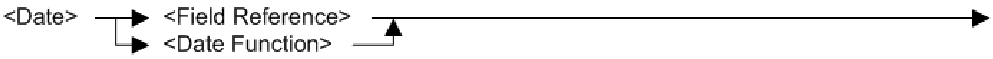
. (fullstop) means any single character

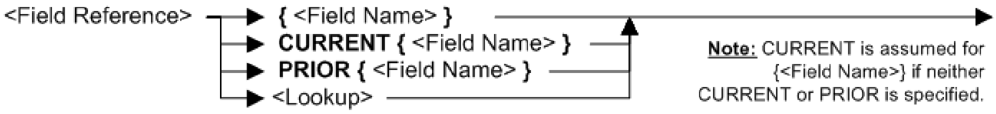
***** (asterisk)means any zero or more characters



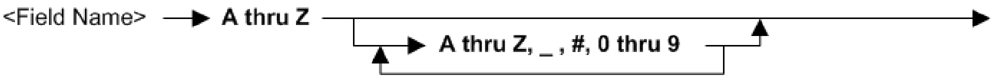


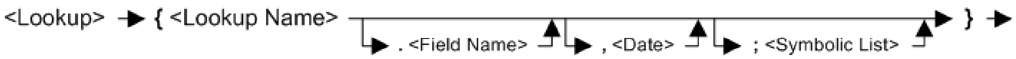

<Arithmetic>
+
-
(<Arithmetic>)
<Arithmetic>
+
-
*
/
<Arithmetic>
<Field Reference>
<Number>
DAYSBETWEEN (<Date> , <Date>)
MONTHSBETWEEN (<Date> , <Date>)
YEARSBETWEEN (<Date> , <Date>)



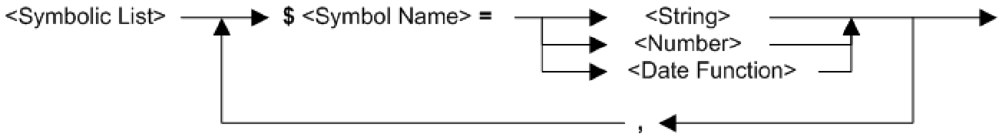


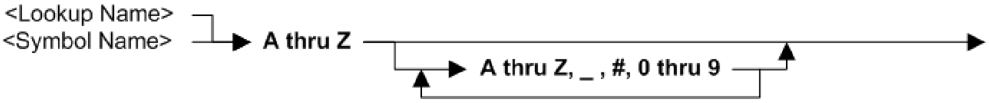
Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.

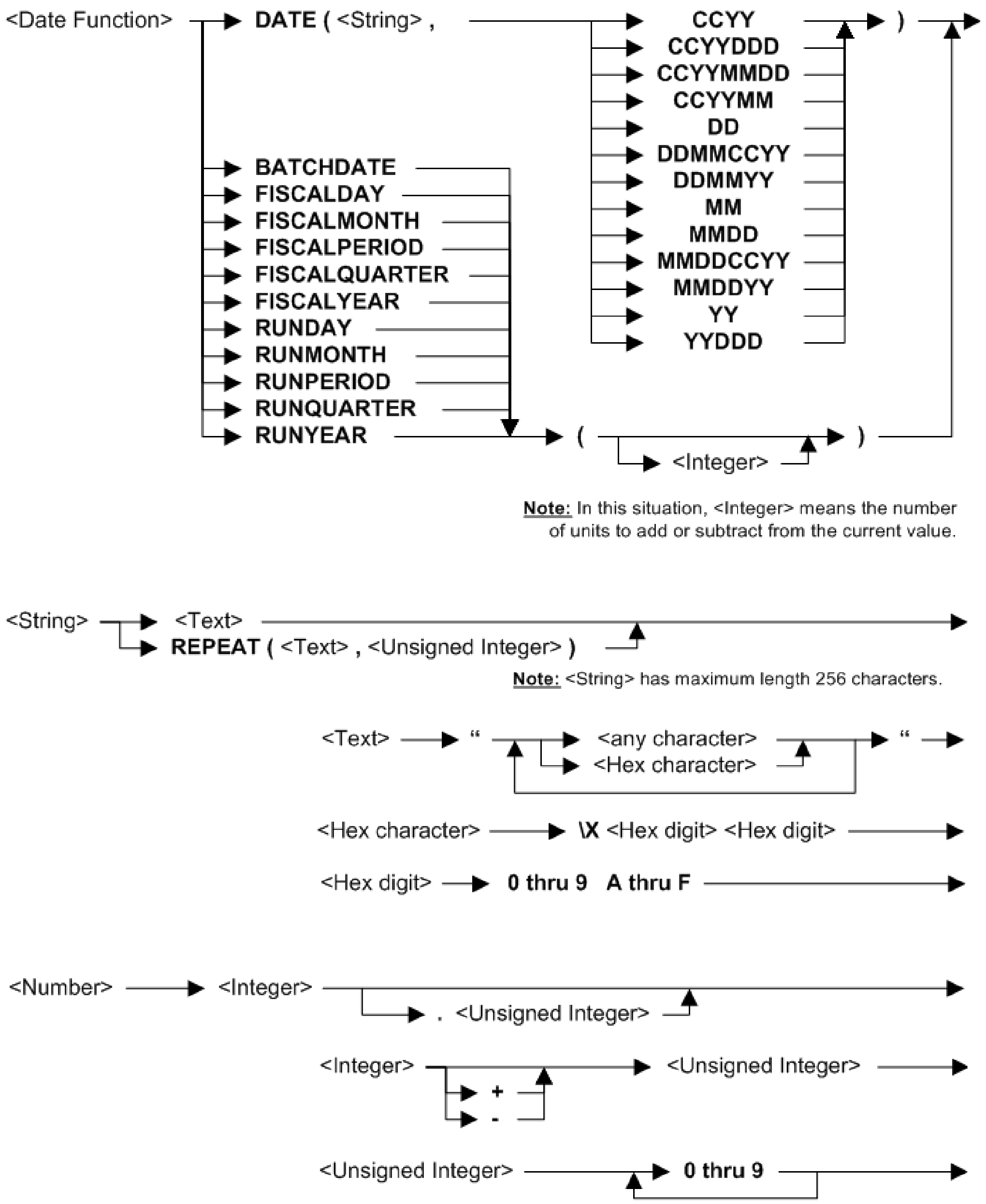




Note: if no date specified, effective date is RUNDAY().







Note: In this situation, <Integer> means the number of units to add or subtract from the current value.

Note: <String> has maximum length 256 characters.

Rules for the syntax

See topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: IF with SELECT in Extract Record Filter

Example logic text	Meaning

`IF (CURRENT({field1}) <>`
`    PRIOR({field1}))`
`   THEN SELECT`
`ENDIF`
	Select only records with unique values for field1. This assumes the input file is sorted into field1 order. This example can also be written:
	`SELECTIF(CURRENT({field1}) <>`
	`         PRIOR({field1}))`

Example logic text	Meaning

`IF ({field3} > 1000)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field3 greater than 1000. Skip all other records. The code at left can also be written as:
	`SELECTIF({field3} > 1000)`

`IF ({field2} = "ABC")`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field2 equal to "ABC". Skip all other records.

`IF NOT ({field2} = "ABC")`
`   THEN SELECT`
`ENDIF`
	Select those output records with field2 not equal to "ABC". Skip all other records. The code at left gives the same result as:
	`SKIPIF({field2} = "ABC")`

`IF ({field2} = "ABC") OR`
`   ({field2} = "DEF")`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field2 equal to "ABC" or "DEF". Skip all other records.

`IF ({field2} = "ABC") AND`
`   ({field3} > 1000)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field2 equal to "ABC" and field3 greater than 1000. Skip all other records.

`IF ({field3} + {field4} >`
`    {field5} * 100)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field3 plus field4 is greater than field5 times 100. Skip all other records.

`IF NOT ({field6} = ALL("-"))`
`   THEN SELECT`
`ENDIF`
	Select for output those records with field6 is not equal to all dashes. Skip all other records. This example gives the same result as:
	`SKIPIF({field6} = ALL("-"))`

`IF ({field6} = REPEAT("-", 13))`
`   THEN SELECT`
`ENDIF`
	Select for output those records with field6 is equal to 13 dashes. Skip all other records.

`IF ({field6} = "\xFF")`
`   THEN SELECT`
`ENDIF`
	Select for output those records with field6 is equal to hexadecimal FF. Skip all other records.

Example logic text

Meaning

```
IF ({field2} = "ABC") AND
   ({field3} > 10)
THEN SELECT
ELSE IF ({field2} = "DEF")
     THEN SELECT
     ELSE IF ({field2} = "GHI")
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Select for output those records with field2 equal to "ABC" and field3 greater than 10.

In addition, select for output those records with field2 equal to "DEF".

In addition, select for output those records with field2 equal to "GHI".

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF ({field2} = "ABC" AND
   {field3} > 10)          OR
   ({field2} = "DEF")      OR
   ({field2} = "GHI")
   THEN SELECT
ENDIF
```

```
IF ({field2} = "ABC") AND
   ({field3} > 10)
THEN IF ({field4} + {field5}
       > {field6})
     THEN SELECT
     ELSE IF ({field7} = 0)
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Consider those records with field2 equal to "ABC" and field3 greater than 10.

Of the considered records, select for output those records with field4 plus field5 is greater then field6.

Of the considered records not yet selected, select also for output those records with field7 equal to zero.

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF ({field2} = "ABC" AND
    {field3} > 10)
                          AND
   (({field4} + {field5}
       > {field6})   OR
   ({field7} = 0))
   THEN SELECT
ENDIF
```

```
IF ISFOUND({Lookup3})
   THEN SELECT
ENDIF
```

Select all input records where the lookup path Lookup3 successfully finds a target record, and skip all other records. This example is the same as:

```
SELECTIF(ISFOUND({Lookup3})
```

Example logic text | **Meaning**

```
IF ISFOUND({Lookup3,field7})
   THEN SELECT
ENDIF
```

Select all input records where the lookup path Lookup3 successfully finds a target record using effective date of field7, and skip all other records. This example is the same as:

```
SELECTIF(ISFOUND({Lookup3,field7})
```

```
IF ISFOUND({Lookup3;$SYM="A"})
   THEN SELECT
ENDIF
```

Select all input records where the lookup path Lookup3 successfully finds a target record using symbol SYM set to "A", and skip all other records. This example is the same as:

```
SELECTIF(ISFOUND({Lookup3;$SYM="A"})
```

```
IF ISFOUND({Lookup3,
            field7;$SYM1=3,$SYM2=0})
   THEN SELECT
ENDIF
```

Select all input records where the lookup path Lookup3 successfully finds a target record using effective date of field7 and symbol SYM1 set to 3 and symbol SYM2 set to zero. Skip all other records. This example is the same as:

```
SELECTIF(ISFOUND({Lookup3,
         field7;$SYM1=3,$SYM2=0})
```

```
IF DAYSBETWEEN({field1},{field2})
     > 7
   THEN SELECT
ENDIF
```

Select only records where there are more than 7 days between field1 and field2, and skip all other records This example can also be written:

```
SELECTIF(DAYSBETWEEN({field1},{field2})
          > 7)
```

```
IF ({field1} BEGINS_WITH "BBB")
   THEN SELECT
ENDIF
```

Select input records where field1 begins with characters "BBB", and skip all other records. This example can be written:

```
SELECTIF({field1} BEGINS_WITH "BBB")
```

It is better to use BEGINS_WITH because the logic text executes faster.

```
IF ({field2} CONTAINS "CCC")
   THEN SELECT
ENDIF
```

Select input records where field2 contains characters "CCC", and skip all other records. This example can be written:

```
SELECTIF({field2} CONTAINS "CCC")
```

It is better to use CONTAINS because the logic text executes faster.

```
IF ({field3} ENDS_WITH "EEE")
   THEN SELECT
ENDIF
```

Select input records where field3 ends with characters "EEE", and skip all other records. This example can be written:

```
SELECTIF({field3} ENDS_WITH "EEE")
```

It is better to use ENDS_WITH because the logic text executes faster.

```
IF ({field4} MATCHES "...")
   THEN SELECT
ENDIF
```

Select input records where field4 matches characters "...", and skip all other records. This example can be written:

```
SELECTIF({field4} MATCHES "...")
```

Example logic text **Meaning**

```
IF ({field1} LIKE "^BBB*")
   THEN SELECT
ENDIF
```

Select input records where field1 begins with characters "BBB", and skip all other records. This example has the same effect as:

```
SELECTIF({field1} BEGINS_WITH "BBB")
```

It is better to use BEGINS_WITH because the logic text executes faster.

```
IF ({field1} LIKE "*CCC*")
   THEN SELECT
ENDIF
```

Select input records where field1 contains characters "CCC", and skip all other records. This example has the same effect as:

```
SELECTIF({field1} CONTAINS "CCC")
```

It is better to use CONTAINS because the logic text executes faster.

```
IF ({field1} LIKE "*EEE$")
   THEN SELECT
ENDIF
```

Select input records where field1 ends with characters "EEE", and skip all other records. This example has the same effect as:

```
SELECTIF({field1} ENDS_WITH "EEE")
```

It is better to use ENDS_WITH because the logic text executes faster.

```
IF ({field1} LIKE "^B*C*E$")
   THEN SELECT
ENDIF
```

Select input records where field1 begins with "B", contains "C" and ends with "E", and skip all other records.

Examples: IF with SKIP in Extract Record Filter

Example logic text **Meaning**

```
IF (CURRENT({field1}) =
    PRIOR({field1}))
   THEN SKIP
ENDIF
```

Skip records where field1 is the same as the previous record. This assumes the input file is sorted into field1 order. This example can also be written:

```
SKIPIF(CURRENT({field1}) =
       PRIOR({field1}))
```

```
IF ({field3} > 1000)
   THEN SKIP
ENDIF
```

Skip for output those records with field3 greater than 1000. Select all other records. The code at left can also be written as:

```
SKIPIF({field3} > 1000)
```

```
IF ({field2} = "ABC")
   THEN SKIP
ENDIF
```

Skip for output those records with field2 equal to "ABC". Select all other records.

```
IF NOT ({field2} = "ABC")
   THEN SKIP
ENDIF
```

Skip those output records with field2 not equal to "ABC". Select all other records. The code at left gives the same result as:

```
SELECTIF({field2} = "ABC")
```

Example logic text **Meaning**

```
IF ({field2} = "A") OR
   ({field2} = "D")
   THEN SKIP
ENDIF
```

Skip for output those records with field2 equal to "A" or "D". Select all other records.

```
IF ({field2} = "A") AND
   ({field3} > 10
   THEN SKIP
ENDIF
```

Skip for output those records with field2 equal to "A" and field3 greater than 10. Select all other records.

```
IF ({field3} + {field4}
    > {field5})
   THEN SKIP
ENDIF
```

Skip for output those records with field3 plus field4 is greater than field5. Select all other records.

```
IF ({field6} = ALL("-"))
   THEN SKIP
ENDIF
```

Skip for output those records with field6 is equal to all dashes. Select all other records. This example gives the same result as:

```
SKIPIF({field6} = ALL("-"))
```

```
IF ({field6} = REPEAT("-", 13))
   THEN SKIP
ENDIF
```

Skip for output those records with field6 is equal to 13 dashes. Select all other records. This example gives the same result as:

```
SKIPIF({field6} = REPEAT("-", 13))
```

```
IF ({field6} = "\xFF")
   THEN SKIP
ENDIF
```

Skip for output those records with field6 is equal to hexadecimal FF. Select all other records. This example gives the same result as:

```
SKIPIF({field6} = "\xFF")
```

```
IF ({field2} = "A") AND
   ({field3} > 10)
THEN SKIP
ELSE IF ({field2} = "D")
     THEN SKIP
     ELSE IF ({field2} = "G")
          THEN SKIP
          ENDIF
     ENDIF
ENDIF
```

Skip for output those records with field2 equal to "A" and field3 greater than 10.

In addition, skip for output those records with field2 equal to "D".

In addition, skip for output those records with field2 equal to "G".

Select all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF ({field2} = "A" AND
    {field3} > 10)         OR
   ({field2} = "D") OR
   ({field2} = "G")
   THEN SKIP
ENDIF
```

Example logic text	Meaning

`IF ISNOTFOUND({Lookup4})`
`THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup4 does not successfully find a target record, and select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup4})`

`IF ISNOTFOUND({Lookup4,field7})`
`THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup4 does not successfully find a target record using effective date of field7, and select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup4,field7})`

`IF ISNOTFOUND({Lookup4;$SYM="A"})`
`THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup4 does not successfully find a target record using symbol SYM set to "A", and select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup4;$SYM="A"})`

`IF ISNOTFOUND`
`({Lookup4,field7;$SYM1=3,$SYM2=0})`
`THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup4 does not successfully find a target record using effective date of field7 and symbol SYM1 set to 3 and symbol SYM2 set to zero. Select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup4,`
	`field7;$SYM1=3,$SYM2=0})`

`IF DAYSBETWEEN({field1},{field2})`
`> 7`
`THEN SKIP`
`ENDIF`
	Skip records where there are more than 7 days between field1 and field2, and select all other records. This example can also be written:
	`SKIPIF(DAYSBETWEEN({field1},{field2})`
	`> 7)`

`IF ({field1} BEGINS_WITH "BBB")`
`THEN SKIP`
`ENDIF`
	Skip input records where field1 begins with characters "BBB", and select all other records. This example can be written:
	`SKIPIF({field1} BEGINS_WITH "BBB")`
	It is better to use BEGINS_WITH because the logic text executes faster.

`IF ({field2} CONTAINS "CCC")`
`THEN SKIP`
`ENDIF`
	Skip input records where field2 contains characters "CCC", and select all other records. This example can be written:
	`SKIPIF({field2} CONTAINS "CCC")`
	It is better to use CONTAINS because the logic text executes faster.

`IF ({field3} ENDS_WITH "EEE")`
`THEN SKIP`
`ENDIF`
	Skip input records where field3 ends with characters "EEE", and select all other records. This example can be written:
	`SKIPIF({field3} ENDS_WITH "EEE")`
	It is better to use ENDS_WITH because the logic text executes faster.

Example logic text	Meaning

`IF ({field4} MATCHES "...")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field4 matches characters "...", and select all other records. This example can be written:
	`SKIPIF({field4} MATCHES "...")`

`IF ({field1} LIKE "^BBB*")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 begins with characters "BBB", and select all other records. This example has the same effect as:
	`SKIPIF({field1} BEGINS_WITH "BBB")`
	It is better to use BEGINS_WITH because the logic text executes faster.

`IF ({field1} LIKE "*CCC*")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 contains characters "CCC", and select all other records. This example has the same effect as:
	`SKIPIF({field1} CONTAINS "CCC")`
	It is better to use CONTAINS because the logic text executes faster.

`IF ({field1} LIKE "*EEE$")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 ends with characters "EEE", and select all other records. This example has the same effect as:
	`SKIPIF({field1} ENDS_WITH "EEE")`
	It is better to use ENDS_WITH because the logic text executes faster.

`IF ({field1} LIKE "^B*C*E$")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 begins with "B", contains "C" and ends with "E", and select all other records.

Syntax: IF statements in Extract Column Assignment

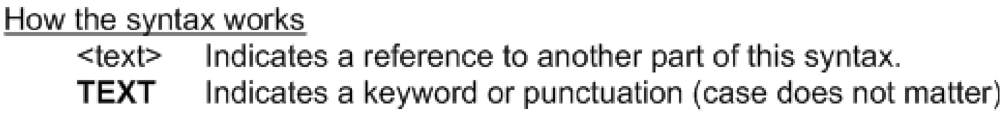
Introduction

IF statements can be part of any logic text. An IF statement allows a condition to control if one or more statements are executed.

IF statements are allowed in all logic text, although the statements that can be called from an IF statement depend on the particular logic text.

An IF statement can call another IF statement - this is called "nesting" of IF statements, and is allowed in all logic text.

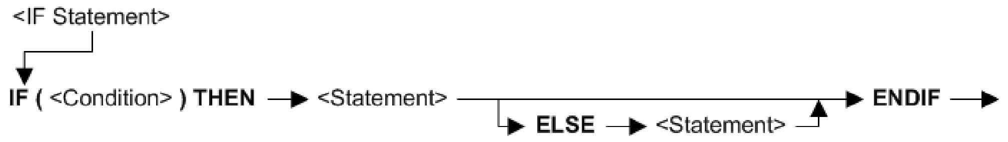
The details of what conditions and what statements are allowed in an IF statement in Extract Column Assignment are shown below.



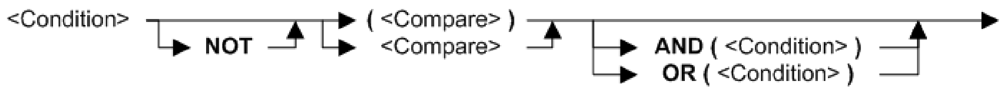
How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

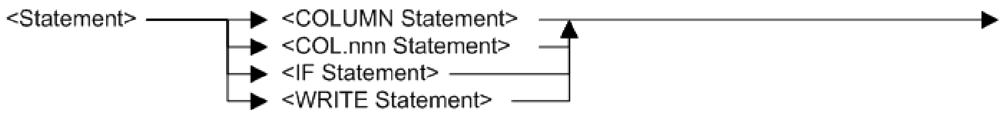
Syntax

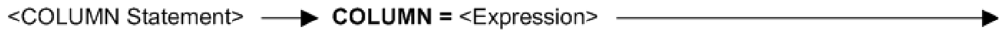


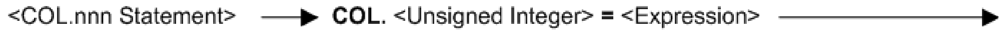
Note: it is recommended to put parentheses "(" ")" around conditions.



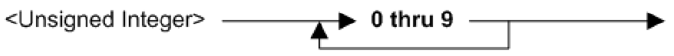
Note: it is recommended to put parentheses "(" ")" around <Compare> text, as long as this does not duplicate parentheses around conditions.

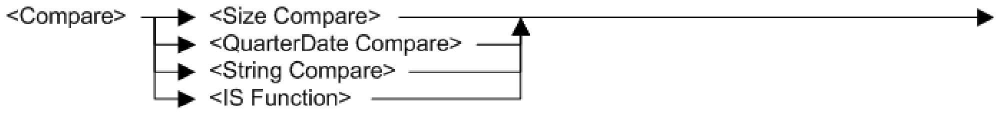


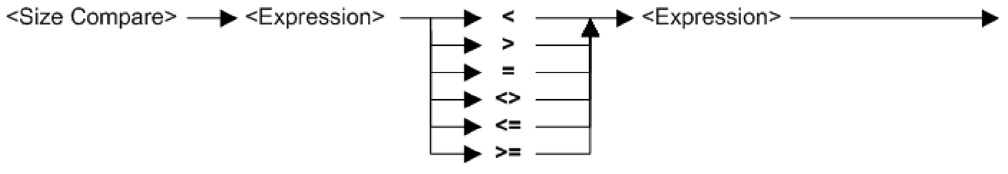


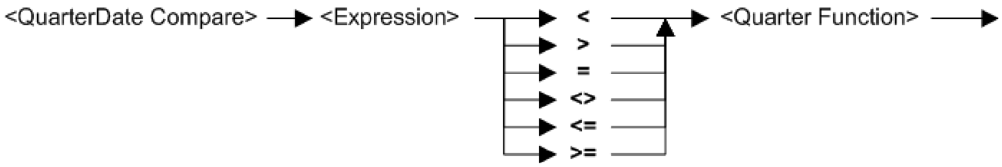


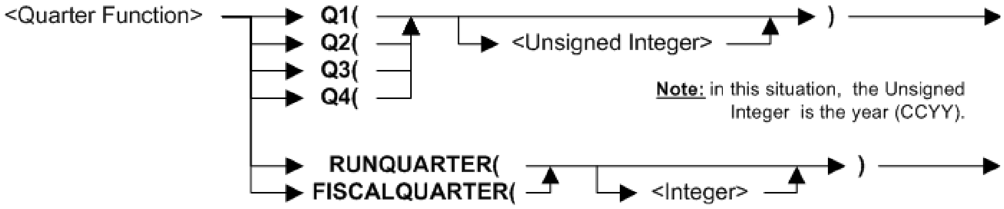
Note: in this situation, <Unsigned Integer> is a column number. That number can be any column number (in the view).





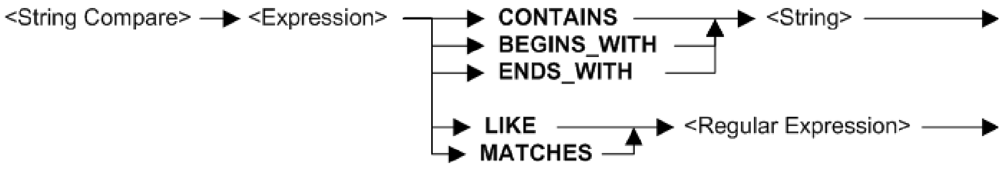


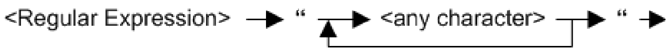




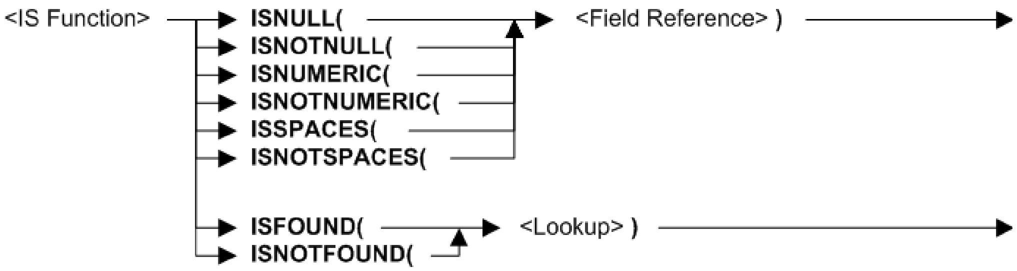
Note: in this situation, the Unsigned Integer is the year (CCYY).

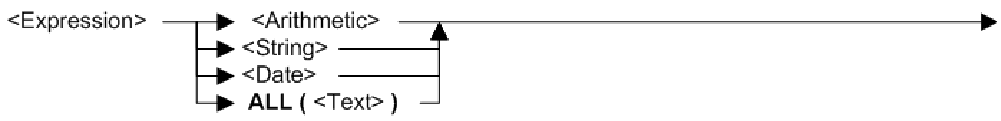
Note: the Integer is the number of quarters to add or subtract from the current value. The default is the current value.

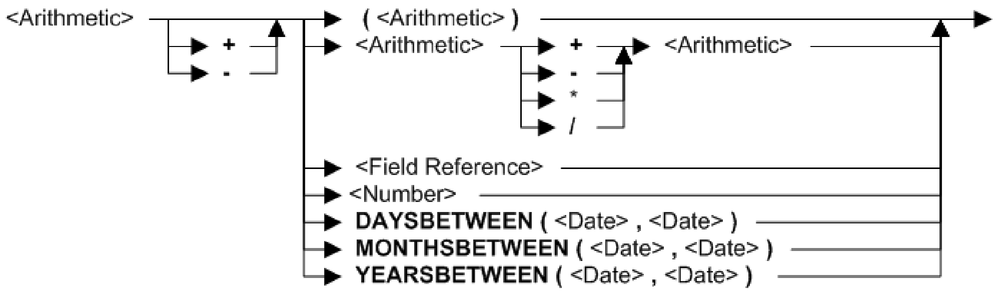




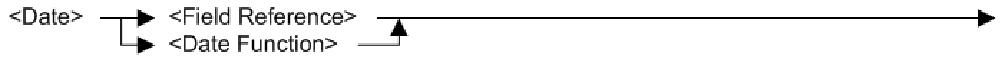
Note: <Regular Expression> uses these meanings:
^ means the start of a string
$ means the end of a string
. (fullstop) means any single character
* (asterisk)means any zero or more characters

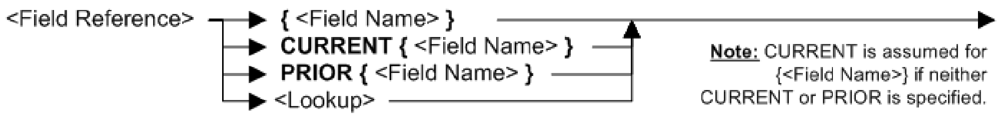




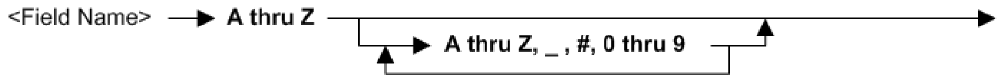


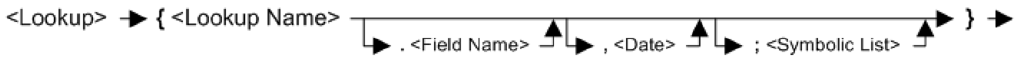
<Arithmetic>
+
-
(<Arithmetic>)
<Arithmetic>
+
-
*
/
<Arithmetic>
<Field Reference>
<Number>
DAYSBETWEEN (<Date> , <Date>)
MONTHSBETWEEN (<Date> , <Date>)
YEARSBETWEEN (<Date> , <Date>)



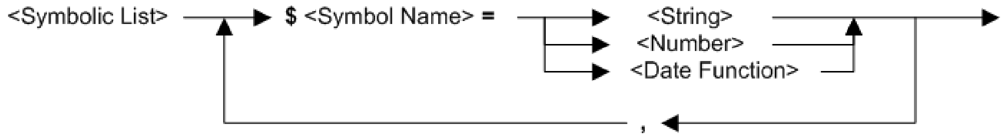


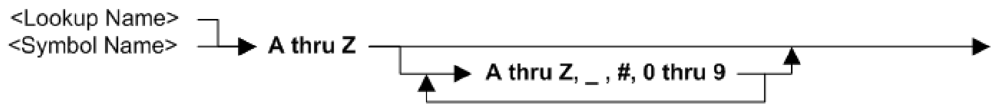
Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.

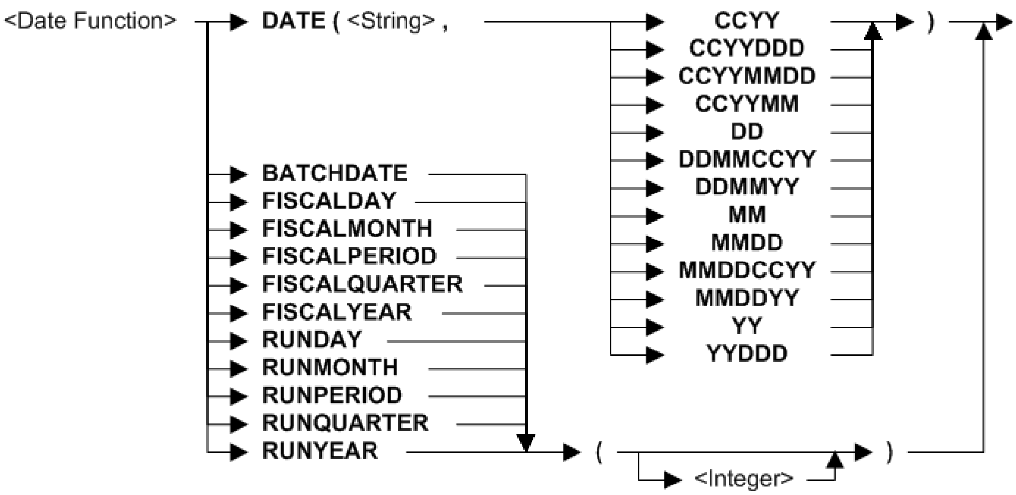




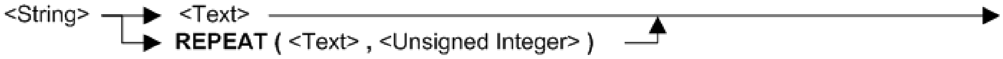
Note: if no date specified, effective date is RUNDAY().



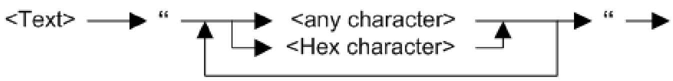


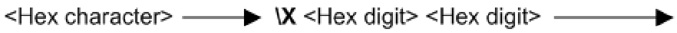


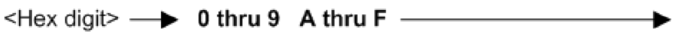
Note: In this situation, <Integer> means the number of units to add or subtract from the current value.

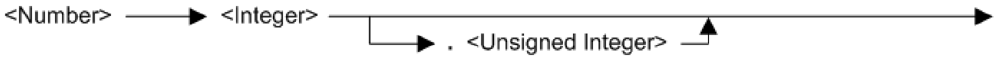


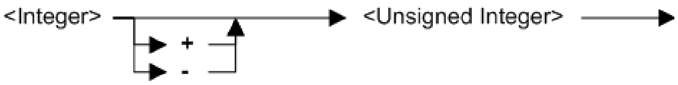
Note: <String> has maximum length 256 characters.

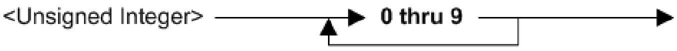












Rules for the syntax

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: IF with COLUMN and COL.nnn in Extract Column Assignment

In all the following examples, **COLUMN can be replaced by COL.nnn**, for example COL.3. You can set the value of any COL.nnn from any other column. You can create multiple IF statements in Extract Column Assignment logic text. However, you cannot inquire on COL.nnn (for example, IF COL.4 = 0 is not allowed).

Example logic text	Meaning

`IF ({field1} > 0) THEN`
`   COLUMN = ({field2}/{field1}) * 100`
`   COL.27 = {field1} * {field26}`
`   COL.28 = {field14} + {field1}`
`ELSE`
`   COLUMN = 0`
`   COL.27 = 0`
`   COL.28 = 0`
`ENDIF`
	If field1 is greater than zero then set the current column to field2 divided by field1 all multiplied by 100, set column 27 to field1 times field26 and set column 28 to field 14 plus field1. If field1 is not greater than zero then set the current column and columns 27 and 28 to zero.

`IF (CURRENT({field5}) <> PRIOR({field5}))`
`   THEN COLUMN = "PRODUCT: "`
`   ELSE COLUMN = " "`
`ENDIF`
	If the current record has a different value of field5 from the previous record, set the current column to "PRODUCT: " otherwise set the current column to blank. This assumes the input file is sorted into field5 order.

`IF ({field5} = "Total")`
`   THEN COLUMN = ALL("-")`
`ENDIF`
	If field5 is "Total" then set the current column to all dashes.

`IF {field6} = ALL("-")`
`   THEN COLUMN = {field2} + {field3}`
`ENDIF`
	If field6 is all dashes, then set the current column to a total of fields 2 and 3.

`IF ({field5} = "Total")`
`   THEN COLUMN = REPEAT("-", 13)`
`ENDIF`
	If field5 is "Total" then set the current column to 13 dashes.

`IF ({field6} = REPEAT("-", 13))`
`   THEN COLUMN = {field2} + {field3}`
`ENDIF`
	If field6 is 13 dashes, then set the current column to a total of fields 2 and 3.

`IF ({field5} = "Total")`
`   THEN COLUMN = "\xFF"`
`ENDIF`
	If field5 is "Total" then set the current column to hexadecimal FF.

`IF ({field6} = "\xFF")`
`   THEN COLUMN = {field2} + {field3}`
`ENDIF`
	If field6 is hexadecimal FF, then set the current column to a total of fields 2 and 3.

`IF ISNOTSPACES({field1})`
`   THEN COLUMN = {field1}`
`   ELSE COLUMN = "DEFAULT"`
`ENDIF`
	If field1 is not spaces then set the current column to field1, otherwise set the current column to "DEFAULT".

`IF ISFOUND({Lookup1})`
`   THEN COLUMN = {Lookup1}`
`   ELSE COLUMN = " "`
`ENDIF`
	If the lookup path Lookup1 uses the current record to successfully find a target record, then set the current column to the lookup path field found, otherwise set the current column to blank.

Example logic text	Meaning

IF ISFOUND({Lookup2;$SYM="A"})
THEN COLUMN = {Lookup2;$SYM="A"}
ELSE COLUMN = 0
ENDIF
	If the lookup path Lookup2 using symbol SYM set to "A", then set the current column to that lookup field, otherwise set the current column to zero.

IF ISNULL({field4}
THEN COLUMN = "EMPTY"
ELSE COLUMN = {field4}
ENDIF
	If field4 for the current record contains null values, then set the current column to "EMPTY", otherwise set the current column to field4.

IF ISNUMERIC({field4}
THEN COLUMN = {field4} * 100
ELSE COLUMN = 0
ENDIF
	If field4 for the current record is numeric, then set the current column to field4 times 100, otherwise set the current column to zero.

IF (DAYSBETWEEN({BUY_DATE},{SHIP_DATE})
> 10)
THEN COLUMN = {SHIP_DATE}
ELSE COLUMN = {BUY_DATE}
ENDIF
	If there are more than 10 days between the transaction date and the shipping date, then set the current column to the shipping date, otherwise set the current column to the transaction date.

IF ({field1} BEGINS_WITH "BBB")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	If field1 begins with characters "BBB" then set the current column to field1, otherwise set the current column to blank.

IF ({field2} CONTAINS "CCC")
THEN COLUMN = {field2}
ELSE COLUMN = " "
ENDIF
	If field2 contains characters "CCC" then set the current column to field2, otherwise set the current column to blank.

IF ({field3} ENDS_WITH "EEE")
THEN COLUMN = {field3}
ELSE COLUMN = " "
ENDIF
	If field3 ends with characters "EEE" then set the current column to field3, otherwise set the current column to blank.

IF ({field4} MATCHES "...")
THEN COLUMN = {field4}
ELSE COLUMN = " "
ENDIF
	If field4 matches characters "..." then set the current column to field4, otherwise set the current column to blank.

IF ({field1} LIKE "MA...")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	Select input records where field1 is exactly 5 characters starting with "MA", and skip all other records.

IF ({field1} LIKE "..VA..")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	Select input records where field1 is exactly 6 characters with characters 3 and 4 as "VA", and skip all other records.

IF ({field1} LIKE ".....NA")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	Select input records where field1 is exactly 6 characters ending in "NA", and skip all other records.

Example logic text | **Meaning**

```
IF ({field1} LIKE "^BBB*")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

If field1 begins with characters "BBB" then set the current column to field1, otherwise set the current column to blank.

```
IF ({field1} LIKE "*CCC*")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

If field1 contains characters "CCC" then set the current column to field1, otherwise set the current column to blank.

```
IF ({field1} LIKE "*EEE$")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

If field1 ends with characters "EEE" then set the current column to field1, otherwise set the current column to blank.

```
IF ({field1} LIKE "^B*C*E$")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

If field1 begins with "B", contains "C" and ends with "E" then set the current column to field1, otherwise set the current column to blank.

Examples: IF with WRITE in Extract Column Assignment

Example logic text | **Meaning**

```
IF (ISNUMERIC({field4}) AND
   ({field5} > {field6} * 10) AND
   (ISNOTSPACES{field7}
   THEN WRITE (SOURCE=DATA,
               USEREXIT={DB2_Update})
ENDIF
```

If field4 is numeric and field5 is greater than field6 times 10 and field7 is not spaces, then call the user-exit routine DB2_Update for the columns up to the current point. This effectively does a writes to a DB2 table the columns in that record up to the current point.

```
IF (ISNOTNULL({field3}) AND
   ({field2} = {field1} + {field5}
   THEN WRITE (SOURCE=INPUT,
               DEST=FILE=
                  {LogicalFile3})
ENDIF
```

If field3 is not nulls and field2 equals field1 plus field 5 then write the entire input record to LogicalFile3. All columns in the input record are included, no matter what column contains this logic text.

```
IF (DAYSBETWEEN({field12},{field15})
     > 10) AND
   (ISFOUND({Lookup3;$SYM="A"}))
   THEN WRITE (SOURCE=VIEW,
               DEST=EXT=03)
ENDIF
```

If field12 and field15 are more than 10 days apart and the lookup path Lookup3 works with symbol SYM set to "A", then write the columns up to the current point to extract work file (EXT) 3. This assumes that the control record for this environment has a Maximum Extract File Number that is at least 3, or any overwrite of the VDP has also set this parameter to at least 3.

Syntax: IF statements in Format Column Calculations

Introduction

IF statements can be part of any logic text. An IF statement allows a condition to control if one or more statements are executed.

IF statements are allowed in all logic text, although the statements that can be called from an IF statement depend on the particular logic text.

An IF statement can call another IF statement - this is called "nesting" of IF statements, and is allowed in all logic text.

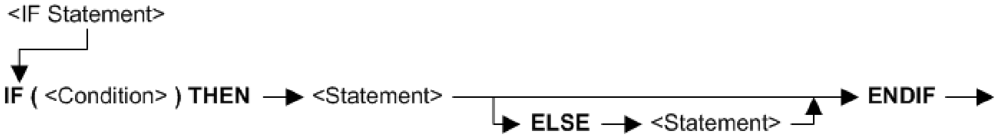
The details of what conditions and what statements are allowed in an IF statement in Format Column Calculations are shown below.

How the syntax works

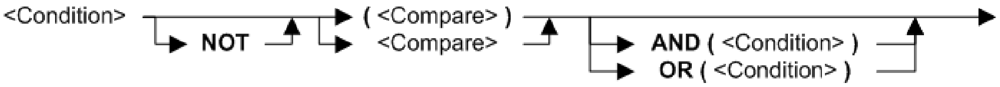
<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

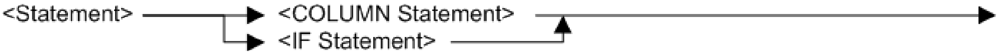
Syntax

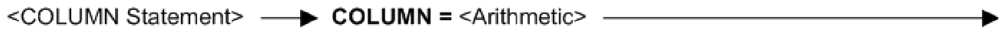


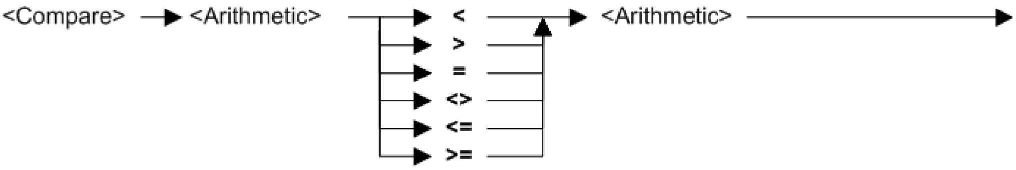
Note: it is recommended to put parentheses "(" ")" around conditions.

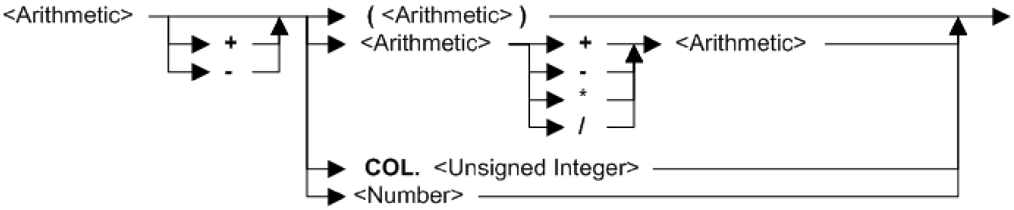


Note: it is recommended to put parentheses "(" ")" around <Compare> text, as long as this does not duplicate parentheses around conditions.

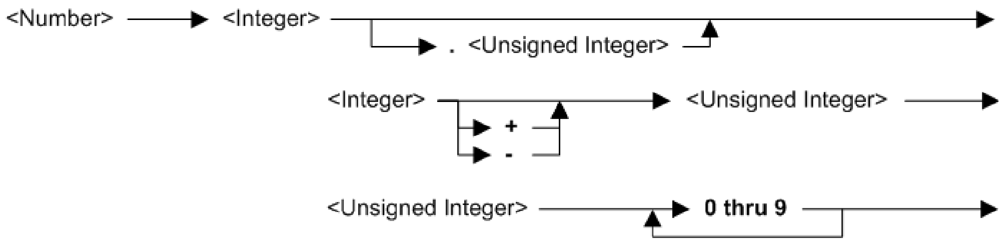








Note: in this situation, <Unsigned Integer> is a column number. That number must be between 1 and the current column number (inclusive).



Rules for the syntax

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: IF with COLUMN in Format Column Calculation

You must set a value of COLUMN.

Example logic text	Meaning

`IF (Col.7 = 999)`
`   THEN COLUMN = "TOTAL"`
`ENDIF`
	If column 7 is 999 then set the current column to "TOTAL". The current column must be to the right of Column 7.

`IF (Col.7 = 999)`
`   THEN COLUMN = Col.3 * Col.4`
`ENDIF`
	If column 7 is 999 then set the current column to column 3 times column 4. The current column must be to the right of Column 7.

`IF (Col.2 = Col.3)`
`   THEN COLUMN = ALL("-")`
`ENDIF`
	If column 2 equals column 3 then set the current column to all dashes. The current column must be to the right of Column 3.

`IF (Col.2 = Col.3)`
`   THEN COLUMN =`
`        REPEAT("-", 13)`
`ENDIF`
	If column 2 equals column 3 then set the current column to 13 dashes. The current column must be to the right of Column 3.

`IF (Col.4 = "14733")`
`   THEN COLUMN = "\xFF"`
`ENDIF`
	If column 4 is "14733" then set the current column to hexadecimal FF. The current column must be to the right of Column 4.

Examples: IF with COL.nnn in Format Column Calculation

In this logic text, you cannot set the value of COL.nnn - you can only enquire on the value of a column between 1 and the current column (inclusive).

Example logic text	Meaning

`IF (Col.7 = "X")`
`   THEN COLUMN = "TOTAL"`
`ENDIF`
	If column 7 is "X" then set the current column to "TOTAL". The current column must be to the right of Column 7.

`IF (Col.4 = "14733")`
`   THEN COLUMN = Col.2`
`ENDIF`
	If column 4 is "14733" then set the current column to column 2. The current column must be to the right of Column 4.

Example logic text	Meaning

`IF ((Col.4 + Col.5)* COl.6`
`    > 1000)`
`   THEN COLUMN = "\xFF"`
`ENDIF`
	If column 4 and column 5 are added and then multiplied by column 6 and the result is greater than 1000 then set the current column to hexadecimal FF. The current column must be to the right of Column 6.

Syntax: IF statements in Format Record Filter

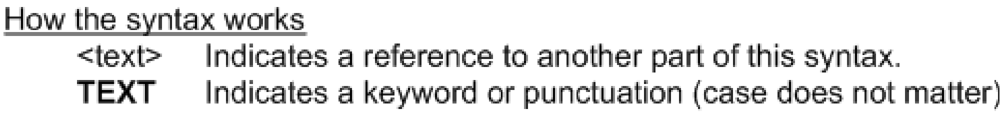
Introduction

IF statements can be part of any logic text. An IF statement allows a condition to control if one or more statements are executed.

IF statements are allowed in all logic text, although the statements that can be called from an IF statement depend on the particular logic text.

An IF statement can call another IF statement - this is called "nesting" of IF statements, and is allowed in all logic text.

The details of what conditions and what statements are allowed in an IF statement in Format Record Filter are shown below.

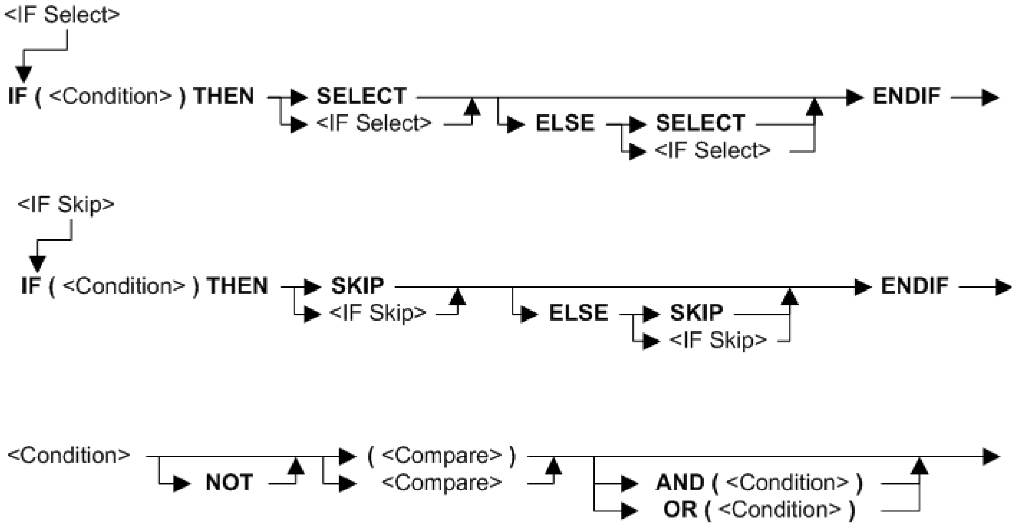


How the syntax works

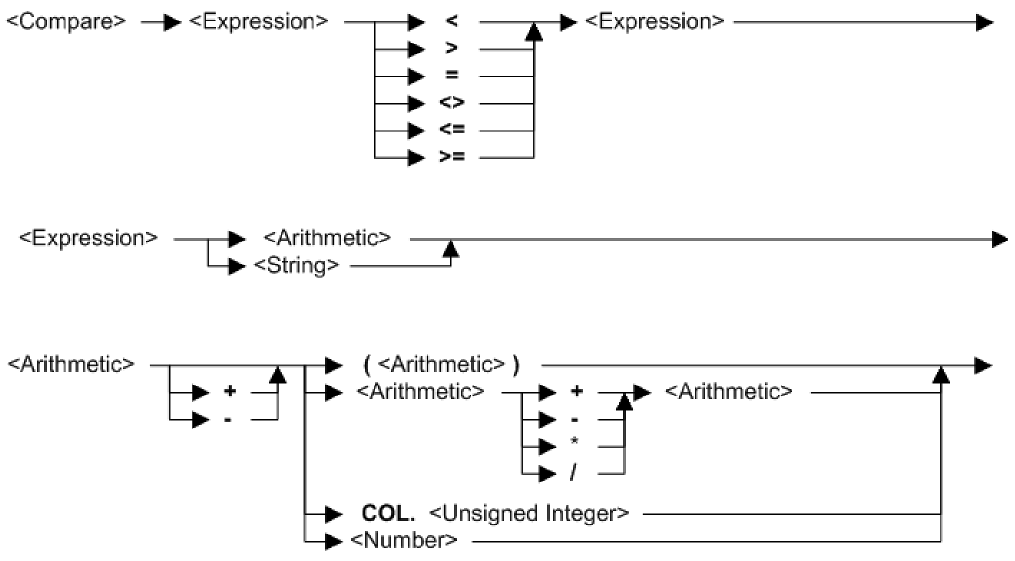
<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

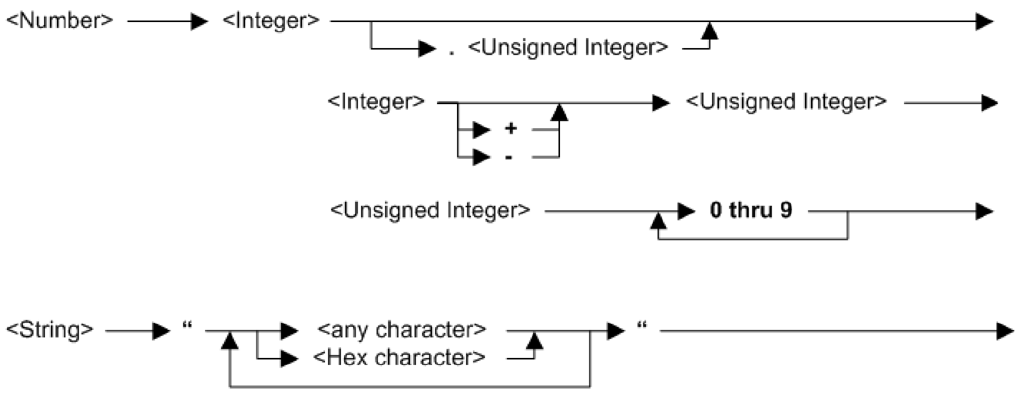
Syntax



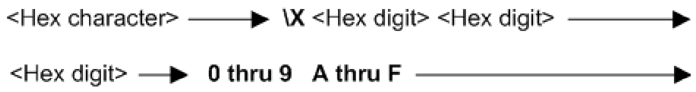
Note: it is recommended to put parentheses "(" ")" around <Compare> text,
as long as this does not duplicate parentheses around conditions.



Note: in this situation, <Unsigned Integer> is a column number. That number can be any column number (in the view).



Note: <String> has maximum length 256 characters.



Rules for the syntax

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: IF with SELECT in Format Record Filter

Example logic text	Meaning

`IF (COL.3 > 1000)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with column 3 greater than 1000. Skip all other records. The code at left can also be written as:
	`SELECTIF(COL.3 > 1000)`

Example logic text	Meaning

`IF (COL.2 = "ABC")`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with column 2 equal to "ABC". Skip all other records.

`IF NOT (COL.2 = "ABC")`
`   THEN SELECT`
`ENDIF`
	Select those output records with field column 2 not equal to "ABC". Skip all other records. The code at left gives the same result as:
	`SKIPIF(COL.2 = "ABC")`

`IF (COL.2 = "A") OR`
`   (COL.2 = "D")`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with column 2 equal to "A" or "D". Skip all other records.

`IF (COL.2 = "A") AND`
`   (COL.3 > 10)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with column 2 equal to "A" and column 3 greater than 10. Skip all other records.

`IF (COL.3 + Col.4 > Col.5 * 100)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with column 3 plus column 4 is greater than column 5 times 100. Skip all other records.

`IF NOT (COL.6 = ALL("-"))`
`   THEN SELECT`
`ENDIF`
	Select for output those records with column 6 is not equal to all dashes. Skip all other records. This example gives the same result as:
	`SKIPIF(COL.6 = ALL("-"))`

`IF (COL.6 = REPEAT("-", 13))`
`   THEN SELECT`
`ENDIF`
	Select for output those records with column 6 is equal to 13 dashes. Skip all other records.

`IF (COL.6 = "\xFF")`
`   THEN SELECT`
`ENDIF`
	Select for output those records with column 6 is equal to hexadecimal FF. Skip all other records.

Example logic text

Meaning

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN SELECT
ELSE IF (COL.2 "D")
     THEN SELECT
     ELSE IF (COL.2 "G")
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Select for output those records with column 2 equal to "A" and column 3 greater than 10.

Also select for output those records with column 2 equal to "D".

Also select for output those records with column 2 equal to "G".

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) OR
   (COL.2 = "D") OR
   (COL.2 = "G")
   THEN SELECT
ENDIF
```

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN IF (COL.4 + COL.5
        > COL.6)
     THEN SELECT
     ELSE IF (COL.7 = 0)
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Consider those records with column 2 equal to "A" and column 3 greater than 10.

Of the considered records, select for output those records with column 4 plus column 5 is greater then column 6.

Of the considered records not yet selected, select also for output those records with column 7 equal to zero.

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) AND
   ((COL.4 + COL.5 > COL.6) OR
   (COL.7 = 0))
   THEN SELECT
ENDIF
```

Examples: IF with SKIP in Format Record Filter

Example logic text

Meaning

```
IF (COL.3 > 1000)
   THEN SKIP
ENDIF
```

Skip for output those records with column 3 greater than 1000. Select all other records. The code at left can also be written as:

```
SKIPIF(COL.3 > 1000)
```

Example logic text	Meaning

`IF (COL.2 = "ABC")`
`   THEN SKIP`
`ENDIF`
	Skip for output those records with column 2 equal to "ABC". Select all other records.

`IF NOT (COL.2 = "ABC")`
`   THEN SKIP`
`ENDIF`
	Skip those output records with field column 2 not equal to "ABC". Select all other records. The code at left gives the same result as:
	`SELECTIF(COL.2 = "ABC")`

`IF (COL.2 = "A") OR`
`   (COL.2 = "D")`
`   THEN SKIP`
`ENDIF`
	Skip for output those records with column 2 equal to "A" or "D". Select all other records.

`IF (COL.2 = "A") AND`
`   (COL.3 > 10)`
`   THEN SKIP`
`ENDIF`
	Skip for output those records with column 2 equal to "A" and column 3 greater than 10. Select all other records.

`IF (COL.3 + Col.4`
`    > Col.5 * 100)`
`   THEN SKIP`
`ENDIF`
	Skip for output those records with column 3 plus column 4 is greater than column 5 times 100. Select all other records.

`IF (COL.6 = ALL("-"))`
`   THEN SKIP`
`ENDIF`
	Skip for output those records with column 6 is equal to all dashes. Select all other records. This example gives the same result as:
	`SKIPIF(COL.6 = ALL("-"))`

`IF (COL.6 = REPEAT("-", 13))`
`   THEN SKIP`
`ENDIF`
	Skip for output those records with column 6 is equal to 13 dashes. Select all other records. This example gives the same result as:
	`SKIPIF(COL.6 = REPEAT("-", 13))`

`IF (COL.6 = "\xFF")`
`   THEN SKIP`
`ENDIF`
	Skip for output those records with column 6 is equal to hexadecimal FF. Select all other records. This example gives the same result as:
	`SKIPIF(COL.6 = "\xFF")`

Example logic text | **Meaning**

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN SKIP
ELSE IF (COL.2 = "D")
     THEN SKIP
     ELSE IF (COL.2 = "G")
          THEN SKIP
          ENDIF
     ENDIF
ENDIF
```

Skip for output those records with column 2 equal to "A" and column 3 greater than 10.

In addition, skip for output those records with column 2 equal to "D".

In addition, skip for output those records with column 2 equal to "G".

Select all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) OR
   (COL.2 = "D") OR
   (COL.2 = "G")
   THEN SKIP
ENDIF
```

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN IF (COL.4 + COL.5
        > COL.6)
     THEN SKIP
     ELSE IF (COL.7 = 0)
          THEN SKIP
          ENDIF
     ENDIF
ENDIF
```

Consider those records with column 2 equal to "A" and column 3 greater than 10.

Of the considered records, skip for output those records with column 4 plus column 5 is greater then column 6.

Of the considered records not yet skipped, skip also for output those records with column 7 equal to zero.

Select all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) AND
   ((COL.4 + COL.5 > COL.6) OR
    (COL.7 = 0))
   THEN SKIP
ENDIF
```

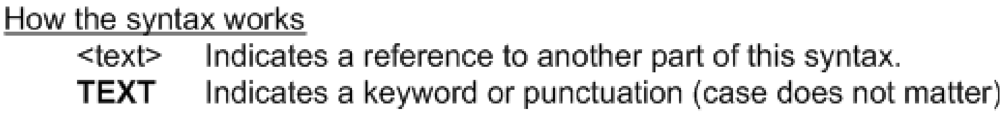
Syntax: lookup paths

How do I use lookup paths?

A lookup path is a method to use one logical file to lookup a record in another logical file. For an introduction, see topic "**Lookup paths overview**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

In logic text, lookup paths are only used in the extract phase, where you can use lookup paths to provide values for columns in view data. You can also use the ISFOUND and ISNOTFOUND functions to check if the lookup path is successful in each particular case.

Lookup paths are used in the format phase only in the Sort Key Title field, which is part of the definition of a sort column in a view. There is no logic text syntax for lookup paths in the format phase.

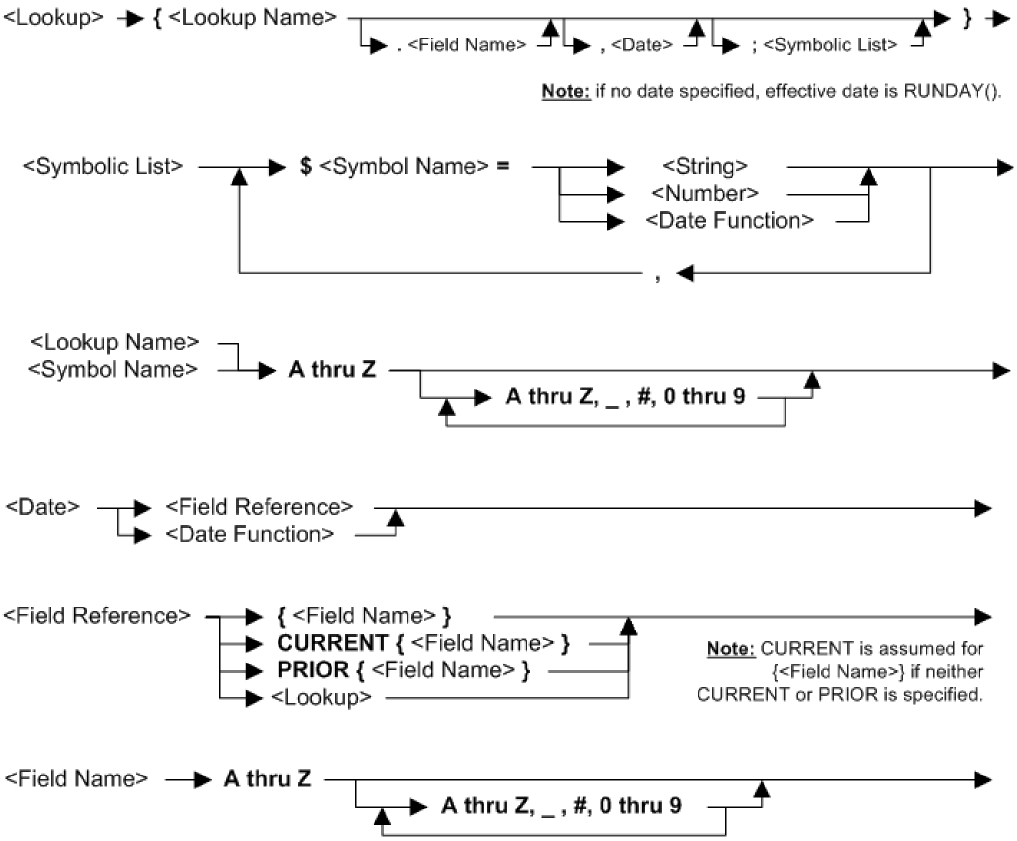


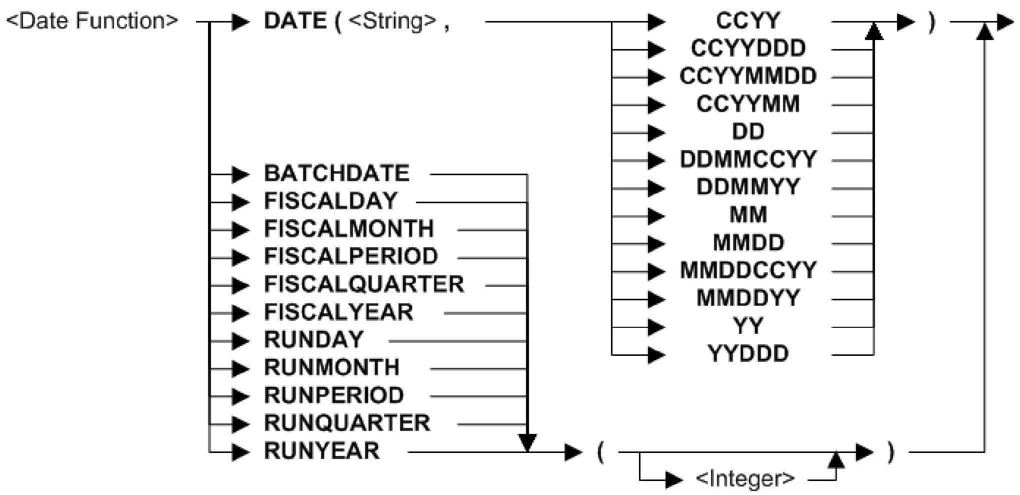
How the syntax works

<text> Indicates a reference to another part of this syntax.

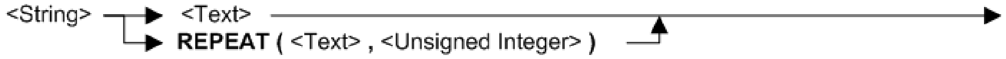
TEXT Indicates a keyword or punctuation (case does not matter)

Syntax

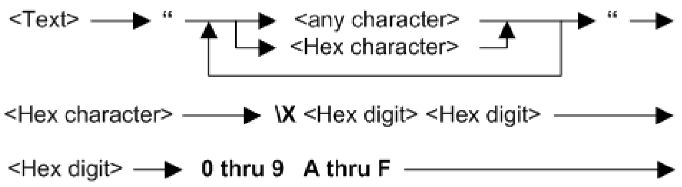


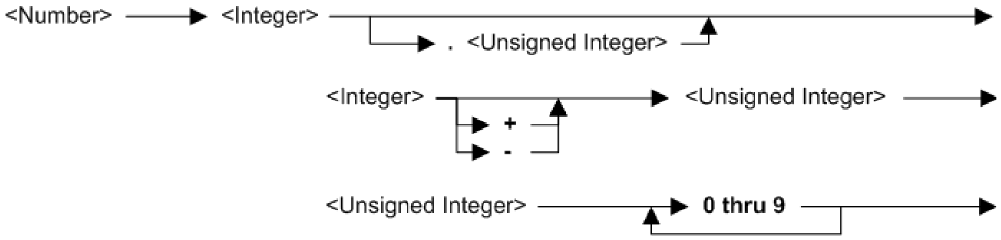


Note: In this situation, <Integer> means the number of units to add or subtract from the current value.



Note: <String> has maximum length 256 characters.





Rules for the syntax

Lookups can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: lookup paths in Extract Record Filter

Example logic text	Meaning

Example logic text	Meaning

`IF ISFOUND({Lookup3})`
`   THEN SELECT`
`ENDIF`
	Select all input records where the lookup path Lookup3 successfully finds a target record, and skip all other records. This example is the same as:
	`SELECTIF(ISFOUND({Lookup3})`

`IF ISFOUND({Lookup3.Field4})`
`   THEN SELECT`
`ENDIF`
	Select all input records where the lookup path Lookup3 successfully finds a target field of Field4, and skip all other records. This example is the same as:
	`SELECTIF(ISFOUND({Lookup3.Field4})`

`IF ISFOUND({Lookup2,field7})`
`   THEN SELECT`
`ENDIF`
	Select all input records where the lookup path Lookup2 successfully finds a target record using effective date of field7, and skip all other records. This example is the same as:
	`SELECTIF(ISFOUND({Lookup2,field7})`

`IF ISFOUND({Lookup2;$SYM="ABC"})`
`   THEN SELECT`
`ENDIF`
	Select all input records where the lookup path Lookup2 successfully finds a target record using symbol SYM set to "ABC", and skip all other records. This example is the same as:
	`SELECTIF(ISFOUND({Lookup2;$SYM="ABC"})`

`IF ISFOUND({Lookup5,`
`            field7;$SYM1=3,$SYM2=0})`
`   THEN SELECT`
`ENDIF`
	Select all input records where the lookup path Lookup5 successfully finds a target record using effective date of field7 and symbol SYM1 set to 3 and symbol SYM2 set to zero. Skip all other records. This example is the same as:
	`SELECTIF(ISFOUND({Lookup5,`
	`                 field7;$SYM1=3,$SYM2=0})`

`IF ISNOTFOUND({Lookup1})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup1 does not successfully find a target record, and select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup1})`

`IF ISNOTFOUND({Lookup1,field7})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup1 does not successfully find a target record using effective date of field7, and select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup1,field7})`

`IF ISNOTFOUND({Lookup5;$SYM1="ABC"})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup5 does not successfully find a target record using symbol SYM1 set to "ABC", and select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup5;$SYM1="ABC"})`

Example logic text | **Meaning**

```
IF ISNOTFOUND({Lookup2,field7
             ;$SYM1=3,$SYM2=0})
   THEN SKIP
ENDIF
```

Skip all input records where the lookup path Lookup2 does not successfully find a target record using effective date of field7 and symbol SYM1 set to 3 and symbol SYM2 set to zero. Select all other records. This example is the same as:

```
SKIPIF(ISNOTFOUND({Lookup2,field7
                  ;$SYM1=3,$SYM2=0})
```

Examples: lookup paths in Extract Column Assignment

Example logic text | **Meaning**

```
COLUMN = {Lookup1.Field2}
```

Set the current column using lookup path Lookup1 to provide Field2 in the target logical record.

```
COLUMN = {Lookup1.Field2,Field7}
```

Set the current column using lookup path Lookup1 to provide Field2 in the target logical record using effective date of Field7.

```
COLUMN = {Lookup1.Field2,;$SYM="ABC"}
```

Set the current column using lookup path Lookup1 to provide Field2 in the target logical record using symbol SYM set to 'ABC".

```
COLUMN = {Lookup3.Field4,
          Field7;$SYM1=3,$SYM2=0}
```

Set the current column using lookup path Lookup3 to provide Field4 in the target logical record using effective date of Field7 and symbols SYM1 set to 3 and SYM2 set to zero.

```
IF ISFOUND({Lookup4;$SYM="A"})
   THEN COLUMN = {Lookup4,
                ;$SYM="A"}
   ELSE COLUMN = 0
ENDIF
```

If lookup path Lookup4 successfully finds a value using symbol SYM set to "A", then set the current column to that lookup field, otherwise set the current column to zero.

Syntax: SELECT, SELECTIF, SKIP and SKIPIF statements

Introduction

The above statements are used to select records for processing as follows:

- In **Extract Record Filter**, these statements select input records for processing in the extract phase.
- In **Format Record Filter**, these statements select the view records for final output.

SELECT directly defines the records to select. **SELECTIF** means to select records based on a condition.

SKIP defines the records to skip. **SKIPIF** means skip records based on a condition.

See below for topics containing the syntax and some examples of these statements.

Syntax: SELECT and SELECTIF statements in Extract Record Filter

The purpose of SELECT and SELECTIF in Extract Record Filter

In this logic text, SELECT or SELECTIF define the input records to be included in the extract phase.

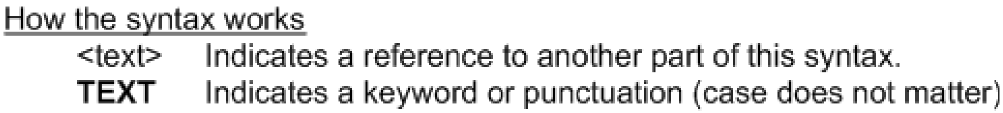
If there are **no SELECT, SELECTIF, SKIP or SKIPIF statements** in Extract Record Filter, then **all input records** are selected.

The idea is that you either SELECT or SKIP but you cannot do both in the same logic text. Once you have selected records then all others are skipped. Alternatively, once you skip records then all others are selected.

SELECTIF defines the input records to select based on a condition.

SELECT must always be inside an IF statement, in a THEN or ELSE case. The path through the IF statement decides which records reach the SELECT statement.

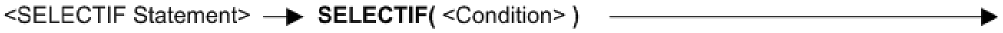
Extract Record Filter can have **one SELECTIF** or **one IF that contains one or more SELECT statements**. Once either of these is present, no SKIP or SKIPIF statements are allowed.



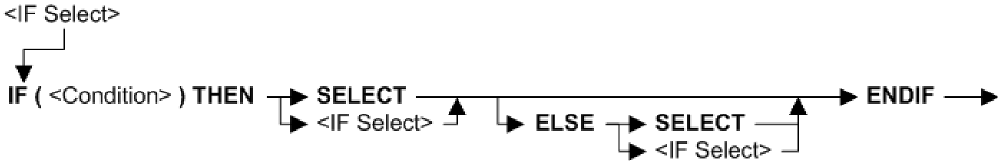
How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: <Condition> is defined under the topic: "**Syntax: IF statements in Extract Record Filter**"



Note: <Condition> is defined under the topic: "**Syntax: IF statements in Extract Record Filter**"

Rules for the syntax

- Extract Record Filter can have one SELECTIF statement or one IF statement that contains one or more SELECT statements. When one SELECTIF or SELECT is present, then no SKIP or SKIPIF statements are allowed.
- One IF statement can have a SELECT for the THEN or ELSE case. One IF statement can have other IF statements nested inside, which may also have SELECT statements inside. All this counts as one IF statement.

The best way to learn is the examples below. See also the topic "**Syntax: IF Statements in Extract Record Filter**".

To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: SELECTIF in Extract Record Filter

Example logic text	Meaning

`SELECTIF(CURRENT({field1}) <>`
`        PRIOR({field1}))`
	Select only records with unique values for field1. This assumes the input file is sorted into field1 order.

`SELECTIF({field2} > 1000)`	Select for output only those records with field2 greater than 1000. Skip all other records. The code at left is a shorthand for:
	`IF {field2} > 1000`
	`   THEN SELECT`
	`ENDIF`

`SELECTIF({field3} = "ABC")`	Select for output only those records with field3 equal to "ABC". Skip all other records.
`SELECTIF(NOT {field3} = "ABC")`	Select those output records with field3 not equal to "ABC". Skip all other records. This example gives the same result as: `SKIPIF({field3} = "ABC")`

`SELECTIF({field3} = "A" OR`
`        {field3} = "D")`
	Select for output only those records with field3 equal to "A" or "D". Skip all other records.

`SELECTIF({field3} = "A" AND`
`        {field4} > 10)`
	Select for output only those records with field3 equal to "A" and field4 greater than 10. Skip all other records.

`SELECTIF({field4} + {field5}`
`         > {field6})`
	Select for output only those records with field4 plus field5 is greater than field6. Skip all other records.

`SELECTIF(NOT {field7} = ALL("-"))`	Select for output those records with field7 is not equal to all dashes. Skip all other records. This example gives the same result as: `SKIPIF({field7} = ALL("-"))`

`SELECTIF(NOT {field7} =`
`        REPEAT("-", 13))`
	Select for output those records with field7 is not equal to 13 dashes. Skip all other records. This example gives the same result as:
	`SKIPIF({field7} = REPEAT("-", 13))`

`SELECTIF(NOT {field7} = "\xFF")`	Select for output those records with field7 is not equal to hexadecimal FF. Skip all other records. This example gives the same result as: `SKIPIF({field7} = "\xFF")`

Example logic text	Meaning
SELECTIF(ISFOUND({Lookup1})	Select all input records where the lookup path Lookup1 successfully finds a target record, and skip all other records.
SELECTIF(ISFOUND({Lookup1,field7})	Select all input records where the lookup path Lookup1 successfully finds a target record using effective date of field7, and skip all other records.
SELECTIF(ISFOUND({Lookup1;$SYM="A"})	Select all input records where the lookup path Lookup1 successfully finds a target record using symbol SYM set to "A", and skip all other records.
SELECTIF(ISFOUND({Lookup1, field7;$SYM1=3,$SYM2=0})	Select all input records where the lookup path Lookup1 successfully finds a target record using effective date of field7 and symbol SYM1 set to 3 and symbol SYM2 set to zero. Skip all other records.
SELECTIF(ISNOTNULL({field1})	Select all input records where field1 does not contain null values, and skip all other records.
SELECTIF(ISNUMERIC({field2})	Select all input records where field2 is numeric, and skip all other records.
SELECTIF(ISNOTSPACES({field3})	Select all input records where field3 is not spaces, and skip all other records.
SELECTIF(DAYSBETWEEN({field1},{field2}) > 7)	Select only records where there are more than 7 days between field1 and field2, and skip all other records.
SELECTIF({field1} BEGINS_WITH "BBB")	Select input records where field1 begins with characters "BBB", and skip all other records.
SELECTIF({field2} CONTAINS "CCC")	Select input records where field2 contains characters "CCC", and skip all other records.
SELECTIF({field3} ENDS_WITH "EEE")	Select input records where field3 ends with characters "EEE", and skip all other records.
SELECTIF({field4} MATCHES "...")	Select input records where field4 matches characters "...", and skip all other records.
SELECTIF({field5} LIKE "MA...")	Select input records where field5 is exactly 5 characters starting with "MA", and skip all other records.

Example logic text	Meaning
`SELECTIF({field6} LIKE "..VA..")`	Select input records where field6 is exactly 6 characters with characters 3 and 4 as "VA", and skip all other records.
`SELECTIF({field7} LIKE ".....NA")`	Select input records where field7 is exactly 6 characters ending in "NA", and skip all other records.
`SELECTIF({field8} LIKE "^BBB*")`	Select input records where field8 begins with characters "BBB", and skip all other records. This example has the same effect as: `SELECTIF({field8} BEGINS_WITH "BBB")` It is better to use BEGINS_WITH because the logic text executes faster.
`SELECTIF({field9} LIKE "*CCC*")`	Select input records where field9 contains characters "CCC", and skip all other records. This example has the same effect as: `SELECTIF({field9} CONTAINS "CCC")` It is better to use CONTAINS because the logic text executes faster.
`SELECTIF({field1} LIKE "*EEE$")`	Select input records where field1 ends with characters "EEE", and skip all other records. This example has the same effect as: `SELECTIF({field1} ENDS_WITH "EEE")` It is better to use ENDS_WITH because the logic text executes faster.
`SELECTIF({field2} LIKE "^B*C*E$")`	Select input records where field2 begins with "B", contains "C" and ends with "E", and skip all other records.

Examples: IF with SELECT in Extract Record Filter

Example logic text	Meaning

`IF (CURRENT({field1}) <>`
`    PRIOR({field1}))`
`   THEN SELECT`
`ENDIF`
	Select only records with unique values for field1. This assumes the input file is sorted into field1 order. This example can also be written:
	`SELECTIF(CURRENT({field1}) <>`
	`         PRIOR({field1}))`

`IF ({field3} > 1000)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field3 greater than 1000. Skip all other records. The code at left can also be written as:
	`SELECTIF({field3} > 1000)`

Example logic text	Meaning

`IF ({field2} = "ABC")`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field2 equal to "ABC". Skip all other records.

`IF NOT ({field2} = "ABC")`
`   THEN SELECT`
`ENDIF`
	Select those output records with field2 not equal to "ABC". Skip all other records. The code at left gives the same result as:
	`SKIPIF({field2} = "ABC")`

`IF ({field2} = "ABC") OR`
`   ({field2} = "DEF")`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field2 equal to "ABC" or "DEF". Skip all other records.

`IF ({field2} = "ABC") AND`
`   ({field3} > 1000)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field2 equal to "ABC" and field3 greater than 1000. Skip all other records.

`IF ({field3} + {field4} >`
`    {field5} * 100)`
`   THEN SELECT`
`ENDIF`
	Select for output only those records with field3 plus field4 is greater than field5 times 100. Skip all other records.

`IF NOT ({field6} = ALL("-"))`
`   THEN SELECT`
`ENDIF`
	Select for output those records with field6 is not equal to all dashes. Skip all other records. This example gives the same result as:
	`SKIPIF({field6} = ALL("-"))`

`IF ({field6} = REPEAT("-", 13))`
`   THEN SELECT`
`ENDIF`
	Select for output those records with field6 is equal to 13 dashes. Skip all other records.

`IF ({field6} = "\xFF")`
`   THEN SELECT`
`ENDIF`
	Select for output those records with field6 is equal to hexadecimal FF. Skip all other records.

Example logic text | **Meaning**

```
IF ({field2} = "ABC") AND
   ({field3} > 10)
THEN SELECT
ELSE IF ({field2} = "DEF")
     THEN SELECT
     ELSE IF ({field2} = "GHI")
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Select for output those records with field2 equal to "ABC" and field3 greater than 10.

In addition, select for output those records with field2 equal to "DEF".

In addition, select for output those records with field2 equal to "GHI".

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF ({field2} = "ABC" AND
   {field3} > 10)          OR
   ({field2} = "DEF")      OR
   ({field2} = "GHI")
   THEN SELECT
ENDIF
```

```
IF ({field2} = "ABC") AND
   ({field3} > 10)
THEN IF ({field4} + {field5}
        > {field6})
     THEN SELECT
     ELSE IF ({field7} = 0)
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Consider those records with field2 equal to "ABC" and field3 greater than 10.

Of the considered records, select for output those records with field4 plus field5 is greater then field6.

Of the considered records not yet selected, select also for output those records with field7 equal to zero.

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF ({field2} = "ABC" AND
    {field3} > 10)
                          AND
   (({field4} + {field5}
       > {field6})   OR
   ({field7} = 0))
   THEN SELECT
ENDIF
```

```
IF ISFOUND({Lookup3})
   THEN SELECT
ENDIF
```

Select all input records where the lookup path Lookup3 successfully finds a target record, and skip all other records. This example is the same as:

```
SELECTIF(ISFOUND({Lookup3})
```

Example logic text **Meaning**

```
IF ISFOUND({Lookup3,field7})
   THEN SELECT
ENDIF
```

Select all input records where the lookup path Lookup3 successfully finds a target record using effective date of field7, and skip all other records. This example is the same as:

```
SELECTIF(ISFOUND({Lookup3,field7})
```

```
IF ISFOUND({Lookup3;$SYM="A"})
   THEN SELECT
ENDIF
```

Select all input records where the lookup path Lookup3 successfully finds a target record using symbol SYM set to "A", and skip all other records. This example is the same as:

```
SELECTIF(ISFOUND({Lookup3;$SYM="A"})
```

```
IF ISFOUND({Lookup3,
            field7;$SYM1=3,$SYM2=0})
   THEN SELECT
ENDIF
```

Select all input records where the lookup path Lookup3 successfully finds a target record using effective date of field7 and symbol SYM1 set to 3 and symbol SYM2 set to zero. Skip all other records. This example is the same as:

```
SELECTIF(ISFOUND({Lookup3,
         field7;$SYM1=3,$SYM2=0})
```

```
IF DAYSBETWEEN({field1},{field2})
     > 7
   THEN SELECT
ENDIF
```

Select only records where there are more than 7 days between field1 and field2, and skip all other records This example can also be written:

```
SELECTIF(DAYSBETWEEN({field1},{field2})
          > 7)
```

```
IF ({field1} BEGINS_WITH "BBB")
   THEN SELECT
ENDIF
```

Select input records where field1 begins with characters "BBB", and skip all other records. This example can be written:

```
SELECTIF({field1} BEGINS_WITH "BBB")
```

It is better to use BEGINS_WITH because the logic text executes faster.

```
IF ({field2} CONTAINS "CCC")
   THEN SELECT
ENDIF
```

Select input records where field2 contains characters "CCC", and skip all other records. This example can be written:

```
SELECTIF({field2} CONTAINS "CCC")
```

It is better to use CONTAINS because the logic text executes faster.

```
IF ({field3} ENDS_WITH "EEE")
   THEN SELECT
ENDIF
```

Select input records where field3 ends with characters "EEE", and skip all other records. This example can be written:

```
SELECTIF({field3} ENDS_WITH "EEE")
```

It is better to use ENDS_WITH because the logic text executes faster.

```
IF ({field4} MATCHES "...")
   THEN SELECT
ENDIF
```

Select input records where field4 matches characters "...", and skip all other records. This example can be written:

```
SELECTIF({field4} MATCHES "...")
```

Example logic text	Meaning

`IF ({field1} LIKE "^BBB*")`
`   THEN SELECT`
`ENDIF`
	Select input records where field1 begins with characters "BBB", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} BEGINS_WITH "BBB")`
	It is better to use BEGINS_WITH because the logic text executes faster.

`IF ({field1} LIKE "*CCC*")`
`   THEN SELECT`
`ENDIF`
	Select input records where field1 contains characters "CCC", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} CONTAINS "CCC")`
	It is better to use CONTAINS because the logic text executes faster.

`IF ({field1} LIKE "*EEE$")`
`   THEN SELECT`
`ENDIF`
	Select input records where field1 ends with characters "EEE", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} ENDS_WITH "EEE")`
	It is better to use ENDS_WITH because the logic text executes faster.

`IF ({field1} LIKE "^B*C*E$")`
`   THEN SELECT`
`ENDIF`
	Select input records where field1 begins with "B", contains "C" and ends with "E", and skip all other records.

Syntax: SKIP and SKIPIF statements in Extract Record Filter

The purpose of SKIP and SKIPIF in Extract Record Filter

In this logic text, SKIP or SKIPIF define the input records to be excluded in the extract phase.

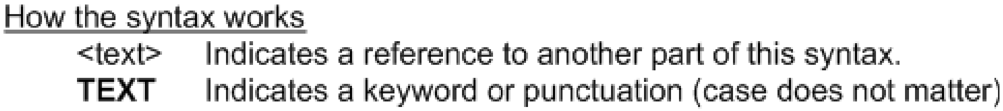
If there are **no SELECT, SELECTIF, SKIP or SKIPIF statements** in Extract Record Filter, then **all input records** are selected.

The idea is that you either SELECT or SKIP but you cannot do both in the same logic text. Once you have selected records then all others are skipped. Alternatively, once you skip records then all others are selected.

SKIPIF defines the input records to skip based on a condition.

SKIP must always be inside an IF statement, in a THEN or ELSE case. The path through the IF statement decides which records reach the SKIP statement.

Extract Record Filter can have **one SKIPIF** or **one IF that contains one or more SKIP statements**. Once either of these is present, no SELECT or SELECTIF statements are allowed.

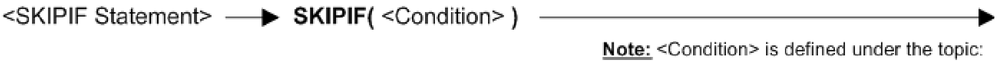


How the syntax works

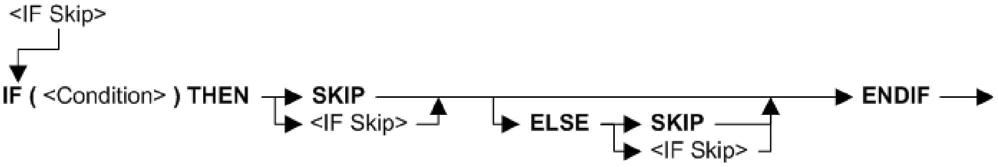
<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax



Note: <Condition> is defined under the topic: "**Syntax: IF statements in Extract Record Filter**"



Note: <Condition> is defined under the topic: "**Syntax: IF statements in Extract Record Filter**"

Rules for the syntax

- Extract Record Filter can have one SKIPIF statement or one IF statement that contains one or more SKIP statements. When one SKIPIF or SKIP is present, then no SELECT or SELECTIF statements are allowed.
- One IF statement can have a SKIP for the THEN or ELSE case. One IF statement can have other IF statements nested inside, which may also have SKIP statements inside. All this counts as one IF statement.

The best way to learn is the examples below. See also the topic "**Syntax: IF Statements in Extract Record Filter**".

To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: SKIPIF in Extract Record Filter

Example logic text	Meaning

`SKIPIF(CURRENT({field4}) =`
`       PRIOR({field4}))`
	Skip records where the new field4 value is the same as the previous field4. This assumes the input file is sorted into field4 order. This selects only the input records where field4 is a new value (compared to the previous record).

`SKIPIF({field1} > 1000)`	Skip for output those records with field1 greater than 1000. Select all other records. The code at left is a shorthand for:
	`IF ({field1} > 1000)`
	`   THEN SKIP`
	`ENDIF`

Example logic text	Meaning
`SKIPIF({field2} = "ABC")`	Skip for output those records with field2 equal to "ABC". Select all other records.
`SKIPIF(NOT {field2} = "ABC")`	Skip those output records with field2 not equal to "ABC". Select all other records. This example gives the same result as: `SELECTIF({field2} = "ABC")`
`SKIPIF({field3} = "A" OR {field3} = "D")`	Skip for output those records with field3 equal to "A" or "D". Select all other records.
`SKIPIF({field4} = "A" AND {field5} > 10)`	Skip for output those records with field4 equal to "A" and field5 greater than 10. Select all other records.
`SKIPIF({field6} * 2 > {field8}+ 5)`	Skip for output those records with field6 times 2 is greater than field8 plus 5. Select all other records.
`SKIPIF({field6} = ALL("-"))`	Skip for output those records with field6 is equal to all dashes. Select all other records.
`SKIPIF({field6} = REPEAT("-", 13))`	Skip for output those records with field6 is equal to 13 dashes. Select all other records.
`SKIPIF({field6} = "\xFF")`	Skip for output those records with field6 is equal to hexadecimal FF. Select all other records.
`SKIPIF(ISNOTFOUND({Lookup2})`	Skip all input records where the lookup path Lookup2 does not successfully find a target record, and select all other records.
`SKIPIF(ISNOTFOUND({Lookup2,field7})`	Skip all input records where the lookup path Lookup2 does not successfully finds a target record using effective date of field7, and select all other records.
`SKIPIF(ISNOTFOUND({Lookup2;$SYM="A"})`	Skip all input records where the lookup path Lookup2 using symbol SYM set to "A" does not successfully finds a target record, and select all other records.

`SKIPIF(ISNOTFOUND({Lookup2,`
`field8;$SYM1=3,$SYM2=0})`
	Skip all input records where the lookup path Lookup2 using effective date of field8 and symbol SYM1 set to 3 and symbol SYM2 set to zero does not successfully finds a target record. Select all other records.

Example logic text	Meaning
`SKIPIF(ISNULL({field1})`	Skip all input records where field1 contains null values, and select all other records.
`SKIPIF(ISNOTNUMERIC({field2})`	Skip all input records where field2 is not numeric, and select all other records.
`SKIPIF(ISSPACES({field3})`	Skip all input records where field3 is spaces, and select all other records.
`SKIPIF(DAYSBETWEEN({field4},{field5}) > 7)`	Skip only records where there are more than 7 days between field4 and field5, and select all other records.
`SKIPIF({field1} BEGINS_WITH "BBB")`	Skip input records where field1 begins with characters "BBB", and select all other records.
`SKIPIF({field2} CONTAINS "CCC")`	Skip input records where field2 contains characters "CCC", and select all other records.
`SKIPIF({field3} ENDS_WITH "EEE")`	Skip input records where field3 ends with characters "EEE", and select all other records.
`SKIPIF({field4} MATCHES "...")`	Skip input records where field4 matches characters "...", and select all other records.
`SKIPIF({field5} LIKE "MA...")`	Skip input records where field5 is exactly 5 characters starting with "MA", and select all other records.
`SKIPIF({field6} LIKE "..VA..")`	Skip input records where field6 is exactly 6 characters with characters 3 and 4 as "VA", and select all other records.
`SKIPIF({field7} LIKE ".....NA")`	Skip input records where field7 is exactly 6 characters ending in "NA", and select all other records.
`SKIPIF({field8} LIKE "^BBB*")`	Skip input records where field8 begins with characters "BBB", and select all other records. This example has the same effect as: `SKIPIF({field8} BEGINS_WITH "BBB")` It is better to use BEGINS_WITH because the logic text executes faster.

Example logic text **Meaning**

```
SKIPIF({field9} LIKE "*CCC*")
```

Skip input records where field9 contains characters "CCC", and select all other records. This example has the same effect as:

```
SKIPIF({field9} CONTAINS "CCC")
```

It is better to use CONTAINS because the logic text executes faster.

```
SKIPIF({field1} LIKE "*EEE$")
```

Skip input records where field1 ends with characters "EEE", and select all other records. This example has the same effect as:

```
SKIPIF({field1} ENDS_WITH "EEE")
```

It is better to use ENDS_WITH because the logic text executes faster.

```
SKIPIF({field2} LIKE "^B*C*E$")
```

Skip input records where field2 begins with "B", contains "C" and ends with "E", and select all other records.

Examples: IF with SKIP in Extract Record Filter

Example logic text **Meaning**

```
IF (CURRENT({field1}) =
    PRIOR({field1}))
   THEN SKIP
ENDIF
```

Skip records where field1 is the same as the previous record. This assumes the input file is sorted into field1 order. This example can also be written:

```
SKIPIF(CURRENT({field1}) =
       PRIOR({field1}))
```

```
IF ({field3} > 1000)
   THEN SKIP
ENDIF
```

Skip for output those records with field3 greater than 1000. Select all other records. The code at left can also be written as:

```
SKIPIF({field3} > 1000)
```

```
IF ({field2} = "ABC")
   THEN SKIP
ENDIF
```

Skip for output those records with field2 equal to "ABC". Select all other records.

```
IF NOT ({field2} = "ABC")
   THEN SKIP
ENDIF
```

Skip those output records with field2 not equal to "ABC". Select all other records. The code at left gives the same result as:

```
SELECTIF({field2} = "ABC")
```

```
IF ({field2} = "A") OR
   ({field2} = "D")
   THEN SKIP
ENDIF
```

Skip for output those records with field2 equal to "A" or "D". Select all other records.

```
IF ({field2} = "A") AND
   ({field3} > 10
   THEN SKIP
ENDIF
```

Skip for output those records with field2 equal to "A" and field3 greater than 10. Select all other records.

Example logic text | **Meaning**

```
IF ({field3} + {field4}
    > {field5})
   THEN SKIP
ENDIF
```

Skip for output those records with field3 plus field4 is greater than field5. Select all other records.

```
IF ({field6} = ALL("-"))
   THEN SKIP
ENDIF
```

Skip for output those records with field6 is equal to all dashes. Select all other records. This example gives the same result as:

```
SKIPIF({field6} = ALL("-"))
```

```
IF ({field6} = REPEAT("-", 13))
   THEN SKIP
ENDIF
```

Skip for output those records with field6 is equal to 13 dashes. Select all other records. This example gives the same result as:

```
SKIPIF({field6} = REPEAT("-", 13))
```

```
IF ({field6} = "\xFF")
   THEN SKIP
ENDIF
```

Skip for output those records with field6 is equal to hexadecimal FF. Select all other records. This example gives the same result as:

```
SKIPIF({field6} = "\xFF")
```

```
IF ({field2} = "A") AND
   ({field3} > 10)
THEN SKIP
ELSE IF ({field2} = "D")
     THEN SKIP
     ELSE IF ({field2} = "G")
          THEN SKIP
          ENDIF
     ENDIF
ENDIF
```

Skip for output those records with field2 equal to "A" and field3 greater than 10.

In addition, skip for output those records with field2 equal to "D".

In addition, skip for output those records with field2 equal to "G".

Select all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF ({field2} = "A" AND
    {field3} > 10)         OR
   ({field2} = "D") OR
   ({field2} = "G")
   THEN SKIP
ENDIF
```

```
IF ISNOTFOUND({Lookup4})
   THEN SKIP
ENDIF
```

Skip all input records where the lookup path Lookup4 does not successfully find a target record, and select all other records. This example is the same as:

```
SKIPIF(ISNOTFOUND({Lookup4})
```

```
IF ISNOTFOUND({Lookup4,field7})
   THEN SKIP
ENDIF
```

Skip all input records where the lookup path Lookup4 does not successfully find a target record using effective date of field7, and select all other records. This example is the same as:

```
SKIPIF(ISNOTFOUND({Lookup4,field7})
```

Example logic text	Meaning

`IF ISNOTFOUND({Lookup4;$SYM="A"})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup4 does not successfully find a target record using symbol SYM set to "A", and select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup4;$SYM="A"})`

`IF ISNOTFOUND`
`   ({Lookup4,field7;$SYM1=3,$SYM2=0})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup4 does not successfully find a target record using effective date of field7 and symbol SYM1 set to 3 and symbol SYM2 set to zero. Select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup4,`
	`                 field7;$SYM1=3,$SYM2=0})`

`IF DAYSBETWEEN({field1},{field2})`
`     > 7`
`   THEN SKIP`
`ENDIF`
	Skip records where there are more than 7 days between field1 and field2, and select all other records. This example can also be written:
	`SKIPIF(DAYSBETWEEN({field1},{field2})`
	`       > 7)`

`IF ({field1} BEGINS_WITH "BBB")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 begins with characters "BBB", and select all other records. This example can be written:
	`SKIPIF({field1} BEGINS_WITH "BBB")`
	It is better to use BEGINS_WITH because the logic text executes faster.

`IF ({field2} CONTAINS "CCC")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field2 contains characters "CCC", and select all other records. This example can be written:
	`SKIPIF({field2} CONTAINS "CCC")`
	It is better to use CONTAINS because the logic text executes faster.

`IF ({field3} ENDS_WITH "EEE")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field3 ends with characters "EEE", and select all other records. This example can be written:
	`SKIPIF({field3} ENDS_WITH "EEE")`
	It is better to use ENDS_WITH because the logic text executes faster.

`IF ({field4} MATCHES "...")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field4 matches characters "...", and select all other records. This example can be written:
	`SKIPIF({field4} MATCHES "...")`

`IF ({field1} LIKE "^BBB*")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 begins with characters "BBB", and select all other records. This example has the same effect as:
	`SKIPIF({field1} BEGINS_WITH "BBB")`
	It is better to use BEGINS_WITH because the logic text executes faster.

Example logic text	Meaning

`IF ({field1} LIKE "*CCC*")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 contains characters "CCC", and select all other records. This example has the same effect as:
	`SKIPIF({field1} CONTAINS "CCC")`
	It is better to use CONTAINS because the logic text executes faster.

`IF ({field1} LIKE "*EEE$")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 ends with characters "EEE", and select all other records. This example has the same effect as:
	`SKIPIF({field1} ENDS_WITH "EEE")`
	It is better to use ENDS_WITH because the logic text executes faster.

`IF ({field1} LIKE "^B*C*E$")`
`   THEN SKIP`
`ENDIF`
	Skip input records where field1 begins with "B", contains "C" and ends with "E", and select all other records.

Syntax: SELECT and SELECTIF statements in Format Record Filter

The purpose of SELECT and SELECTIF in Format Record Filter

In this logic text, SELECT or SELECTIF define the view records to be included in the final output of the view.

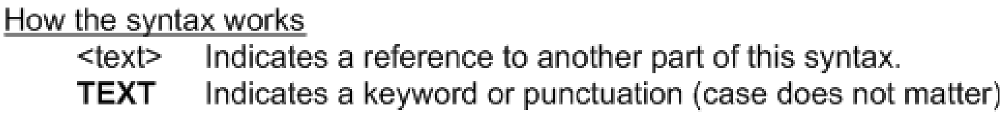
If there are **no SELECT, SELECTIF, SKIP or SKIPIF statements** in Format Record Filter, then **all view records** are selected.

The idea is that you either SELECT or SKIP but you cannot do both in the same logic text. Once you have selected records then all others are skipped. Alternatively, once you skip records then all others are selected.

SELECTIF defines the view records to select based on a condition.

SELECT must always be inside an IF statement, in a THEN or ELSE case. The path through the IF statement decides which records reach the SELECT statement.

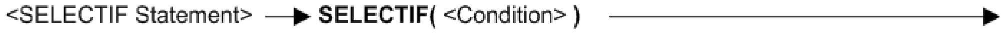
Format Record Filter can have **one SELECTIF** or **one IF that contains one or more SELECT statements**. Once either of these is present, no SKIP or SKIPIF statements are allowed.



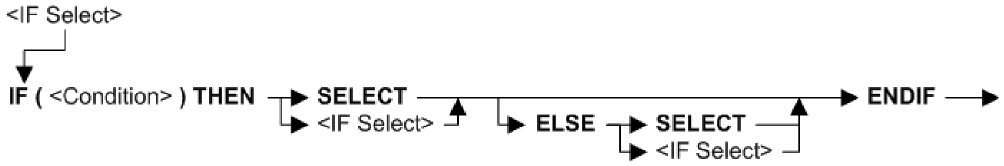
How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: <Condition> is defined under the topic: "**Syntax: IF statements in Format Record Filter**"



Note: <Condition> is defined under the topic: "**Syntax: IF statements in Format Record Filter**"

Rules for the syntax

- Format Record Filter can have one SELECTIF statement or one IF statement that contains one or more SELECT statements. When one SELECTIF or SELECT is present, then no SKIP or SKIPIF statements are allowed.
- One IF statement can have a SELECT for the THEN or ELSE case. One IF statement can have other IF statements nested inside, which may also have SELECT statements inside. All this counts as one IF statement.

The best way to learn is the examples below. See also the topic "**Syntax: IF Statements in Format Record Filter**".

To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: SELECTIF in Format Record Filter

Example logic text	**Meaning**

`SELECTIF(COL.3 > 1000)`	Select for output only those records with column 3 greater than 1000. Skip all other records. The code at left is a shorthand for:
	`IF (COL.3 > 1000)`
	`   THEN SELECT`
	`ENDIF`

`SELECTIF(COL.2 = "ABC")`	Select for output only those records with column 2 equal to "ABC". Skip all other records.
`SELECTIF(NOT COL.2 = "ABC")`	Select those output records with field column 2 not equal to "ABC". Skip all other records. This example gives the same result as: `SKIPIF(COL.2 = "ABC")`

`SELECTIF(COL.2 = "A" OR`
`         COL.2 = "D")`
	Select for output only those records with column 2 equal to "A" or "D". Skip all other records.

`SELECTIF(COL.2 = "A" AND`
`         COL.3 > 10)`
	Select for output only those records with column 2 equal to "A" and column 3 greater than 10. Skip all other records.

Example logic text

Meaning

```
SELECTIF(COL.3 + Col.4 > Col.5)
```

Select for output only those records with column 3 plus column 4 is greater than column 5. Skip all other records.

```
SELECTIF(NOT COL.6 = ALL("-"))
```

Select for output those records with column 6 is not equal to all dashes. Skip all other records. This example gives the same result as:

```
SKIPIF(COL.6 = ALL("-"))
```

```
SELECTIF(NOT COL.6 =
         REPEAT("-", 13))
```

Select for output those records with column 6 is not equal to 13 dashes. Skip all other records. This example gives the same result as:

```
SKIPIF(COL.6 = REPEAT("-", 13))
```

```
SELECTIF(NOT COL.6 = "\xFF")
```

Select for output those records with column 6 is not equal to hexadecimal FF. Skip all other records. This example gives the same result as:

```
SKIPIF(COL.6 = "\xFF")
```

Examples: IF with SELECT in Format Record Filter

Example logic text

Meaning

```
IF (COL.3 > 1000)
   THEN SELECT
ENDIF
```

Select for output only those records with column 3 greater than 1000. Skip all other records. The code at left can also be written as:

```
SELECTIF(COL.3 > 1000)
```

```
IF (COL.2 = "ABC")
   THEN SELECT
ENDIF
```

Select for output only those records with column 2 equal to "ABC". Skip all other records.

```
IF NOT (COL.2 = "ABC")
   THEN SELECT
ENDIF
```

Select those output records with field column 2 not equal to "ABC". Skip all other records. The code at left gives the same result as:

```
SKIPIF(COL.2 = "ABC")
```

```
IF (COL.2 = "A") OR
   (COL.2 = "D")
   THEN SELECT
ENDIF
```

Select for output only those records with column 2 equal to "A" or "D". Skip all other records.

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
   THEN SELECT
ENDIF
```

Select for output only those records with column 2 equal to "A" and column 3 greater than 10. Skip all other records.

```
IF (COL.3 + Col.4 > Col.5 * 100)
   THEN SELECT
ENDIF
```

Select for output only those records with column 3 plus column 4 is greater than column 5 times 100. Skip all other records.

Example logic text **Meaning**

```
IF NOT (COL.6 = ALL("-"))
   THEN SELECT
ENDIF
```

Select for output those records with column 6 is not equal to all dashes. Skip all other records. This example gives the same result as:

```
SKIPIF(COL.6 = ALL("-"))
```

```
IF (COL.6 = REPEAT("-", 13))
   THEN SELECT
ENDIF
```

Select for output those records with column 6 is equal to 13 dashes. Skip all other records.

```
IF (COL.6 = "\xFF")
   THEN SELECT
ENDIF
```

Select for output those records with column 6 is equal to hexadecimal FF. Skip all other records.

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN SELECT
ELSE IF (COL.2 "D")
     THEN SELECT
     ELSE IF (COL.2 "G")
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Select for output those records with column 2 equal to "A" and column 3 greater than 10.

Also select for output those records with column 2 equal to "D".

Also select for output those records with column 2 equal to "G".

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) OR
   (COL.2 = "D") OR
   (COL.2 = "G")
   THEN SELECT
ENDIF
```

Example logic text

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN IF (COL.4 + COL.5
         > COL.6)
     THEN SELECT
     ELSE IF (COL.7 = 0)
          THEN SELECT
          ENDIF
     ENDIF
ENDIF
```

Meaning

Consider those records with column 2 equal to "A" and column 3 greater than 10.

Of the considered records, select for output those records with column 4 plus column 5 is greater then column 6.

Of the considered records not yet selected, select also for output those records with column 7 equal to zero.

Skip all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) AND
   ((COL.4 + COL.5 > COL.6) OR
   (COL.7 = 0))
   THEN SELECT
ENDIF
```

Syntax: SKIP and SKIPIF statements in Format Record Filter

The purpose of SKIP and SKIPIF in Format Record Filter

In this logic text, SKIP or SKIPIF define the view records to be excluded from the final output of the view.

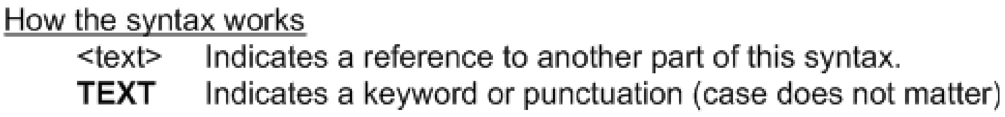
If there are **no SELECT, SELECTIF, SKIP or SKIPIF statements** in Format Record Filter, then **all view records** are selected.

The idea is that you either SELECT or SKIP but you cannot do both in the same logic text. Once you have selected records then all others are skipped. Alternatively, once you skip records then all others are selected.

SKIPIF defines the view records to skip based on a condition.

SKIP must always be inside an IF statement, in a THEN or ELSE case. The path through the IF statement decides which records reach the SKIP statement.

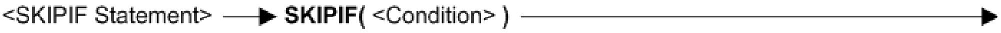
Format Record Filter can have **one SKIPIF** or **one IF that contains one or more SKIP statements**. Once either of these is present, no SELECT or SELECTIF statements are allowed.



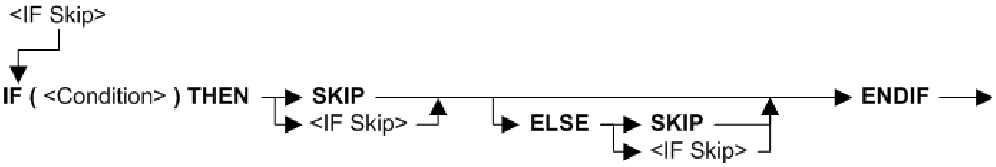
How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: <Condition> is defined under the topic: "**Syntax: IF statements in Format Record Filter**"



Note: <Condition> is defined under the topic: "**Syntax: IF statements in Format Record Filter**"

Rules for the syntax

- Format Record Filter can have one SKIPIF statement or one IF statement that contains one or more SKIP statements. When one SKIPIF or SKIP is present, then no SELECT or SELECTIF statements are allowed.
- One IF statement can have a SKIP for the THEN or ELSE case. One IF statement can have other IF statements nested inside, which may also have SKIP statements inside. All this counts as one IF statement.

The best way to learn is the examples below. See also the topic "**Syntax: IF Statements in Format Record Filter**".

To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: SKIPIF in Format Record Filter

Example logic text	**Meaning**

`SKIPIF(COL.3 > 1000)`	Skip for output those records with column 3 greater than 1000. Select all other records. The code at left is a shorthand for:
	`IF (COL.3 > 1000)`
	`   THEN SKIP`
	`ENDIF`

`SKIPIF(COL.2 = "ABC")`	Skip for output those records with column 2 equal to "ABC". Select all other records.
`SKIPIF(NOT COL.2 = "ABC")`	Skip those output records with field column 2 not equal to "ABC". Select all other records. This example gives the same result as: `SELECTIF(COL.2 = "ABC")`

`SKIPIF(COL.2 = "A" OR`
`       COL.2 = "D")`
	Skip for output those records with column 2 equal to "A" or "D". Select all other records.

`SKIPIF(COL.2 = "A" AND`
`       COL.3 > 10)`
	Skip for output those records with column 2 equal to "A" and column 3 greater than 10. Select all other records.

Example logic text **Meaning**

```
SKIPIF(COL.3 + Col.4 > Col.5)
```

Skip for output those records with column 3 plus column 4 is greater than column 5. Select all other records.

```
SKIPIF(COL.6 = ALL("-"))
```

Skip for output those records with column 6 is equal to all dashes. Select all other records.

```
SKIPIF(COL.6 = REPEAT("-", 13))
```

Skip for output those records with column 6 is equal to 13 dashes. Select all other records.

```
SKIPIF(COL.6 = "\xFF")
```

Skip for output those records with column 6 is equal to hexadecimal FF. Select all other records.

Examples: IF with SKIP in Format Record Filter

Example logic text **Meaning**

```
IF (COL.3 > 1000)
   THEN SKIP
ENDIF
```

Skip for output those records with column 3 greater than 1000. Select all other records. The code at left can also be written as:

```
SKIPIF(COL.3 > 1000)
```

```
IF (COL.2 = "ABC")
   THEN SKIP
ENDIF
```

Skip for output those records with column 2 equal to "ABC". Select all other records.

```
IF NOT (COL.2 = "ABC")
   THEN SKIP
ENDIF
```

Skip those output records with field column 2 not equal to "ABC". Select all other records. The code at left gives the same result as:

```
SELECTIF(COL.2 = "ABC")
```

```
IF (COL.2 = "A") OR
   (COL.2 = "D")
   THEN SKIP
ENDIF
```

Skip for output those records with column 2 equal to "A" or "D". Select all other records.

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
   THEN SKIP
ENDIF
```

Skip for output those records with column 2 equal to "A" and column 3 greater than 10. Select all other records.

```
IF (COL.3 + Col.4
    > Col.5 * 100)
   THEN SKIP
ENDIF
```

Skip for output those records with column 3 plus column 4 is greater than column 5 times 100. Select all other records.

```
IF (COL.6 = ALL("-"))
   THEN SKIP
ENDIF
```

Skip for output those records with column 6 is equal to all dashes. Select all other records. This example gives the same result as:

```
SKIPIF(COL.6 = ALL("-"))
```

```
IF (COL.6 = REPEAT("-", 13))
   THEN SKIP
ENDIF
```

Skip for output those records with column 6 is equal to 13 dashes. Select all other records. This example gives the same result as:

```
SKIPIF(COL.6 = REPEAT("-", 13))
```

Example logic text **Meaning**

```
IF (COL.6 = "\xFF")
   THEN SKIP
ENDIF
```

Skip for output those records with column 6 is equal to hexadecimal FF. Select all other records. This example gives the same result as:

```
SKIPIF(COL.6 = "\xFF")
```

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN SKIP
ELSE IF (COL.2 = "D")
     THEN SKIP
     ELSE IF (COL.2 = "G")
          THEN SKIP
          ENDIF
     ENDIF
ENDIF
```

Skip for output those records with column 2 equal to "A" and column 3 greater than 10.

In addition, skip for output those records with column 2 equal to "D".

In addition, skip for output those records with column 2 equal to "G".

Select all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) OR
   (COL.2 = "D") OR
   (COL.2 = "G")
   THEN SKIP
ENDIF
```

```
IF (COL.2 = "A") AND
   (COL.3 > 10)
THEN IF (COL.4 + COL.5
         > COL.6)
     THEN SKIP
     ELSE IF (COL.7 = 0)
          THEN SKIP
          ENDIF
     ENDIF
ENDIF
```

Consider those records with column 2 equal to "A" and column 3 greater than 10.

Of the considered records, skip for output those records with column 4 plus column 5 is greater then column 6.

Of the considered records not yet skipped, skip also for output those records with column 7 equal to zero.

Select all other records.

Notice that the logic text at left counts as only one IF statement, because the extra IF statements are nested inside the first.

The code at left can also be written as follows (note the use of brackets to control the evaluation of the conditions):

```
IF (COL.2 = "A" AND COL.3 > 10) AND
   ((COL.4 + COL.5 > COL.6) OR
    (COL.7 = 0))
   THEN SKIP
ENDIF
```

Syntax: WRITE statements

Introduction

WRITE statements are optional in your logic text.

If your logic text contains **no WRITE statements**, then a WRITE occurs automatically. This is called an **implicit** WRITE. An implicit WRITE ensures that all your extract phase records are processed in the format phase (if the view has a format phase). Alternatively, an implicit WRITE ensures all your extract phase records are included in the view output file from the extract phase (if the view has no format phase). An implicit WRITE still occurs if you provide WRITE statements in your logic text.

A WRITE statement in your logic text allows the following:
- Writing records to logical files of your choice.
- Performing a Procedure or UserExit Routine on input records.
- A combination of the above.

There are **two areas of logic text** where you can use a WRITE statement - see the two topics below.

Syntax: WRITE statements in Extract Record Filter

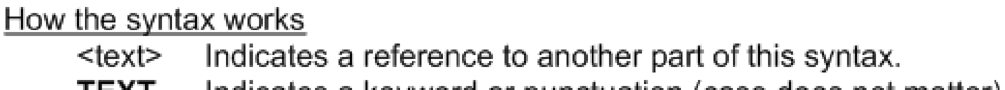
The purpose of WRITE in Extract Record Filter

A WRITE in Extract Record Filter is essentially making a copy of input records in a Logical File. You may write all input records, or you may write only the selected input records.

The purpose of these WRITE statements might be:
- Create an audit trail or backup of input records.
- Create a trace for debugging purposes

Any records written to extract logical files by these WRITE statements are not processed any further in the extract phase or the format phase. The logical files for the written input records are considered view output files.

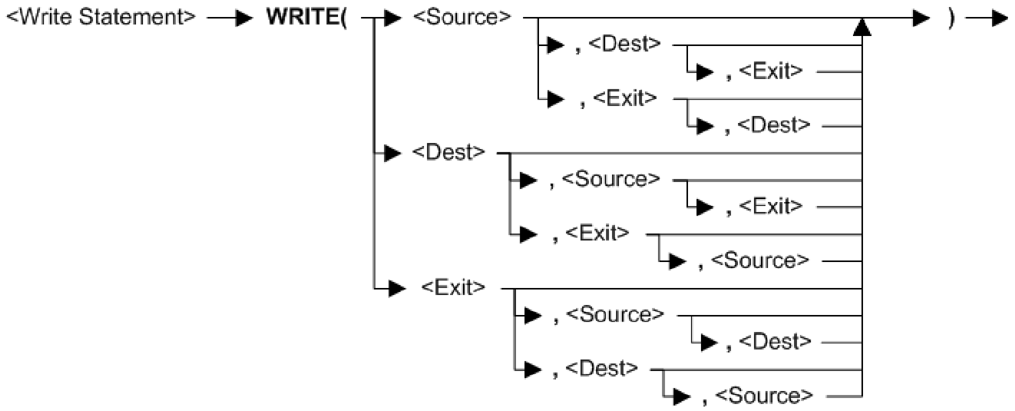
The SOURCE parameter of the WRITE statement must be INPUT, and there is no choice of DESTINATION=EXTRACT. Full details of the syntax are below.

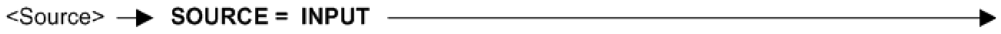


How the syntax works

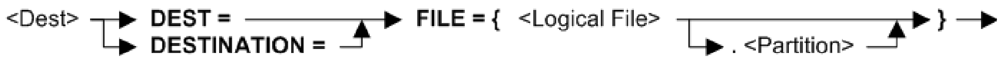
<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax

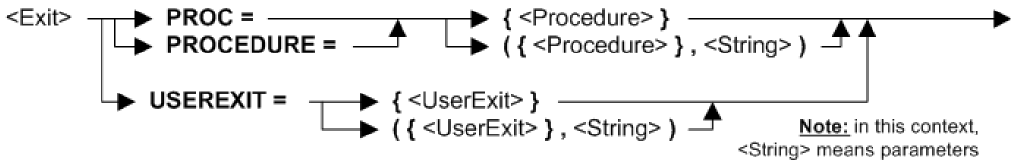




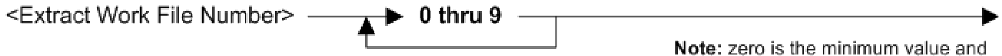
Note: this is required for a WRITE in Extract Record Filter.



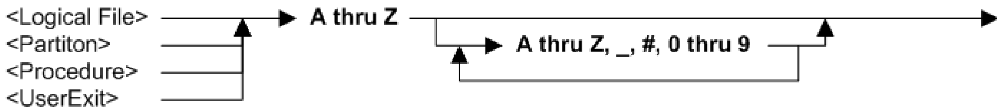
Note: <Partition> is a relevant Physical File name. The default is the first partition if <Partition> is not provided.

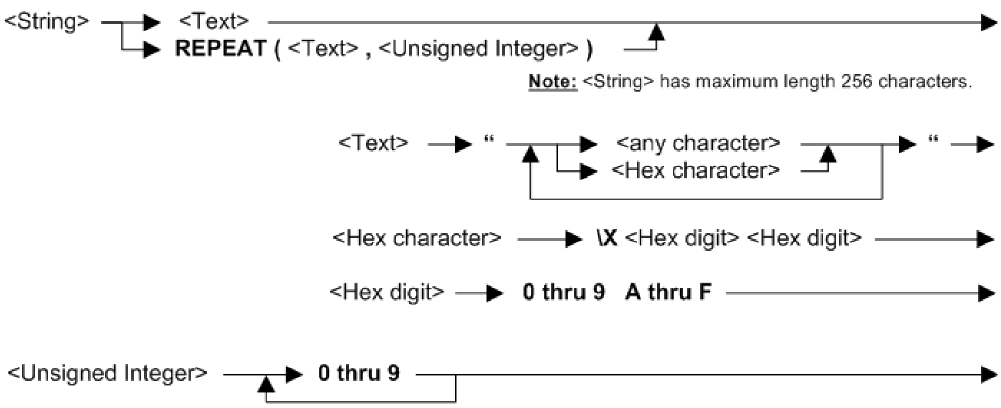


Note: in this context, <String> means parameters to the Procedure or UserExit.



Note: zero is the minimum value and 9999 is the maximum value.





Note: <String> has maximum length 256 characters.

Rules for the syntax

- You can have as many WRITE statements as required in your logic text.
- If the WRITE is **before** any selects or skips in your Extract Record Filter, then the WRITE covers **all input records.**
- If the WRITE is **after** any selects or skips in your Extract Record Filter, then the WRITE covers **only selected input records.**
- Do not include WRITE statements in an IF statement that uses SELECT or SKIP. This means WRITE statements may be before or after (or both before and after) any selects or skips.
- For **Extract Record Filter**, then **SOURCE=INPUT** is a mandatory parameter.
- **PROCEDURE** and **USEREXIT** are both optional parameters. The WRITE statement can have one or the other **but not both**. If you use a **PROCEDURE** or **USEREXIT** parameter, then the DESTINATION may not be necessary.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

What is the difference between PROCEDURE and USEREXIT?

The idea of a procedure is the same as for a user-exit routine.

Both a procedure and a user-exit routine have an executable name on your mainframe. The executable name must conform to mainframe standards, which makes them short and often unfriendly. For example, the executable name might be ABCSYBKUP that performs a backup for the ABC system.

A procedure is the executable name. A user-exit routine allows a longer name to apply to the same executable - for example: "ABC System Backup".

Your WRITE statement can use this parameter:

```
PROCEDURE={ABCSYBKUP}
```

If you define a user-exit routine called "ABC System Backup" that points to ABCSYBKUP, then your WRITE statement can use this USEREXIT parameter:

```
USEREXIT={ABC System Backup}
```

User-exit routines are recommended for all situations like this.

For more, see topic "**User-exit routines overview**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: WRITE in Extract Record Filter

Example logic text	Meaning

`' This WRITE statement is before`
`' any SELECT or SKIP.`
`WRITE (SOURCE = INPUT,`
`      USEREXIT = {Backup} )`
	This WRITE sends all input records to the UserExit Routine called Backup.

`' This WRITE statement is before`
`' any SELECT or SKIP.`
`WRITE (SOURCE = INPUT,`
`      USEREXIT = {Decryption} )`
	This WRITE decrypts all input records and puts the results "in place" so that the input records are processed later in the extract phase.

`SELECTIF ( {product_code}`
`            = "ABC" )`
`WRITE (SOURCE = INPUT,`
`      DEST = FILE = {All_ABC} )`
	The WRITE sends only input records with product_code = "ABC" to the logical file "All_ABC".

`IF ( {product_code} = "ABC" )`
`   THEN SELECT`
`ENDIF`
`WRITE (SOURCE = INPUT,`
`      DEST = FILE = {All_ABC} )`
	This example has the same effect as the previous example. Ensure the IF statement is purely for SELECT statements only.

Syntax: WRITE statements in Extract Column Assignment

What is the purpose of this?

A WRITE in Extract Column Assignment has many options:

- Write a full **copy of the selected input record** to a view output file (SOURCE=INPUT, DEST=FILE=LogicalFile).
- Write **all columns up to that point to a logical file** that is a view output file (SOURCE=DATA, DEST=FILE=LogicalFile)
- Write **all columns up to that point to an extract work file (EXT file)** that can be processed in the format phase (SOURCE=VIEW, DEST=EXT=number). The number for the EXT files must be within a limit (see below). An EXT file is optimized for the format phase and so the structure of an EXT file is different from what is specified in the view. See your IBM representative if you have questions about the structure of an EXT file.
- Write records to s**ome other location** determined by a **procedure or user-exit routine**.
- Some combination of the above.

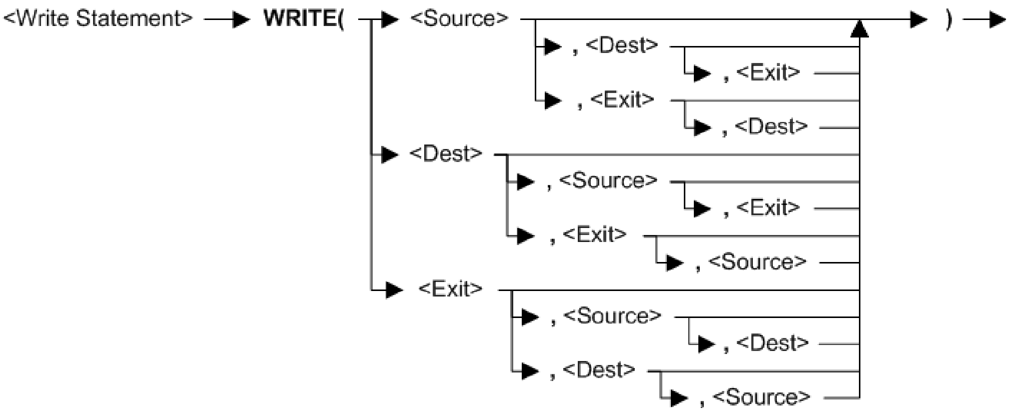
Only the EXT files are processed in the format phase. All other records written in the above choices are written to view output files and not processed any further after the extract phase.

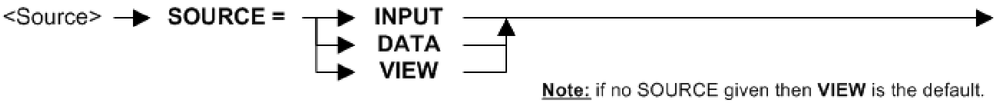
How the syntax works

<text> Indicates a reference to another part of this syntax.

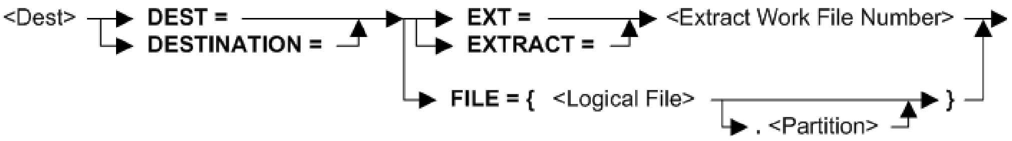
TEXT Indicates a keyword or punctuation (case does not matter)

Syntax

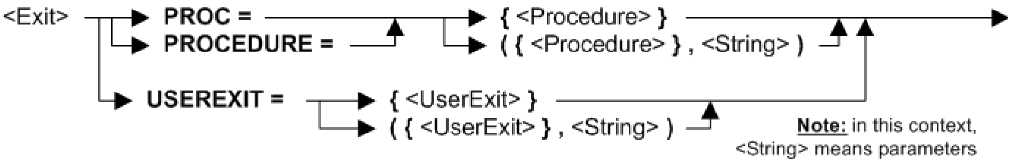




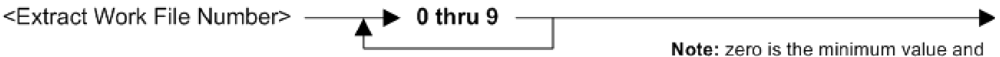
Note: if no SOURCE given then **VIEW** is the default. **VIEW** means write the columns up to that point to an extract work file passed to the format phase. **DATA** means write the columns up to that point to a view output file. **INPUT** means the input records selected in the Extract Record Filter logic text (if present).



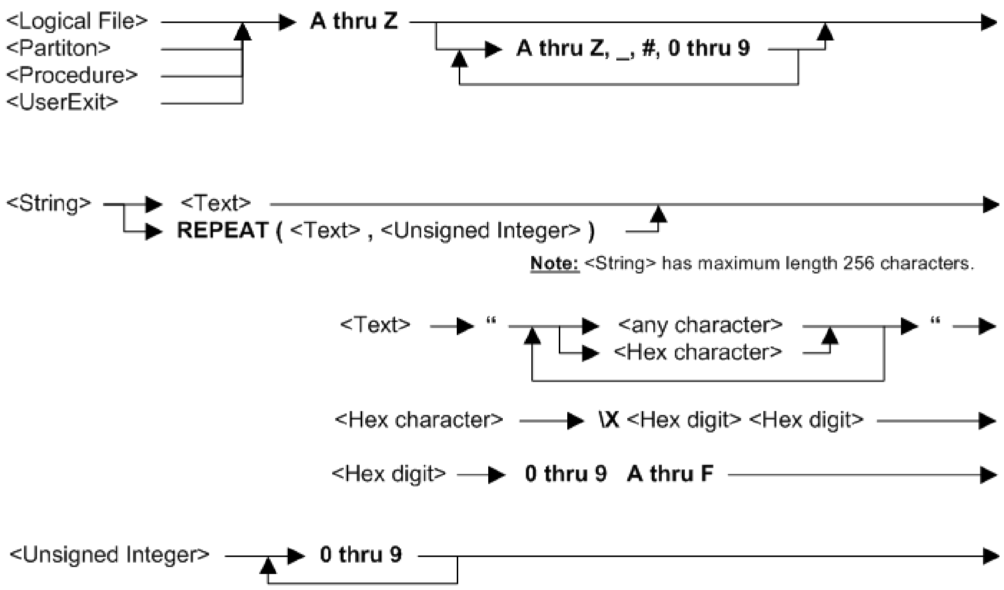
Note: <Partition> is a relevant Physical File name. The default is the first partition if <Partition> is not provided.



Note: in this context, <String> means parameters to the Procedure or UserExit.



Note: zero is the minimum value and 9999 is the maximum value.



Rules

- You can have as many WRITE statements as required in your logic text for a column. The WRITE statements can be positioned anywhere a statement is valid in your logic text.
- If you do not put a SOURCE parameter then **VIEW** is assumed.
- For SOURCE = VIEW or SOURCE = DATA, the WRITE includes **only the columns up to that point (the current column)**. If the WRITE is not in the last column then the record created is truncated compared to all the columns in the view.
- **PROCEDURE** and **USEREXIT** are both optional parameters. The WRITE statement can have one or the other **but not both**. If you use a **PROCEDURE** or **USEREXIT** parameter, then the DESTINATION may not be necessary.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

What is the difference between PROCEDURE and USEREXIT?

The idea of a procedure is the same as for a user-exit routine.

Both a procedure and a user-exit routine have an executable name on your mainframe. The executable name must conform to mainframe standards, which makes them short and often unfriendly. For example, the executable name might be ABCSYBKUP that performs a backup for the ABC system.

A procedure is the executable name. A user-exit routine allows a longer name to apply to the same executable - for example: "ABC System Backup".

Your WRITE statement can use this parameter:

```
PROCEDURE={ABCSYBKUP}
```

If you define a user-exit routine called "ABC System Backup" that points to ABCSYBKUP, then your WRITE statement can use this USEREXIT parameter:

```
USEREXIT={ABC System Backup}
```

User-exit routines are recommended for all situations like this.

For more, see topic "**User-exit routines overview**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: IF with WRITE in Extract Column Assignment

Example logic text	Meaning

`IF (ISNUMERIC({field4}) AND`
`({field5} > {field6} * 10) AND`
`(ISNOTSPACES{field7}`
`THEN WRITE (SOURCE=DATA,`
`USEREXIT={DB2_Update})`
`ENDIF`
	If field4 is numeric and field5 is greater than field6 times 10 and field7 is not spaces, then call the user-exit routine DB2_Update for the columns up to the current point. This effectively does a writes to a DB2 table the columns in that record up to the current point.

`IF (ISNOTNULL({field3}) AND`
`({field2} = {field1} + {field5}`
`THEN WRITE (SOURCE=INPUT,`
`DEST=FILE=`
`{LogicalFile3})`
`ENDIF`
	If field3 is not nulls and field2 equals field1 plus field 5 then write the entire input record to LogicalFile3. All columns in the input record are included, no matter what column contains this logic text.

`IF (DAYSBETWEEN({field12},{field15})`
`> 10) AND`
`(ISFOUND({Lookup3;$SYM="A"}))`
`THEN WRITE (SOURCE=VIEW,`
`DEST=EXT=03)`
`ENDIF`
	If field12 and field15 are more than 10 days apart and the lookup path Lookup3 works with symbol SYM set to "A", then write the columns up to the current point to extract work file (EXT) 3. This assumes that the control record for this environment has a Maximum Extract File Number that is at least 3, or any overwrite of the VDP has also set this parameter to at least 3.

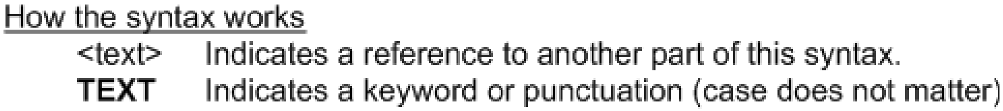
Chapter 3. Syntax: functions

Syntax: function ALL

How do I use ALL?

If you provide a text string then ALL can check if all of a field value is that text string (repeated). ALL is different from REPEAT because REPEAT has a fixed number of repetitions, whereas ALL is flexible and compares with fields of different lengths.

ALL can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

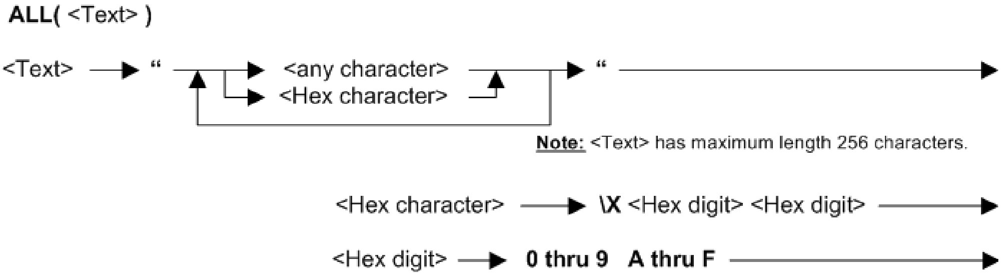


How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax



Rules for the syntax

ALL can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: ALL function in Extract Record Filter

Example logic text	Meaning

`IF NOT ({field1} = ALL("-"))`
`   THEN SELECT`
`ENDIF`
	Select for output those records with field1 is not equal to all dashes. Skip all other records. This example gives the same result as:
	`SKIPIF({field1} = ALL("-"))`

Example logic text	Meaning

`IF ({field2} = ALL("-"))`
`   THEN SKIP`
`ENDIF`
	Skip for output those records with field2 is equal to all dashes. Select all other records. This example gives the same result as:
	`SKIPIF({field2} = ALL("-"))`

Examples: ALL function in Extract Column Assignment

Example logic text	Meaning

`IF (field3} = "Total")`
`   THEN COLUMN = ALL("-")`
`ENDIF`
	If field3 is "Total" then set the current column to all dashes.

`IF (field4} = ALL("-"))`
`   THEN COLUMN = (field5} + (field6}`
`ENDIF`
	If field4 is all dashes, then set the current column to a total of fields 5 and 6.

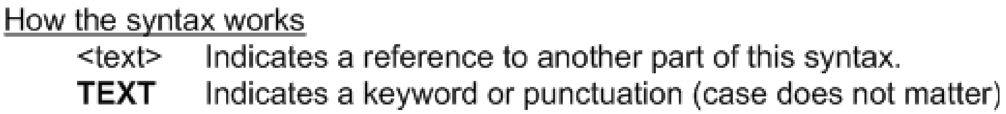
Syntax: function BATCHDATE

How do I use BATCHDATE?

The BATCHDATE returns a date in CCYYMMDD format that is when the view is run. The group of views all running in the same batch in the Performance Engine all use the same BATCHDATE.

The parameter for BATCHDATE is a number of days to add or delete from the default BATCHDATE. For example, BATCHDATE(-5) provides the day five days before the date the view is run.

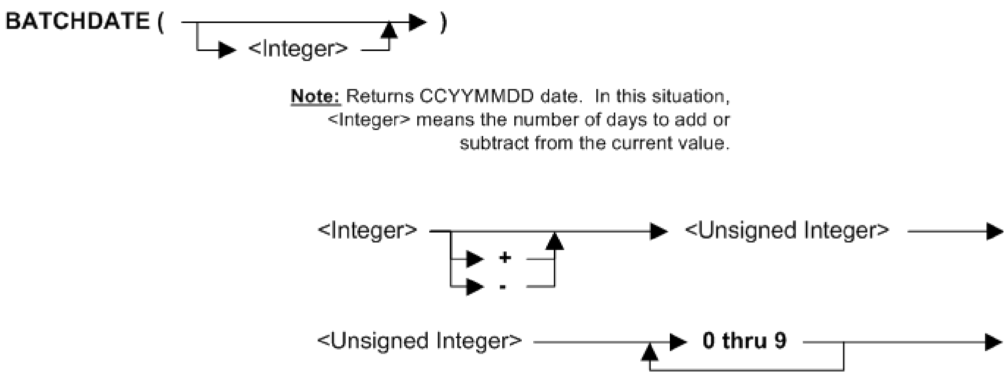
BATCHDATE can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMMDD date. In this situation, <Integer> means the number of days to add or subtract from the current value.

Rules for the syntax

BATCHDATE can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: BATCHDATE function in Extract Record Filter

Example logic text	Meaning

`IF ({field1} < BATCHDATE(-7))`
`   THEN SKIP`
`ENDIF`
	Skip any input records where field1 is more than 7 days before the date of running this view. Select all other records. This example can also be written:
	`SKIPIF({field1} < BATCHDATE(-7))`

Examples: BATCHDATE function in Extract Column Assignment

Example logic text	Meaning
`COLUMN = BATCHDATE()`	Set the current column to the date that this view is run.
`COLUMN = BATCHDATE(-7)`	Set the current column to the date that is 7 days before the view is run.
`COLUMN = {Lookup3.Field4,BATCHDATE(-32))`	Set the current column to Field4 found by Lookup3 as of 32 days before the view is run.

Syntax: function CURRENT

See PRIOR

Function CURRENT is always used when using function PRIOR - so see topic "**Syntax: function PRIOR**".

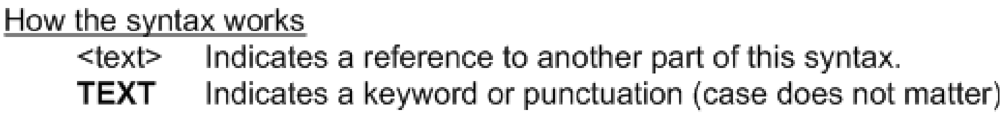
That topic is elsewhere in this PDF - see the table of contents.

Syntax: function DATE

How do I use DATE?

DATE is used whenever you want to specify some date. You can use DATE to set a value, or as part of a comparison.

DATE can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

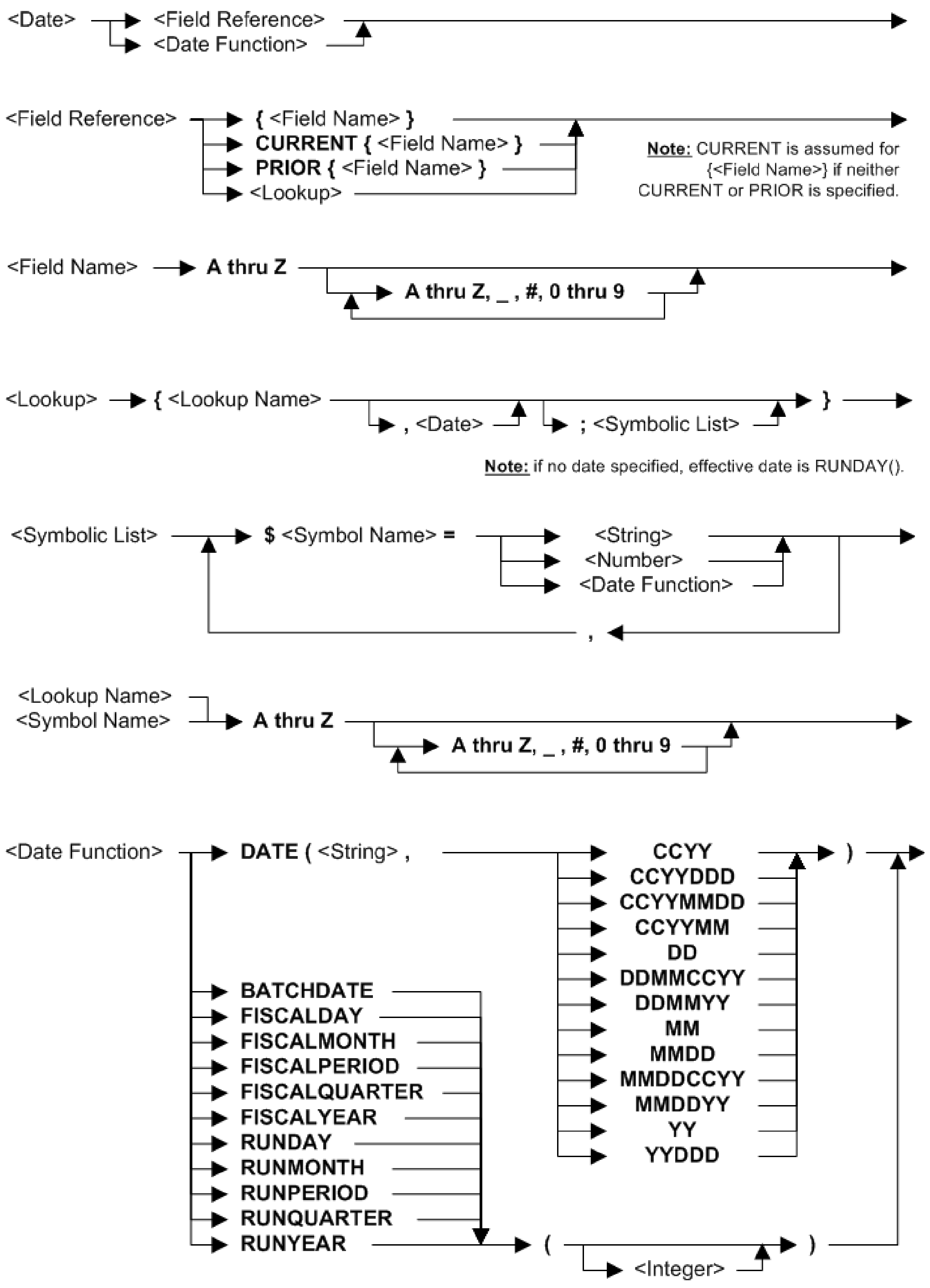


How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

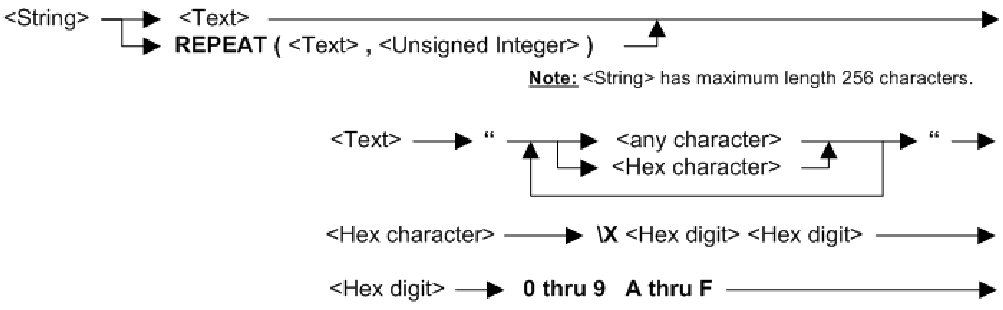
Syntax

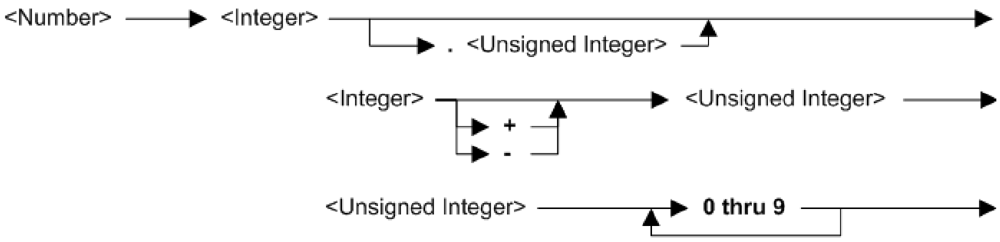


Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.

Note: if no date specified, effective date is RUNDAY().

Note: In this situation, <Integer> means the number of units to add or subtract from the current value.


<String>
<Text>
REPEAT (<Text> , <Unsigned Integer>)
Note: <String> has maximum length 256 characters.
<Text>
“
<any character>
<Hex character>
“
<Hex character>
\X <Hex digit> <Hex digit>
<Hex digit>
0 thru 9 A thru F



Rules for the syntax

DATE can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: DATE function in Extract Record Filter

Example logic text	Meaning

`IF ({field1} = BATCHDATE())`
`   THEN SELECT`
`ENDIF`
	Select those input records where field1 is in the date the view is run. The code at left can also be written as:
	`SELECTIF({field1} = BATCHDATE())`

Examples: DATE function in Extract Column Assignment

Example logic text	Meaning
`COLUMN = DATE("20111201",CCYYMMDD)`	Set the current column to a date of December 1, 2011 in CCYYMMDD format.

Syntax: function DAYSBETWEEN

How do I use DAYSBETWEEN?

Use DAYSBETWEEN to compare dates and give an different in days.

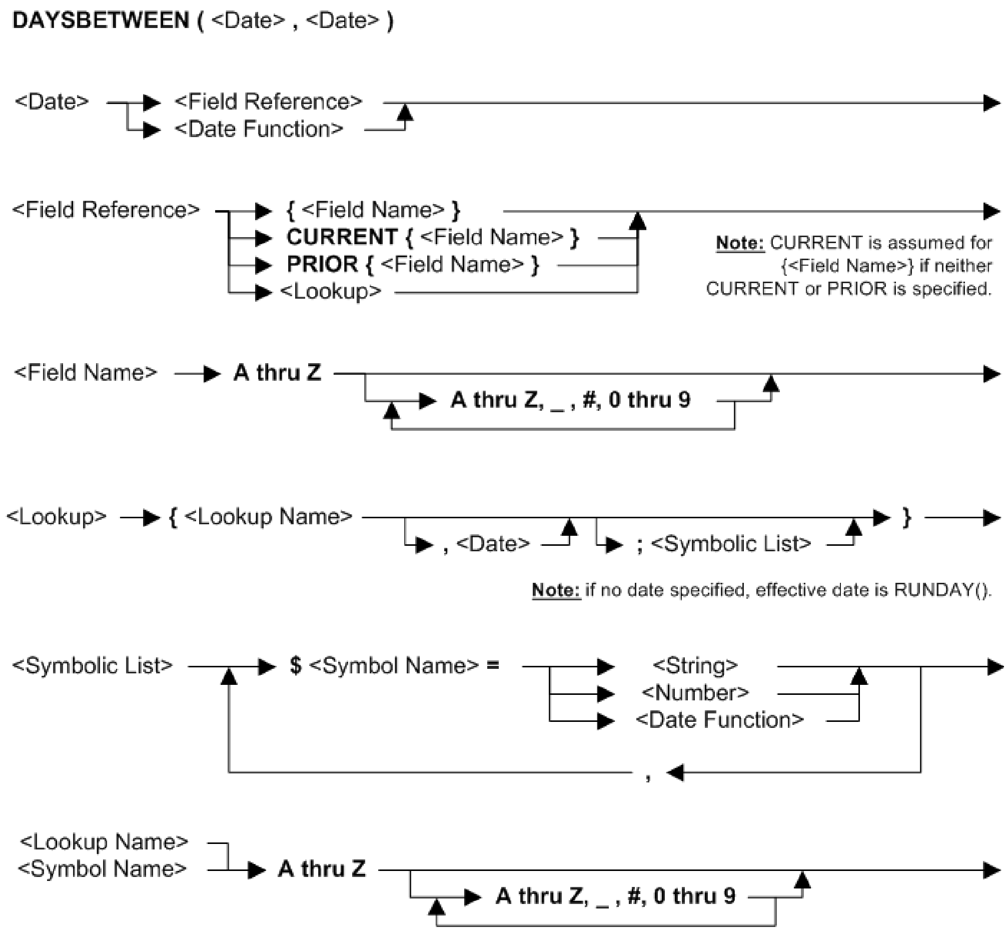
DAYSBETWEEN can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

How the syntax works

<text> Indicates a reference to another part of this syntax.

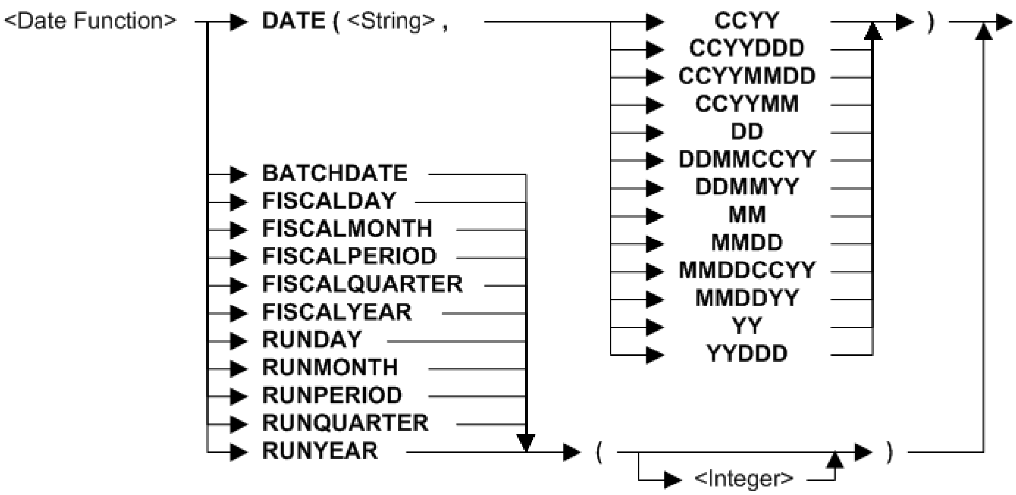
TEXT Indicates a keyword or punctuation (case does not matter)

Syntax

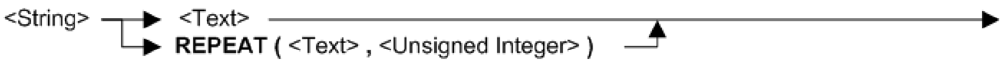


Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.

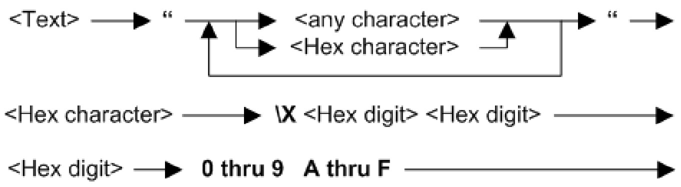
Note: if no date specified, effective date is RUNDAY().

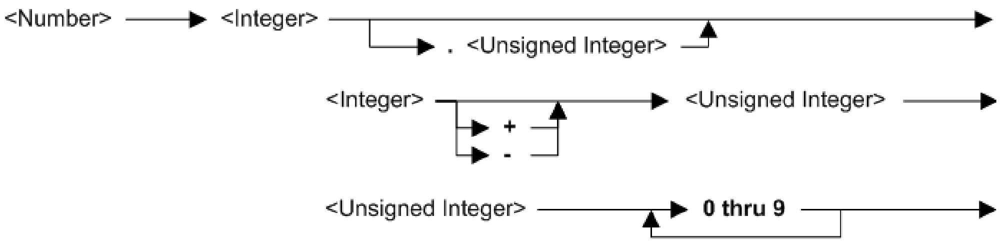


Note: In this situation, <Integer> means the number of units to add or subtract from the current value.



Note: <String> has maximum length 256 characters.





Rules for the syntax

DAYSBETWEEN can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: function DAYSBETWEEN in Extract Record Filter

Example logic text	Meaning

IF (DAYSBETWEEN({field1},{field2})
    > 7)
   THEN SELECT
ENDIF
	Select only records where there are more than 7 days between field1 and field2, and skip all other records. This example can also be written:
	SELECTIF(DAYSBETWEEN({field1},{field2})
	          > 7)

Examples: function DAYSBETWEEN in Extract Column Assignment

Example logic text	Meaning
COLUMN = DAYSBETWEEN({BUY_DATE},{SHIP_DATE})	Set the current column to the days between the transaction date and the shipping date.

IF (DAYSBETWEEN({BUY_DATE},{SHIP_DATE}) > 10)
   THEN COLUMN = {SHIP_DATE}
   ELSE COLUMN = {BUY_DATE}
ENDIF
	If there are more than 10 days between the transaction date and the shipping date, then set the current column to the shipping date, otherwise set the current column to the transaction date.

IF (DAYSBETWEEN({BUY_DATE},{SHIP_DATE}) > 30)
   THEN WRITE(SOURCE=VIEW,DEST=EXT=03)
ENDIF
	Write to extract 3 those records where there are more than 30 days between the transaction date and the shipping date.

Syntax: function FISCALDAY

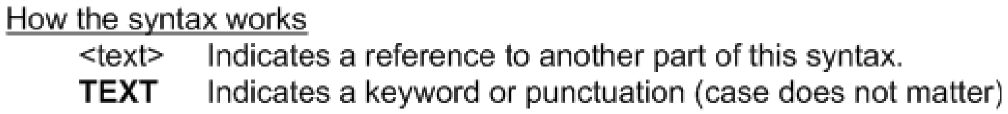
How do I use FISCALDAY?

FISCALDAY returns a day based on the fiscal values in the control record for the environment for a view. This means that different views in the same batch run can have different fiscal dates because they come from different environments. By comparison, RUNDAY is the same for all views in a batch.

The FISCALDAY returns a date in CCYYMMDD format that is appropriate for the environment of that view.

The parameter for FISCALDAY is a number of days to add or delete from the default FISCALDAY. For example, FISCALDAY(-5) provides the day that is five days before the date the view is run.

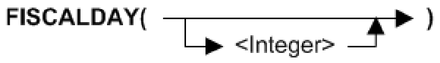
FISCALDAY can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



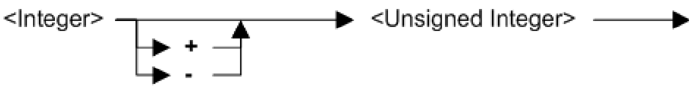
How the syntax works

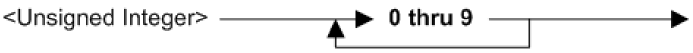
<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMMDD date. In this situation, <Integer> means the number of days to add or subtract from the current value.





Rules for the syntax

FISCALDAY can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: FISCALDAY function in Extract Record Filter

Example logic text	Meaning

`IF ({field1} < FISCALDAY(-7))`
`   THEN SKIP`
`ENDIF`
	Skip any input records where field1 is more than 7 fiscal days before the date of running this view, and select all other records. Fiscal days means that if the date of running this view is fiscal day five of a fiscal year, then only records from fiscal days one to five are selected. This example assumes that field1 is a fiscal day number. The code at left can also be written as:
	`SKIPIF({field1} < FISCALDAY(-7))`

Examples: FISCALDAY function in Extract Column Assignment

Example logic text	Meaning
`COLUMN = FISCALDAY()`	Set the current column to the fiscal day number of the day the view is run.

Syntax: function FISCALMONTH

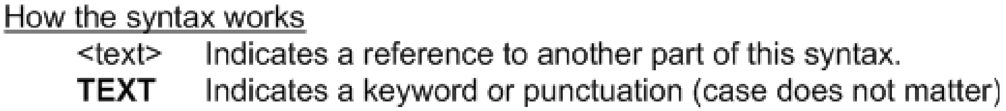
How do I use FISCALMONTH?

FISCALMONTH returns a month based on the Fiscal Parameters in the control record for the environment for a view. This means that different views running in the same batch can have different Fiscal dates because they come from different environments. By comparison, RUNDAY is the same for all views in a batch.

The FISCALMONTH returns a date in CCYYMM format that is appropriate for the environment of that view.

The parameter for FISCALMONTH is a number of months to add or delete from the default FISCALMONTH. For example, FISCALMONTH(-5) provides the month that is five months before the date the view is run.

FISCALMONTH can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

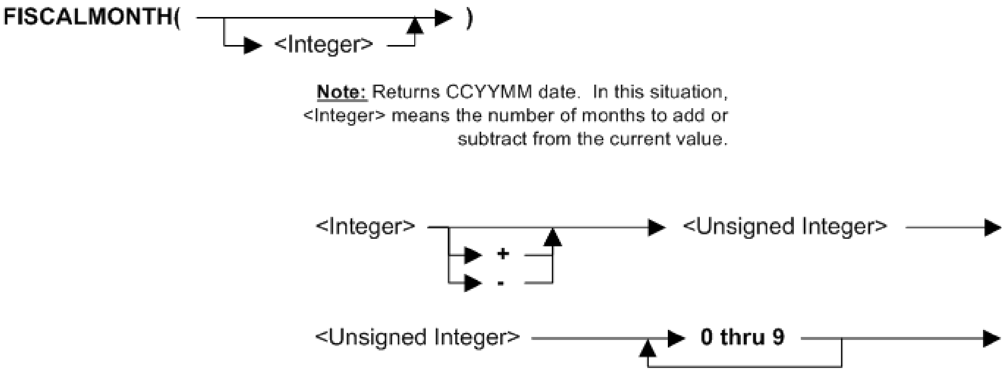


How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMM date. In this situation, <Integer> means the number of months to add or subtract from the current value.

Rules for the syntax

FISCALMONTH can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: FISCALMONTH function in Extract Record Filter

Example logic text	**Meaning**

`IF ({field2} >= FISCALMONTH(-1))`
`   THEN SELECT`
`ENDIF`
	Select any input records where field2 is the previous fiscal month or later, and skip all other records. Fiscal months means that if the date of running this view is fiscal month one, then only records from fiscal month one are selected. The example at left assumes that field2 is a fiscal month number. The code at left can also be written as:
	`SELECTIF({field2} >= FISCALMONTH(-1))`

Examples: FISCALMONTH function in Extract Column Assignment

Example logic text	**Meaning**
`COLUMN = FISCALMONTH()`	Set the current column to the current fiscal month number.

Syntax: function FISCALPERIOD

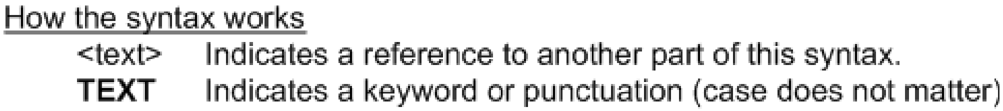
How do I use FISCALPERIOD?

FISCALPERIOD is a similar concept to FISCALMONTH. The difference is that there can be 13 periods in a year rather than 12. FISCALPERIOD returns a period based on the Fiscal Parameters in the control record for the environment for a view. This means that different views running in the same batch can have different Fiscal dates because they come from different environments. By comparison, RUNDAY is the same for all views in a batch.

The FISCALPERIOD returns a date in CCYYMM format that is appropriate for the environment of that view.

The parameter for FISCALPERIOD is a number of periods to add or delete from the default FISCALPERIOD. For example, FISCALPERIOD(-5) provides the month that is five periods before the date the view is run.

FISCALPERIOD can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

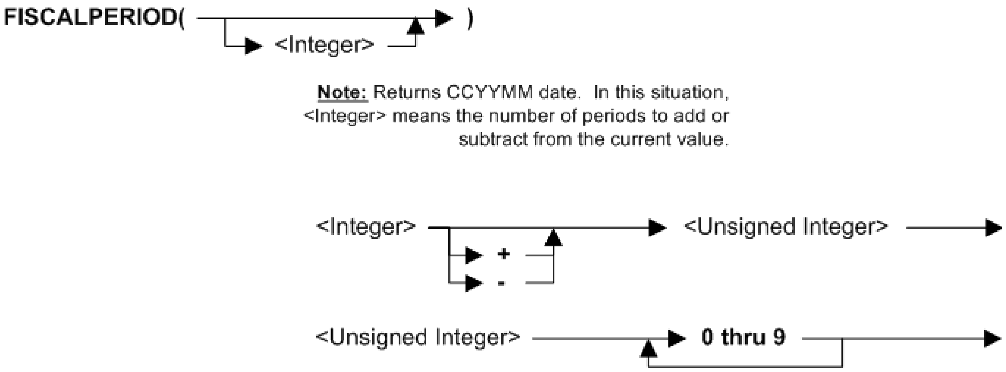


How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMM date. In this situation, <Integer> means the number of periods to add or subtract from the current value.

Rules for the syntax

FISCALPERIOD can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: FISCALPERIOD function in Extract Record Filter

Example logic text	Meaning

IF ({field3} >= FISCALPERIOD(-1))
   THEN SELECT
ENDIF
	Select any input records where field3 is the previous fiscal period or later, and skip all other records. The example at left assumes that field3 is a fiscal period number. The code at left can also be written as:
	SELECTIF({field3} >= FISCALPERIOD(-1))

Examples: FISCALPERIOD function in Extract Column Assignment

Example logic text	Meaning
COLUMN = FISCALPERIOD()	Set the current column to the current fiscal period number.

Syntax: function FISCALQUARTER

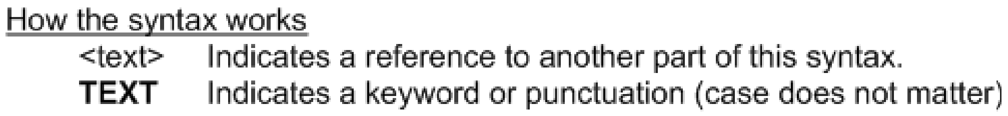
How do I use FISCALQUARTER?

FISCALQUARTER returns a month (at a quarter start) based on the Fiscal Parameters in the control record for the environment for a view. This means that different views running in the same batch can have different Fiscal dates because they come from different environments. By comparison, RUNDAY is the same for all views in a batch.

The FISCALQUARTER returns a date in CCYYMM format that is appropriate for the environment of that view.

The parameter for FISCALQUARTER is a number of quarters to add or delete from the default FISCALQUARTER. For example, FISCALQUARTER(-5) provides the month that is five quarters before the date the view is run.

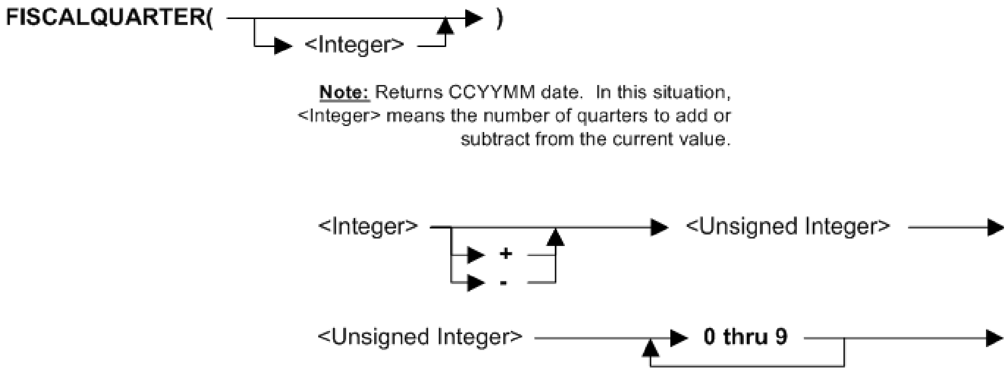
FISCALQUARTER can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMM date. In this situation, <Integer> means the number of quarters to add or subtract from the current value.

Rules for the syntax

FISCALQUARTER can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Syntax: function FISCALYEAR

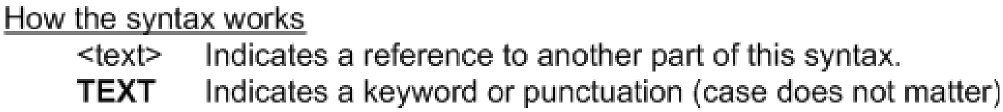
How do I use FISCALYEAR?

FISCALYEAR returns a year based on the Fiscal Parameters in the control record for the environment for a view. This means that different views running in the same batch can have different Fiscal dates because they come from different environments. By comparison, RUNDAY is the same for all views in a batch.

The FISCALYEAR returns a date in CCYY format that is appropriate for the environment of that view.

The parameter for FISCALYEAR is a number of years to add or delete from the default FISCALYEAR. For example, FISCALYEAR(-5) provides the year that is five years before the date the view is run.

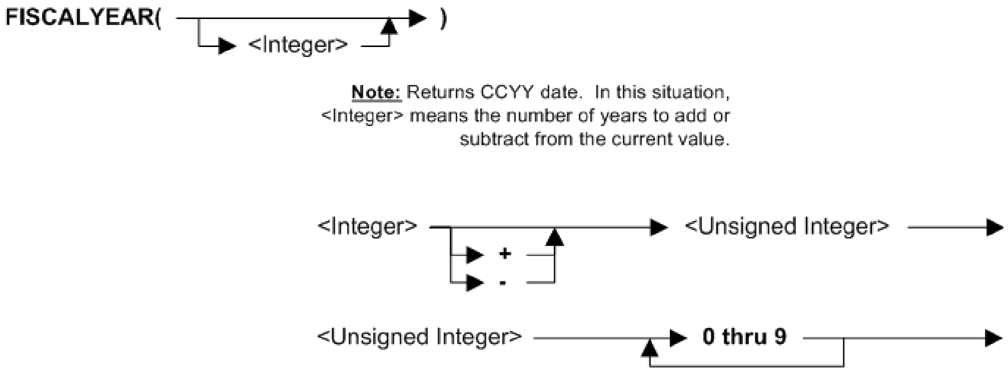
FISCALYEAR can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYY date. In this situation, <Integer> means the number of years to add or subtract from the current value.

Rules for the syntax

FISCALYEAR can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: FISCALYEAR function in Extract Record Filter

Example logic text	Meaning

`IF ({field4} = FISCALYEAR(-1))`
`   THEN SELECT`
`ENDIF`
	Select any input records where field4 is the previous fiscal year, and skip all other records. The example at left assumes that field4 is a fiscal year number. The code at left can also be written as:
	`SELECTIF({field4} = FISCALYEAR(-1))`

Examples: FISCALYEAR function in Extract Column Assignment

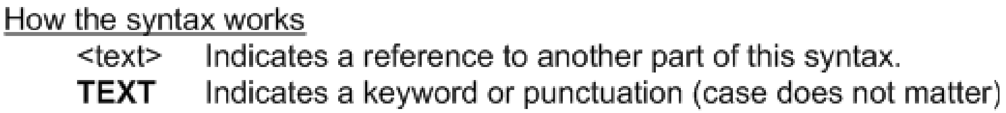
Example logic text	Meaning
`COLUMN = FISCALYEAR()`	Set the current column to the current fiscal year number.

Syntax: function ISFOUND

How do I use ISFOUND?

If you provide a lookup path then ISFOUND returns true if the lookup path is successful for the current input record, and false if the lookup path fails.

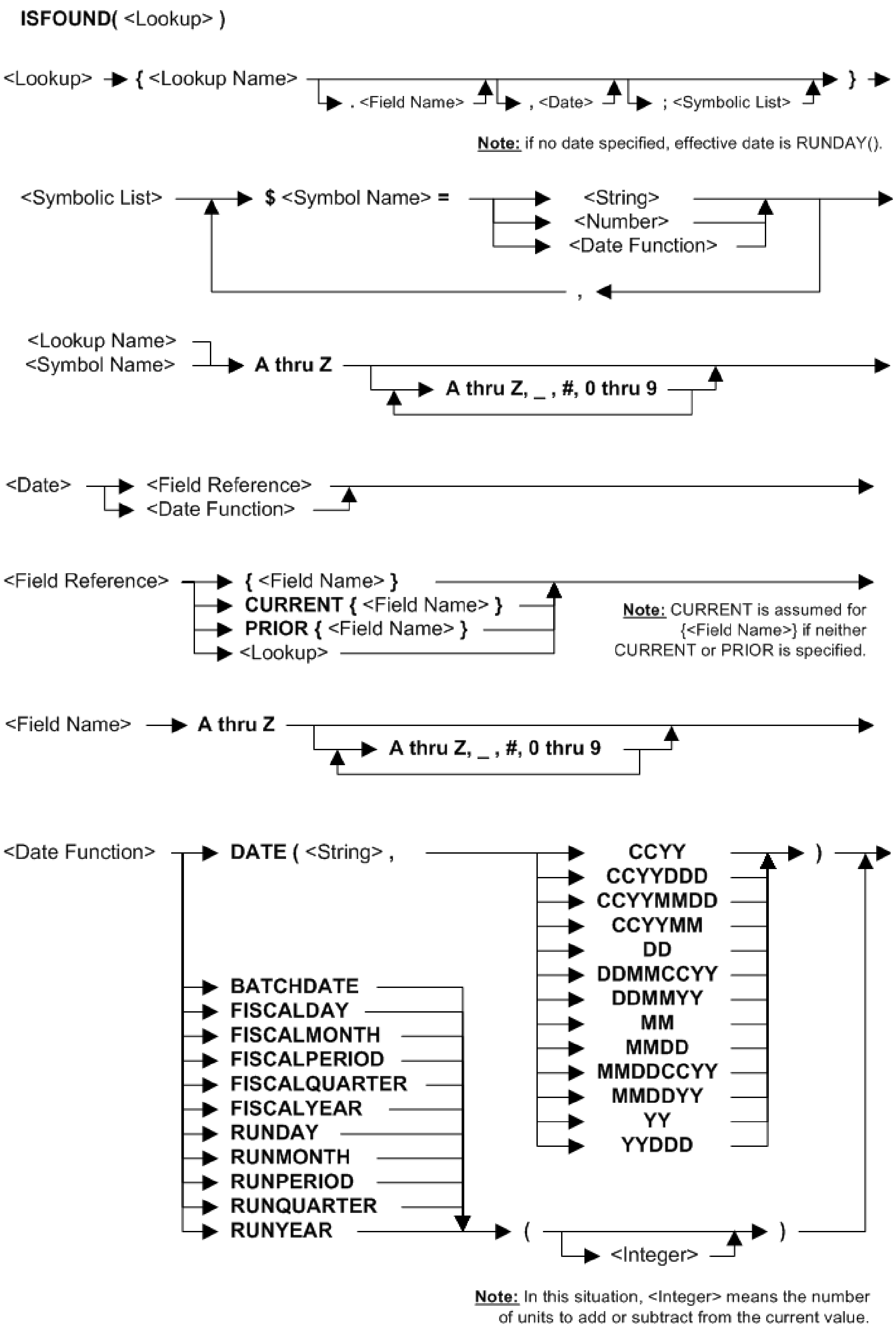
ISFOUND can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

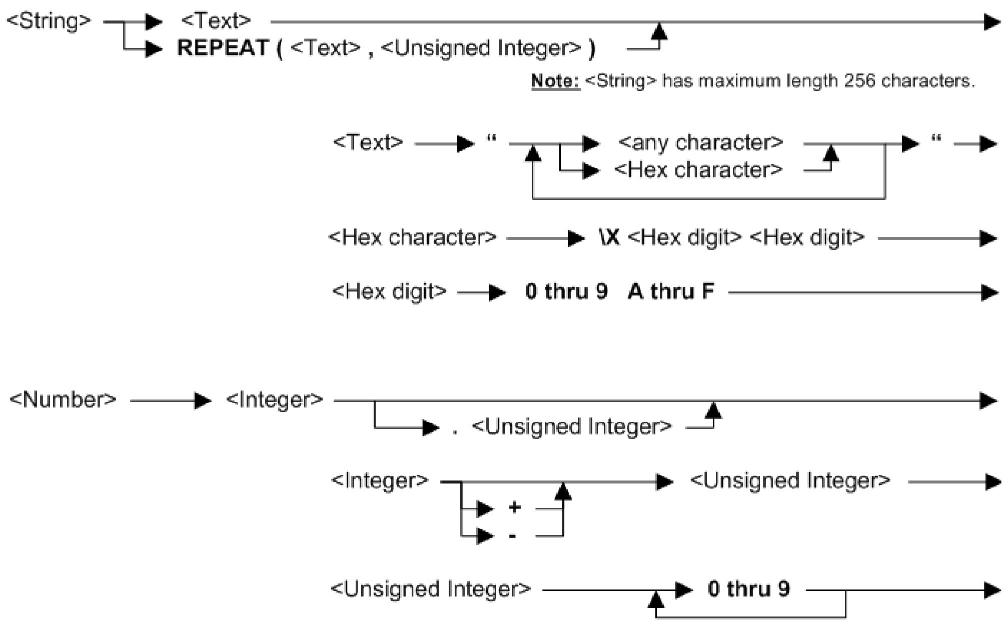
Syntax



Note: if no date specified, effective date is RUNDAY().

Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.

Note: In this situation, <Integer> means the number of units to add or subtract from the current value.



Rules for the syntax

ISFOUND can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

If **no effective date** is specified, ISFOUND uses RUNDAY( ) - see topic "**Syntax: function RUNDAY**". That topic is elsewhere in this PDF - see the table of contents.

See also topic "**Rules for all logic text**". That topic is elsewhere in this PDF - see the table of contents.

Examples: ISFOUND function in Extract Record Filter

Example logic text	Meaning

`IF ISFOUND({Lookup2})`
`   THEN SELECT`
`ENDIF`
	Select all input records where lookup path Lookup2 successfully finds a target record, and skip all other records. This example is the same as:
	`SELECTIF(ISFOUND({Lookup2})`

Examples: ISFOUND function in Extract Column Assignment

Example logic text	Meaning

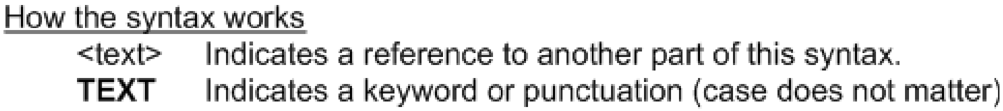
`IF ISFOUND({Lookup2})`
`   THEN COLUMN = {Lookup2}`
`   ELSE COLUMN = " "`
`ENDIF`
	If the lookup path Lookup2 uses the current record to successfully find a target record, then set the current column to the lookup field, otherwise set the current column to blank.

Syntax: function ISNOTFOUND

How do I use ISNOTFOUND?

If you provide a lookup path then ISNOTFOUND returns true if the lookup path fails for the current input record and false if the lookup path is successful .

ISNOTFOUND can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

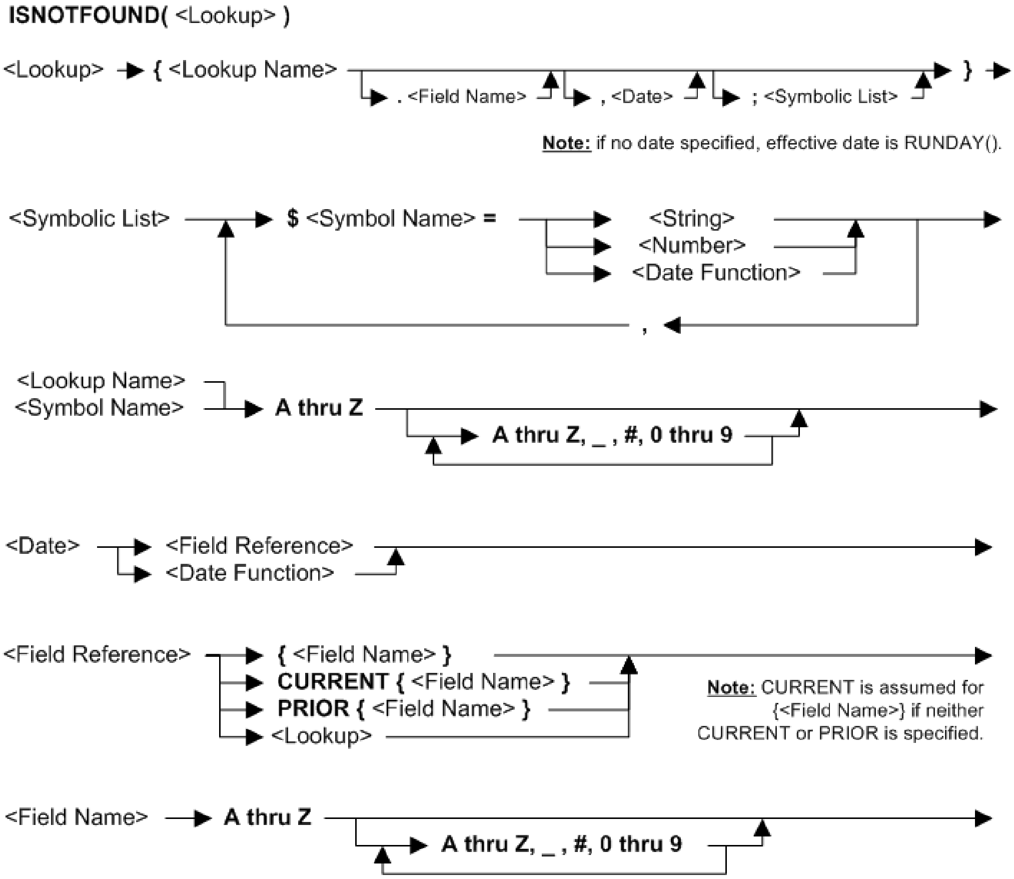


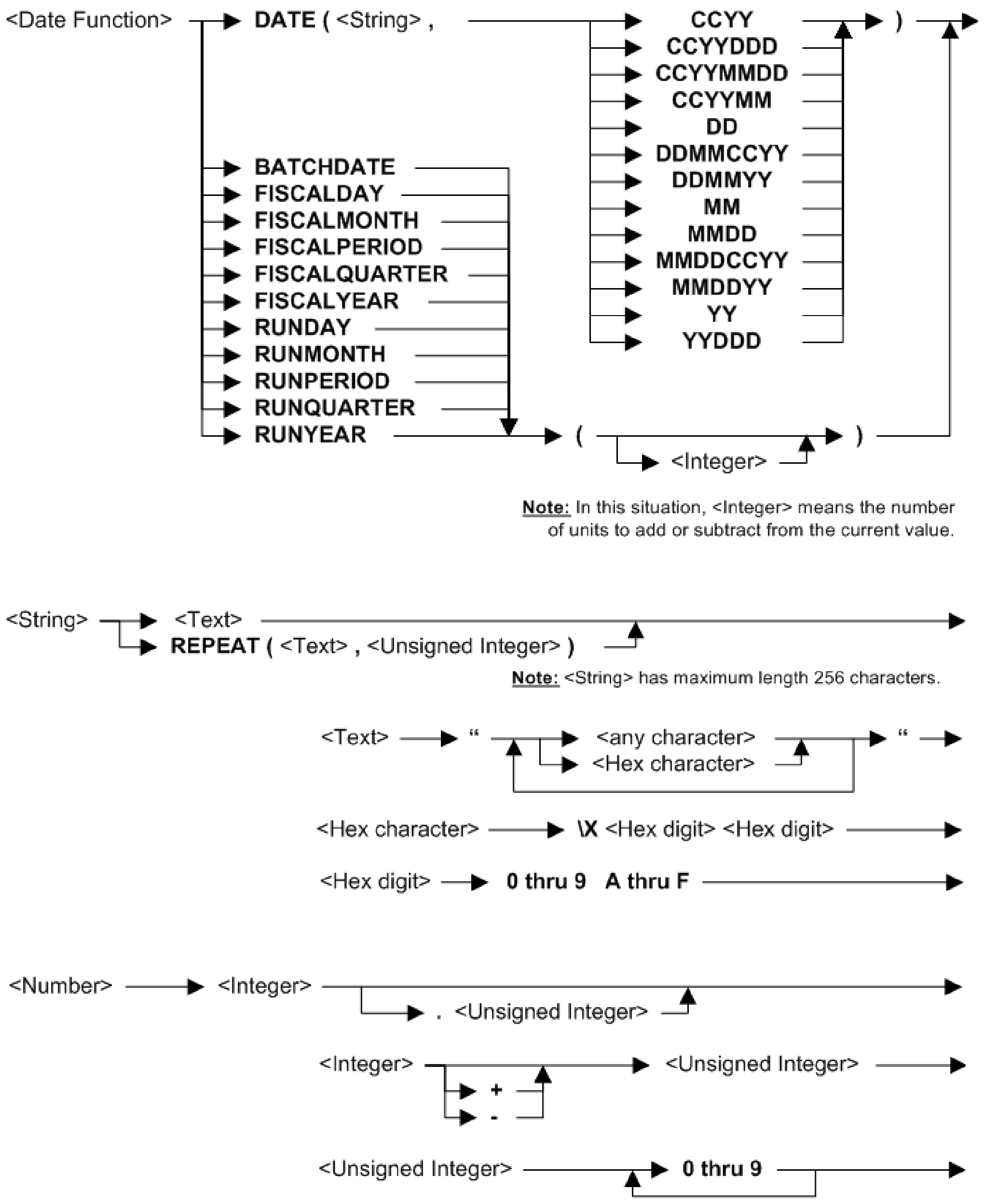
How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax





Note: In this situation, <Integer> means the number of units to add or subtract from the current value.

Note: <String> has maximum length 256 characters.

Rules for the syntax

ISNOTFOUND can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

If **no effective date** is specified, ISNOTFOUND uses RUNDAY( ) - see topic "**Syntax: function RUNDAY**". That topic is elsewhere in this PDF - see the table of contents.

See also topic "**Rules for all logic text**". That topic is elsewhere in this PDF - see the table of contents.

Examples: ISNOTFOUND function in Extract Record Filter

Example logic text	Meaning

`IF ISNOTFOUND({Lookup3})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where the lookup path Lookup3 does not successfully find a target record, and select all other records. This example is the same as:
	`SKIPIF(ISNOTFOUND({Lookup3})`

Examples: ISNOTFOUND function in Extract Column Assignment

Example logic text	Meaning

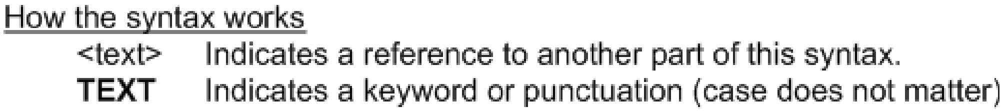
`IF ISNOTFOUND({Lookup1})`
`   THEN COLUMN = "PROBLEM"`
`   ELSE COLUMN = " "`
`ENDIF`
	If the lookup path Lookup1 uses the current record and does not successfully find a target record, then set the current column to "PROBLEM", otherwise set the current column to blank.

Syntax: function ISNOTNULL

How do I use ISNOTNULL?

If you provide an input field or lookup path then ISNOTNULL returns true if the input field or lookup path field is some other than null values, and false if the value is null values.

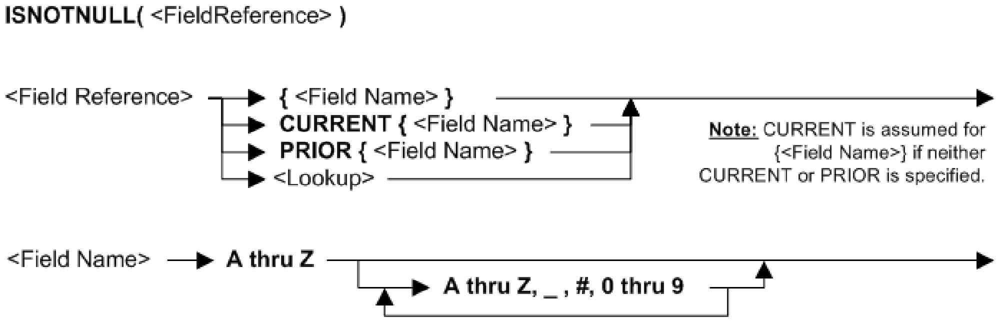
ISNOTNULL can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

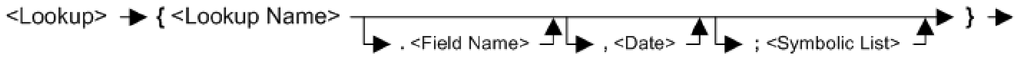


How the syntax works

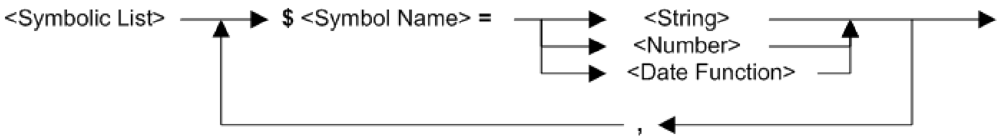
<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

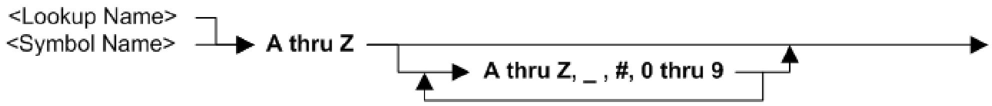
Syntax

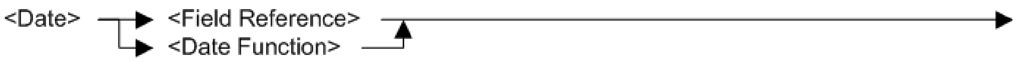

ISNOTNULL(<FieldReference> **)**
<Field Reference>
{ <Field Name> }
CURRENT { <Field Name> }
PRIOR { <Field Name> }
<Lookup>
Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.
<Field Name>
A thru Z
A thru Z, _ , #, 0 thru 9

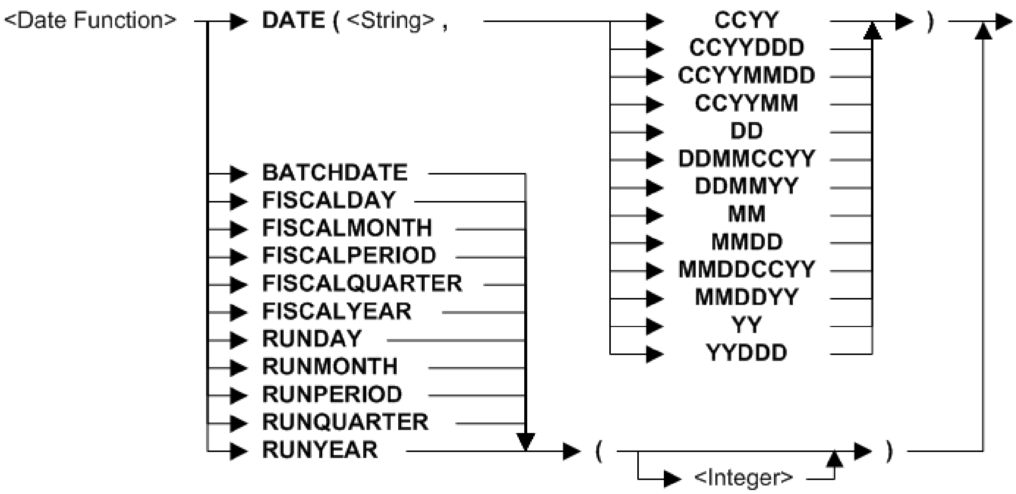


Note: if no date specified, effective date is RUNDAY().

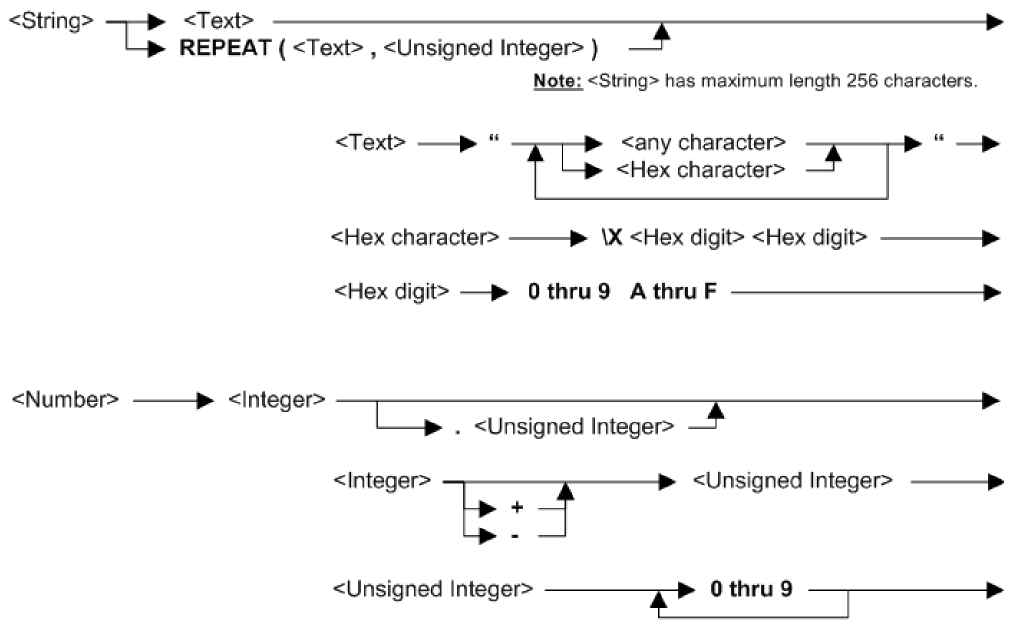








Note: In this situation, <Integer> means the number of units to add or subtract from the current value.



Rules for the syntax

ISNOTNULL can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

If this functions checks a lookup, and **no effective date** is specified, the lookup uses RUNDAY( ) - see topic "**Syntax: function RUNDAY**". That topic is elsewhere in this PDF - see the table of contents.

See also topic "**Rules for all logic text**". That topic is elsewhere in this PDF - see the table of contents.

Examples: ISNOTNULL function in Extract Record Filter

Example logic text	Meaning

`IF ISNOTNULL({field3})`
`   THEN SELECT`
`ENDIF`
	Select all input records where field3 does not contain null values, and skip all other records. This example is the same as:
	`SELECTIF(ISNOTNULL({field3})`

`IF ISNOTNULL({Lookup1})`
`   THEN SELECT`
`ENDIF`
	Select all input records where the lookup field for lookup path Lookup1 does not contain null values, and skip all other records. This example is the same as:
	`SELECTIF(ISNOTNULL({Lookup1})`

Examples: ISNOTNULL function in Extract Column Assignment

Example logic text	Meaning

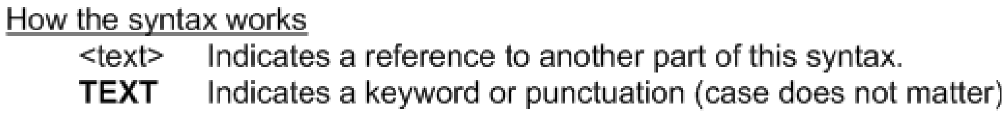
`IF ISNOTNULL({field4})`
`   THEN COLUMN = {field4}`
`   ELSE COLUMN = "NOT AVAILABLE"`
`ENDIF`
	If field4 for the current record does not contain null values, then set the current column to field4, otherwise set the current column to "NOT AVAILABLE".

Syntax: function ISNOTNUMERIC

How do I use ISNOTNUMERIC?

If you provide an input field or lookup path then ISNOTNUMERIC returns true if the input field or lookup path field not a numeric value, and false if the value is numeric.

ISNOTNUMERIC can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

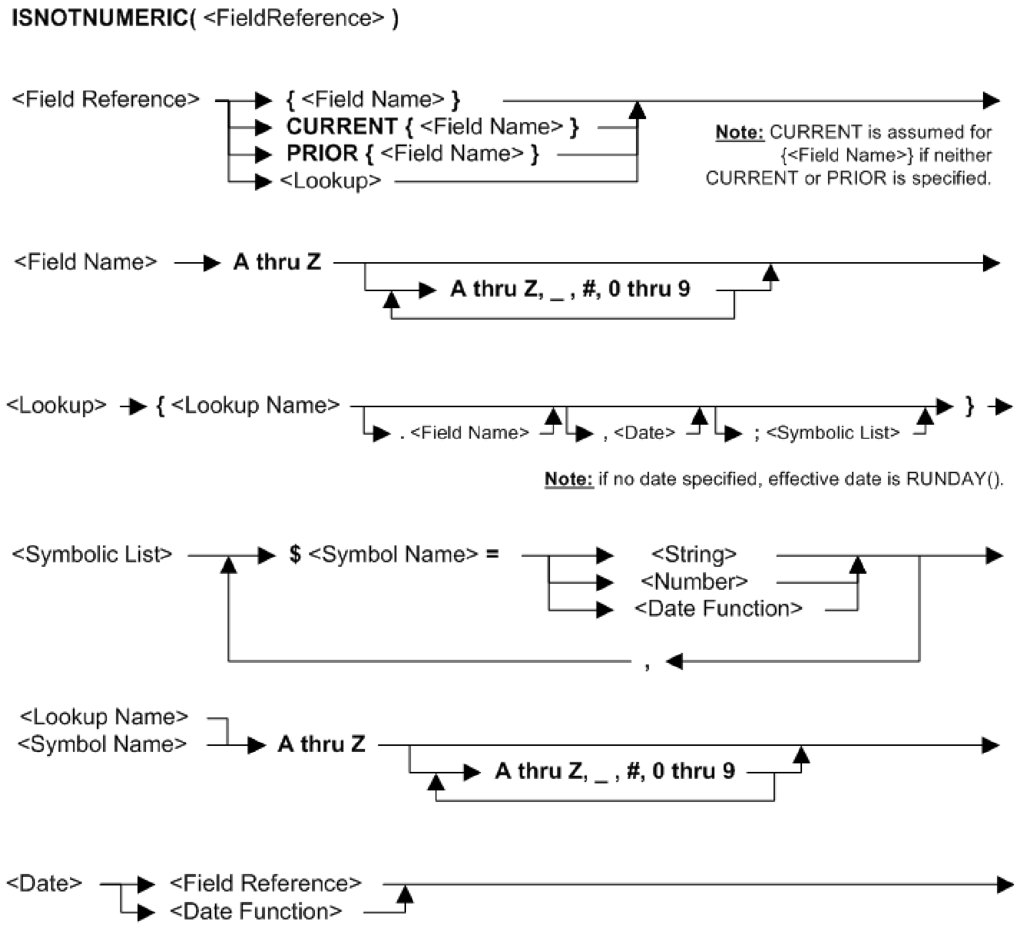


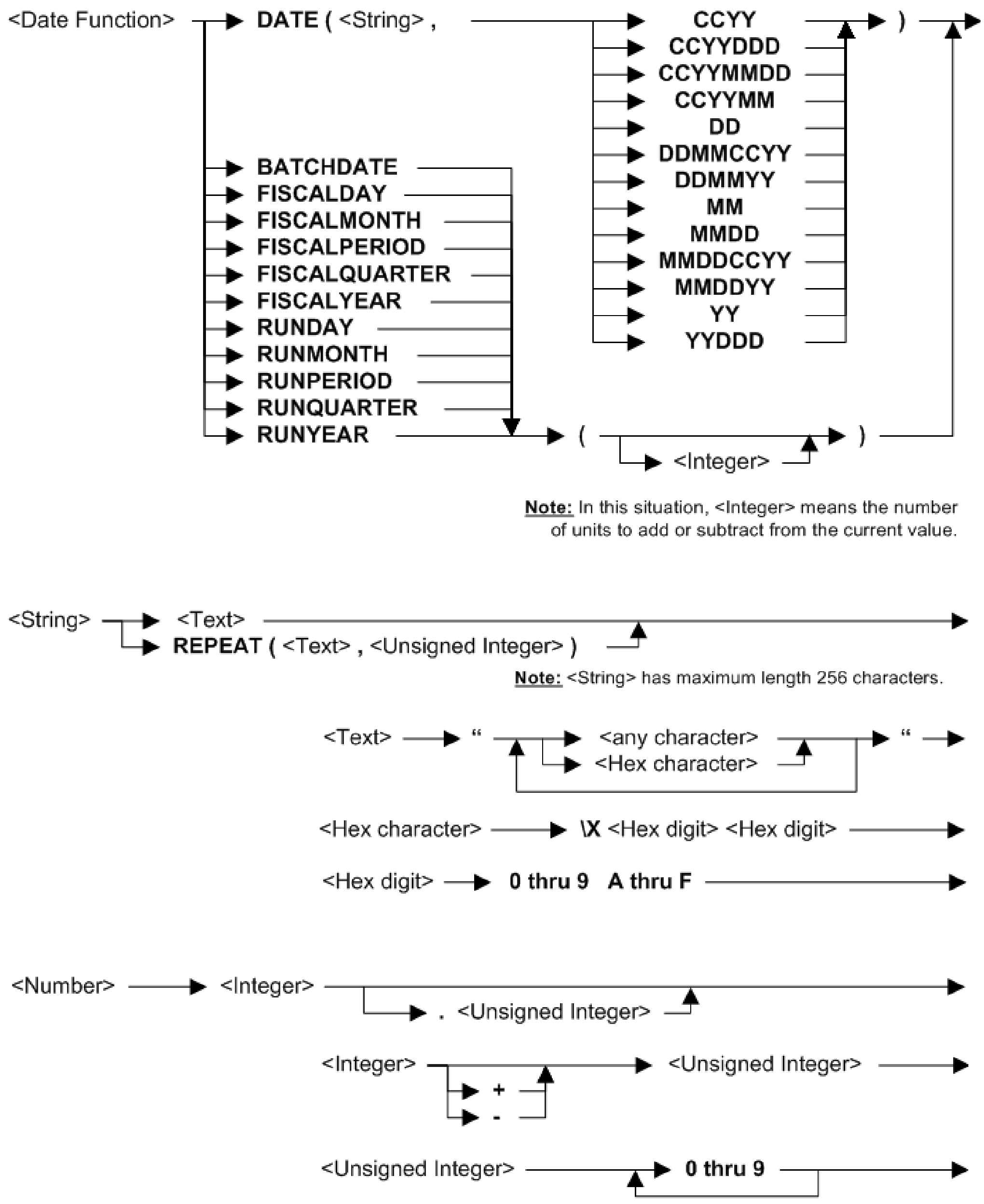
How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax





Note: In this situation, <Integer> means the number of units to add or subtract from the current value.

Note: <String> has maximum length 256 characters.

Rules for the syntax

ISNOTNUMERIC can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

If this functions checks a lookup, and **no effective date** is specified, the lookup uses RUNDAY( ) - see topic "**Syntax: function RUNDAY**". That topic is elsewhere in this PDF - see the table of contents.

See also topic "**Rules for all logic text**". That topic is elsewhere in this PDF - see the table of contents.

Examples: ISNOTNUMERIC function in Extract Record Filter

Example logic text	Meaning

`IF ISNOTNUMERIC({field7})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where field7 is not numeric, and select all other records. This example is the same as:
	`SKIPIF(ISNOTNUMERIC({field7})`

Examples: ISNOTNUMERIC function in Extract Column Assignment

Example logic text	Meaning

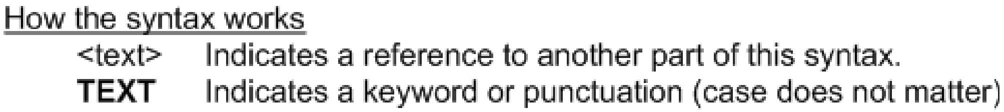
`IF ISNOTNUMERIC({field8})`
`   THEN COLUMN = 0`
`   ELSE COLUMN = {field8}`
`ENDIF`
	If field8 for the current record is not numeric, then set the current column to zero, otherwise set the current column to field8.

Syntax: function ISNOTSPACES

How do I use ISNOTSPACES?

If you provide an input field or lookup path then ISNOTSPACES returns true if the input field or lookup path field is not spaces, and false if the value is spaces.

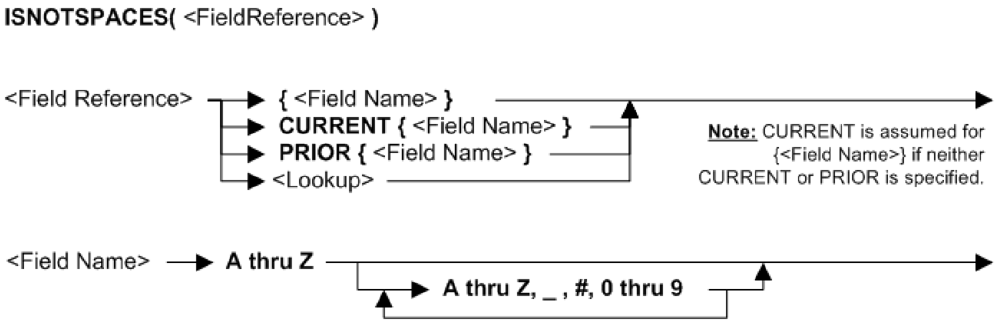
ISNOTSPACES can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

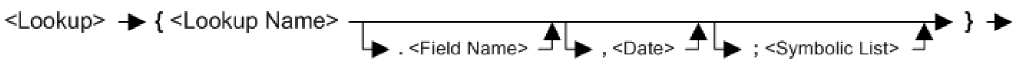


How the syntax works

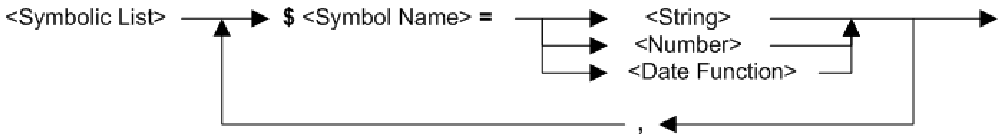
<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

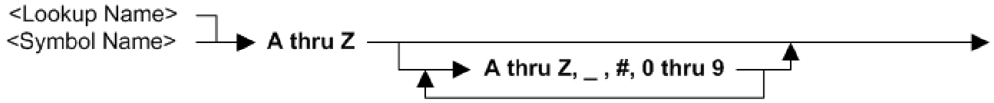
Syntax

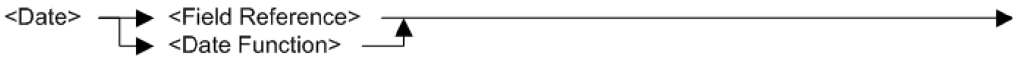

ISNOTSPACES(<FieldReference> **)**
<Field Reference>
{ <Field Name> **}**
CURRENT { <Field Name> **}**
PRIOR { <Field Name> **}**
<Lookup>
Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.
<Field Name>
A thru Z
A thru Z, _ , #, 0 thru 9

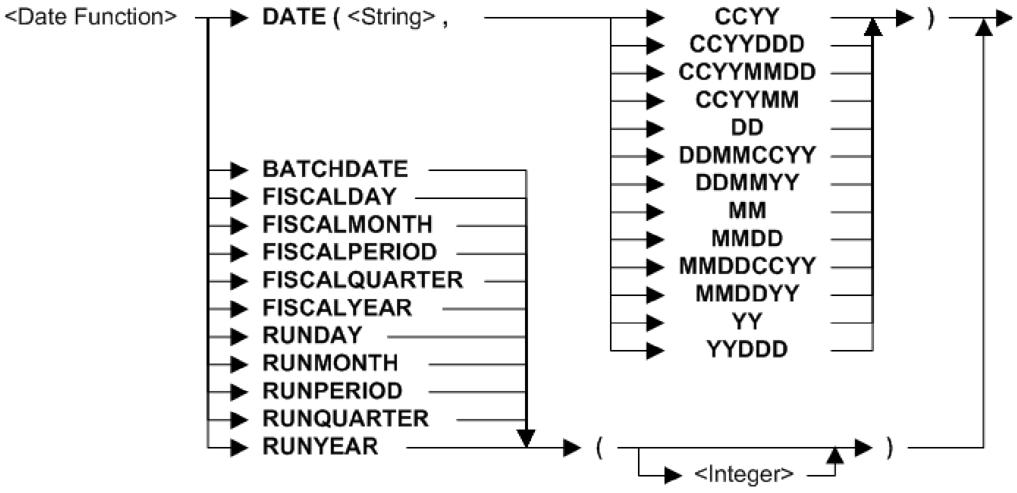


Note: if no date specified, effective date is RUNDAY().

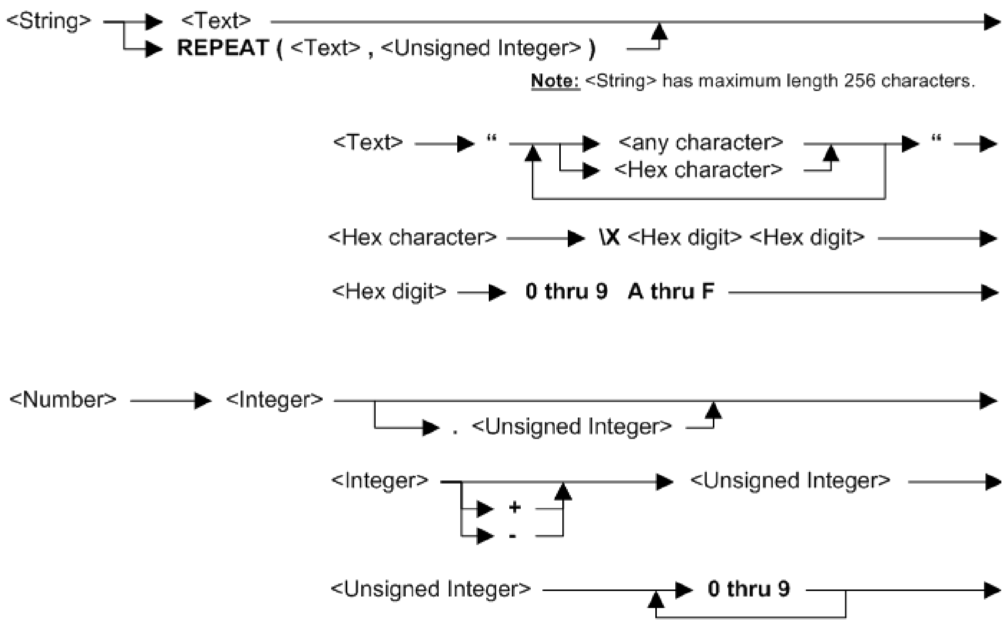








Note: In this situation, <Integer> means the number of units to add or subtract from the current value.



Rules for the syntax

ISNOTSPACES can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

If this functions checks a lookup, and **no effective date** is specified, the lookup uses RUNDAY( ) - see topic "**Syntax: function RUNDAY**". That topic is elsewhere in this PDF - see the table of contents.

See also topic "**Rules for all logic text**". That topic is elsewhere in this PDF - see the table of contents.

Examples: ISNOTSPACES function in Extract Record Filter

Example logic text	Meaning

`IF ISNOTSPACES({field2})`
`   THEN SELECT`
`ENDIF`
	Select all input records where field2 is not spaces, and skip all other records. This example is the same as:
	`SELECTIF(ISNOTSPACES({field2})`

Examples: ISNOTSPACES function in Extract Column Assignment

Example logic text	Meaning

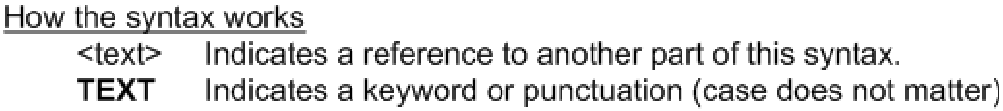
`IF ISNOTSPACES({field3})`
`   THEN COLUMN = {field3}`
`   ELSE COLUMN = "NOT SET"`
`ENDIF`
	If field3 for the current record is not spaces, then set the current column to field3, otherwise set the current column to "NOT SET".

Syntax: function ISNULL

How do I use ISNULL?

If you provide an input field or lookup path then ISNULL returns true if the input field or lookup path field is null values, and false if the value is anything else.

ISNULL can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

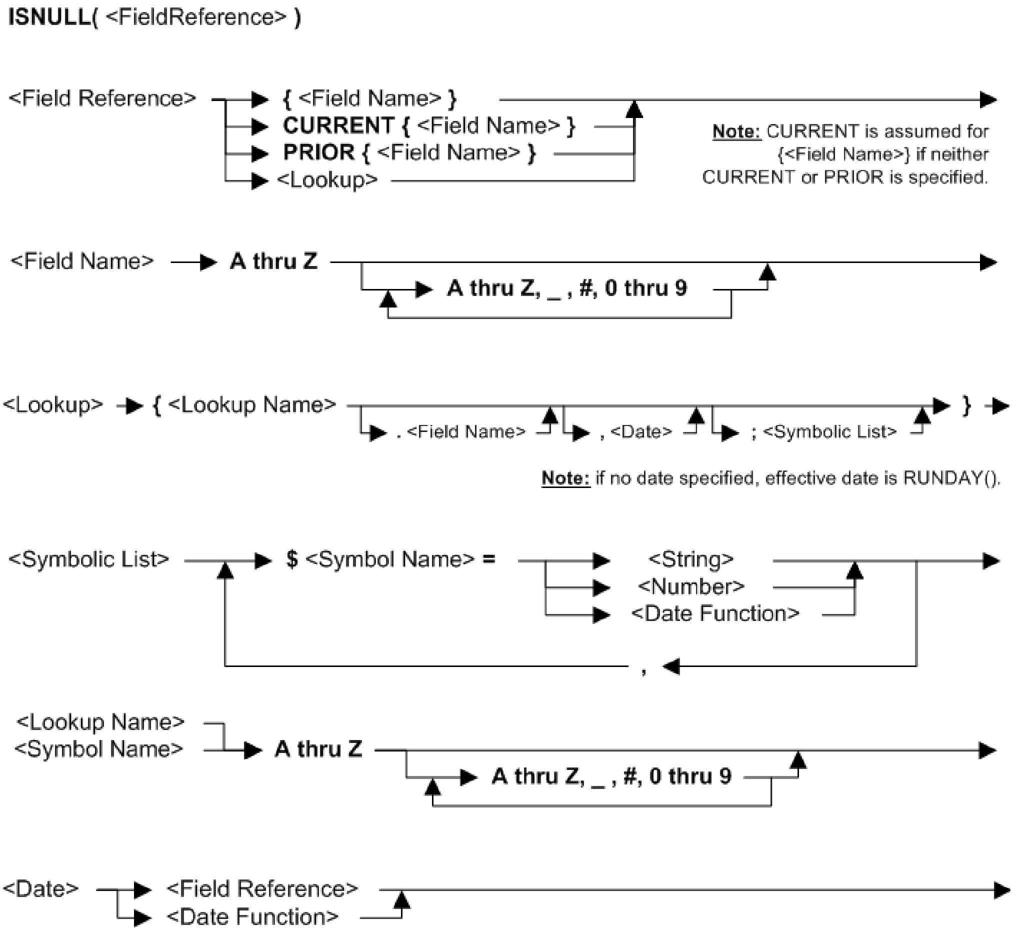


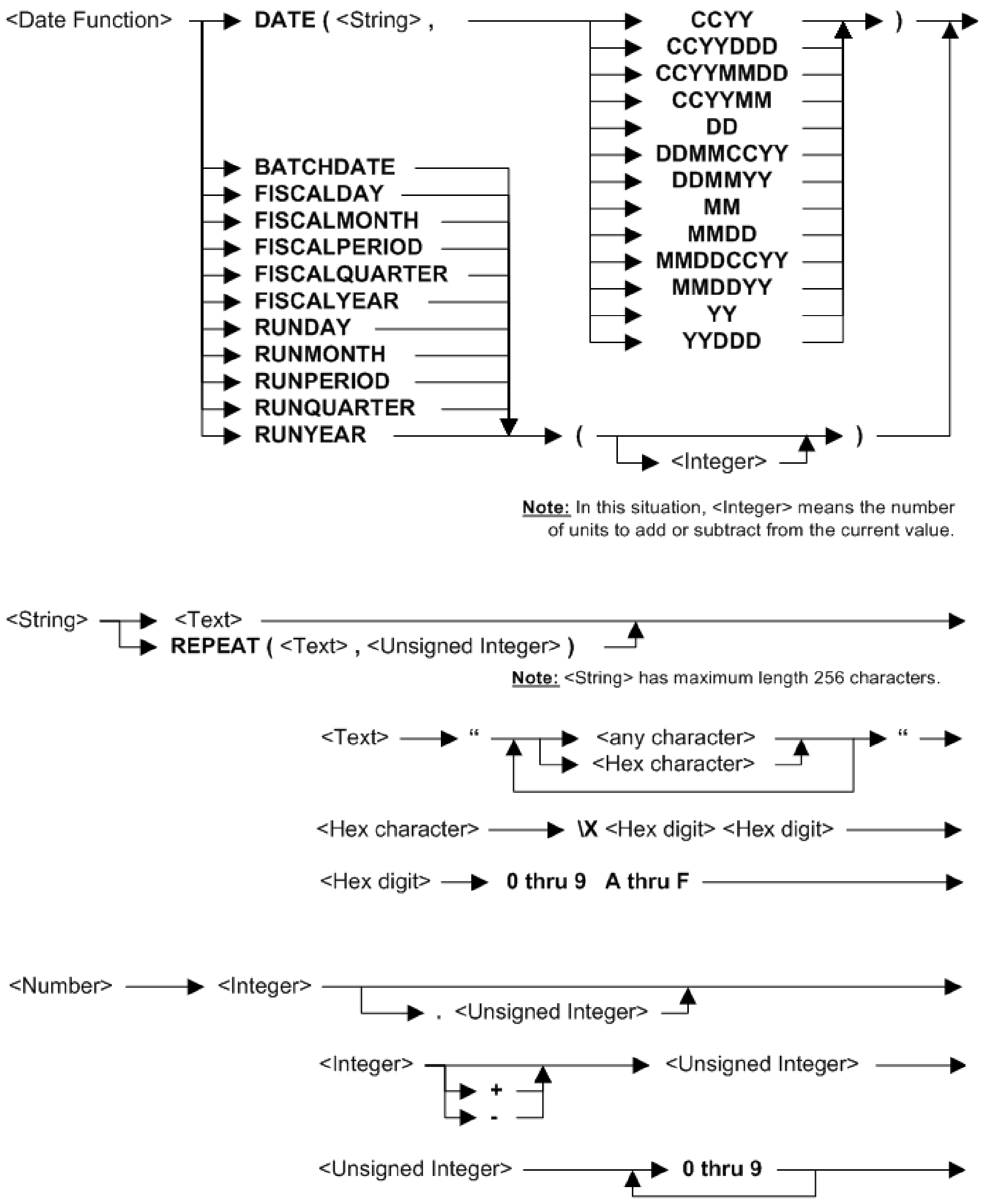
How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax





Note: In this situation, <Integer> means the number of units to add or subtract from the current value.

Note: <String> has maximum length 256 characters.

Rules for the syntax

ISNULL can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

If this functions checks a lookup, and **no effective date** is specified, the lookup uses RUNDAY( ) - see topic "**Syntax: function RUNDAY**". That topic is elsewhere in this PDF - see the table of contents.

See also topic "**Rules for all logic text**". That topic is elsewhere in this PDF - see the table of contents.

Examples: ISNULL function in Extract Record Filter

Example logic text	Meaning

`IF ISNULL({field1})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where field1 contains null values, and select all other records. This example is the same as:
	`SKIPIF(ISNULL({field1})`

Examples: ISNULL function in Extract Column Assignment

Example logic text	Meaning

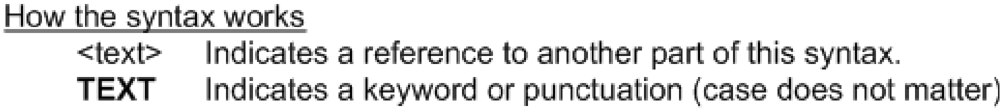
`IF ISNULL({field2})`
`   THEN COLUMN = "EMPTY"`
`   ELSE COLUMN = {field2}`
`ENDIF`
	If field2 for the current record contains null values, then set the current column to "EMPTY", otherwise set the current column to field2.

Syntax: function ISNUMERIC

How do I use ISNUMERIC?

If you provide an input field or lookup path then ISNUMERIC returns true if the input field or lookup path field has a numeric value, and false if the value is anything else.

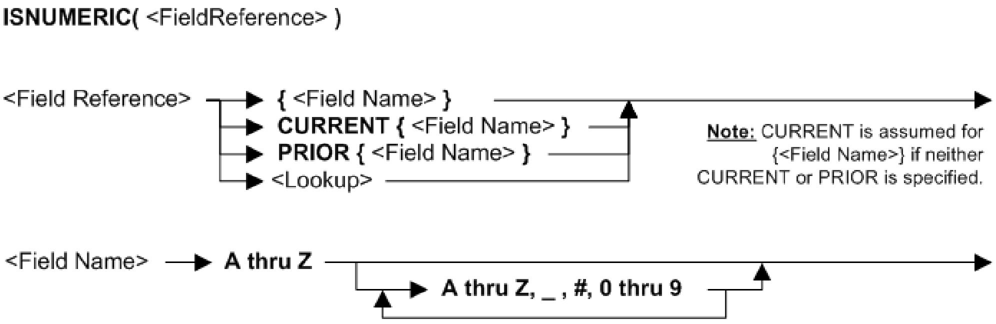
ISNUMERIC can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

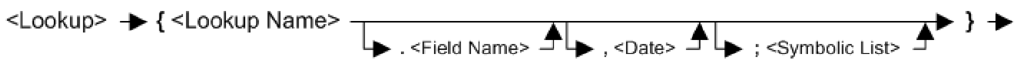


How the syntax works

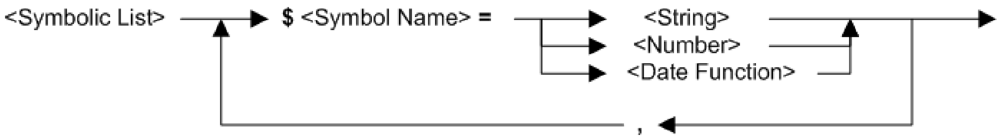
<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

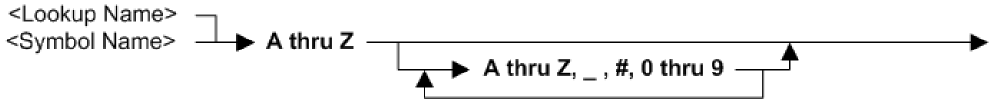
Syntax

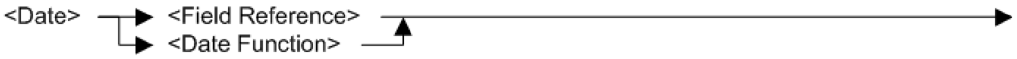

ISNUMERIC(<FieldReference> **)**
<Field Reference>
{ <Field Name> **}**
CURRENT { <Field Name> **}**
PRIOR { <Field Name> **}**
<Lookup>
Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.
<Field Name>
A thru Z
A thru Z, _ , #, 0 thru 9

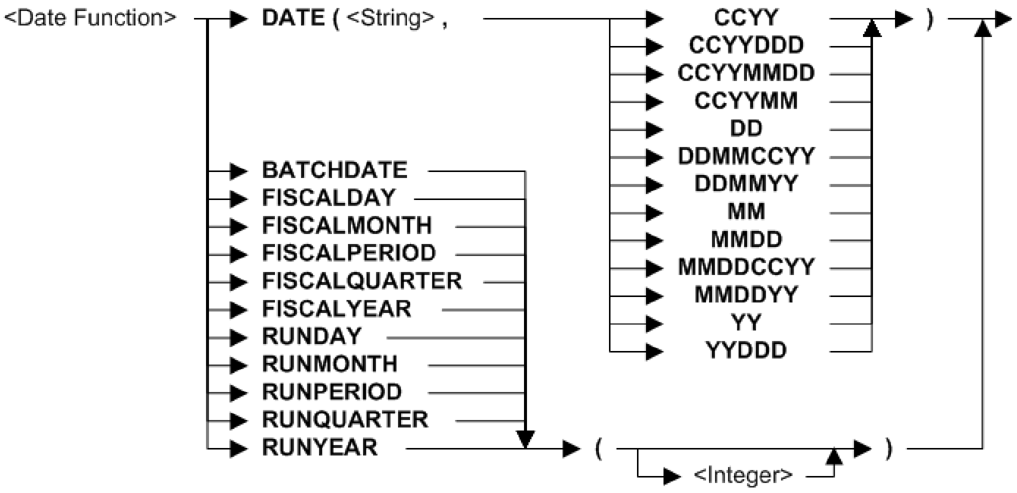


Note: if no date specified, effective date is RUNDAY().

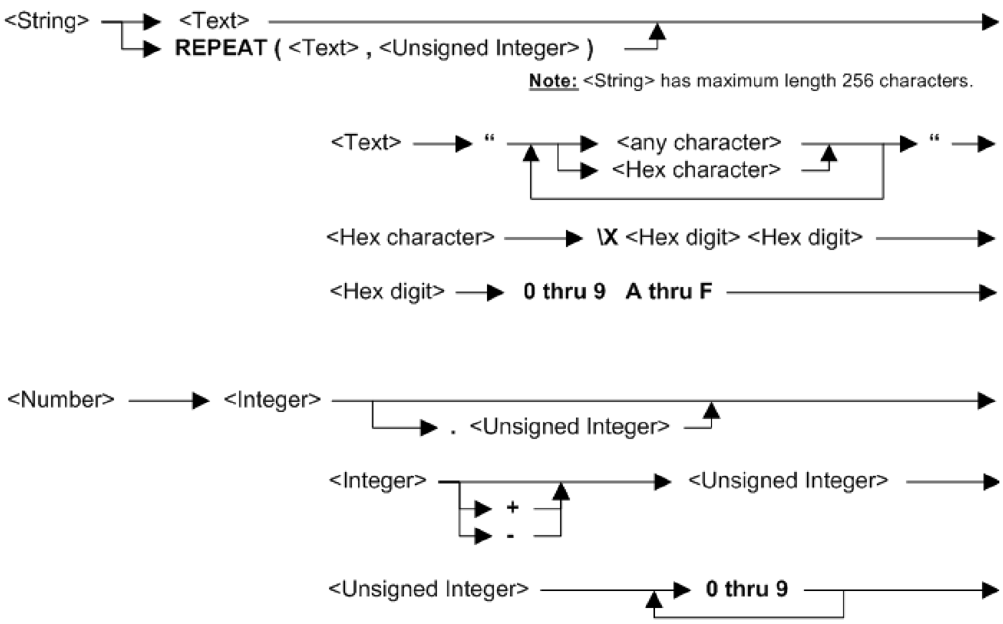








Note: In this situation, <Integer> means the number of units to add or subtract from the current value.



Rules for the syntax

ISNUMERIC can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

If this functions checks a lookup, and **no effective date** is specified, the lookup uses RUNDAY( ) - see topic "**Syntax: function RUNDAY**". That topic is elsewhere in this PDF - see the table of contents.

See also topic "**Rules for all logic text**". That topic is elsewhere in this PDF - see the table of contents.

Examples: ISNUMERIC function in Extract Record Filter

Example logic text	**Meaning**

`IF ISNUMERIC({field5})`
`   THEN SELECT`
`ENDIF`
	Select all input records where field5 is numeric, and skip all other records. This example is the same as:
	`SELECTIF(ISNUMERIC({field5})`

Examples: ISNUMERIC function in Extract Column Assignment

Example logic text	**Meaning**

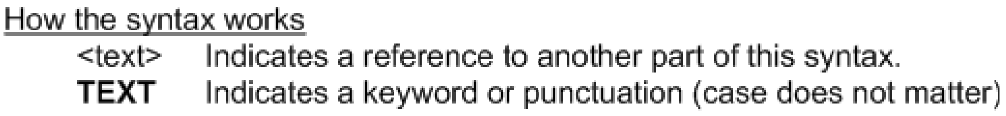
`IF ISNUMERIC({field6})`
`   THEN COLUMN = {field6} * 100`
`   ELSE COLUMN = 0`
`ENDIF`
	If field6 for the current record is numeric, then set the current column to field6 times 100, otherwise set the current column to zero.

Syntax: function ISSPACES

How do I use ISSPACES?

If you provide an input field or lookup path then ISSPACES returns true if the input field or lookup path field contains spaces, and false if the value is anything else.

ISSPACES can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

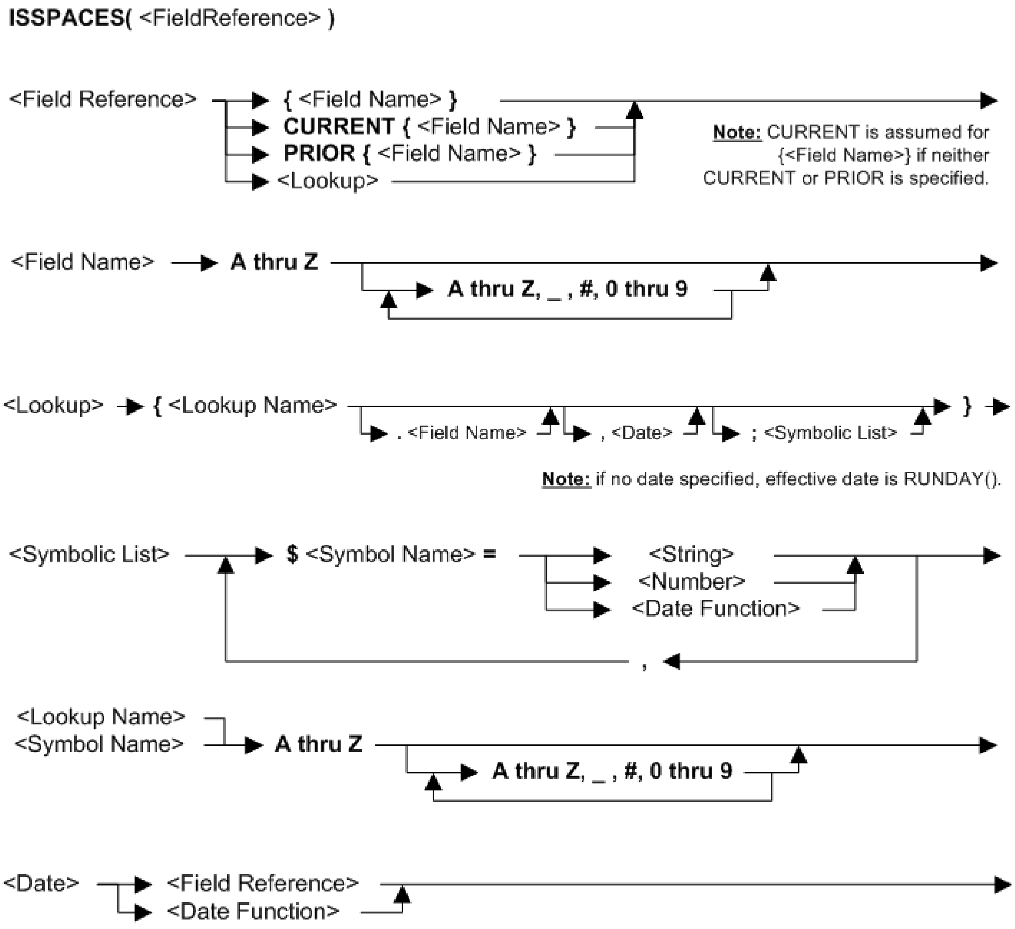


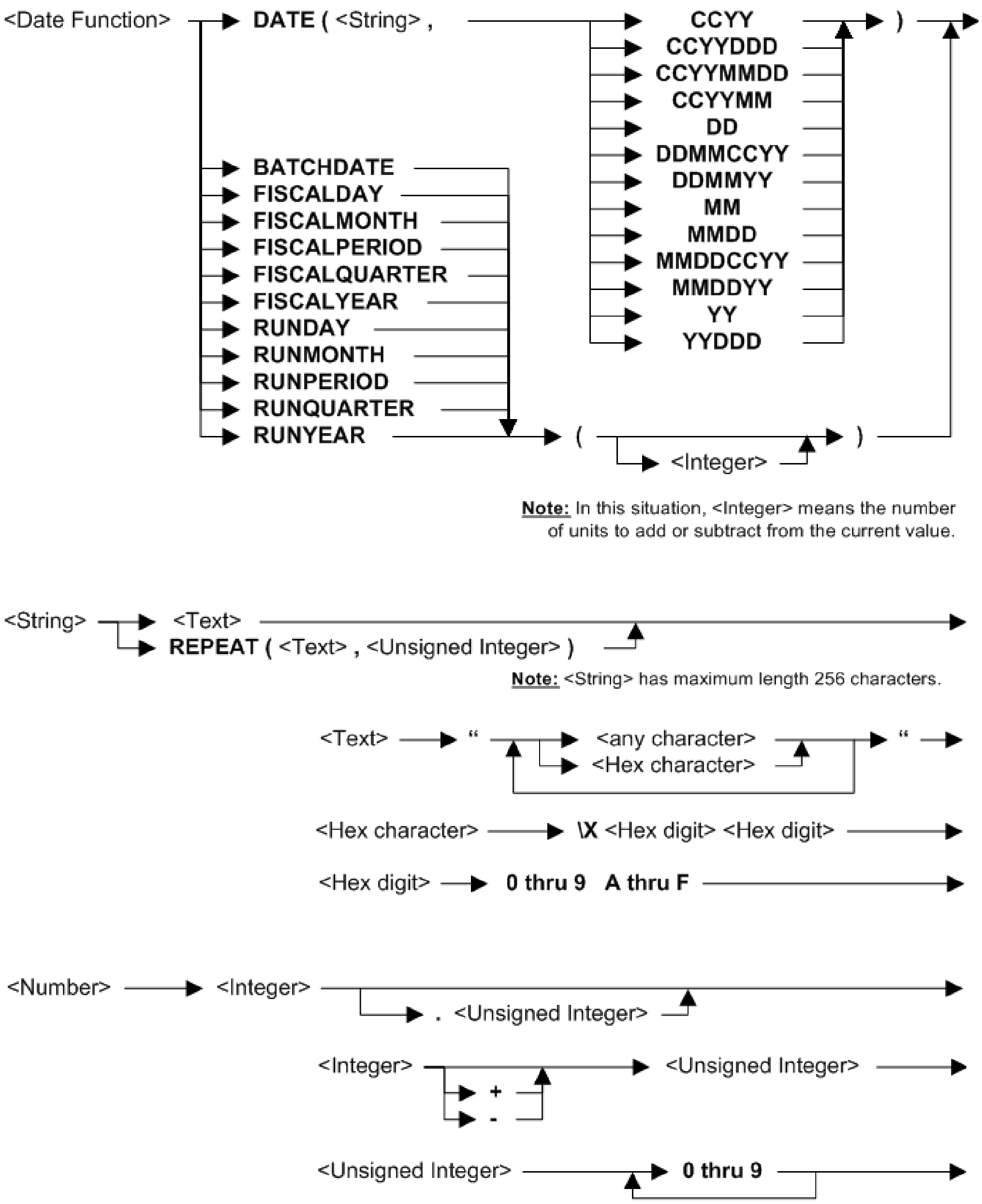
How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax





Note: In this situation, <Integer> means the number of units to add or subtract from the current value.

Note: <String> has maximum length 256 characters.

Rules for the syntax

ISSPACES can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

If this functions checks a lookup, and **no effective date** is specified, the lookup uses RUNDAY( ) - see topic "**Syntax: function RUNDAY**". That topic is elsewhere in this PDF - see the table of contents.

See also topic "**Rules for all logic text**". That topic is elsewhere in this PDF - see the table of contents.

Examples: ISSPACES function in Extract Record Filter

Example logic text	Meaning

`IF ISSPACES({field9})`
`   THEN SKIP`
`ENDIF`
	Skip all input records where field9 is spaces, and select all other records. This example is the same as:
	`SKIPIF(ISSPACES({field9})`

Examples: ISSPACES function in Extract Column Assignment

Example logic text	Meaning

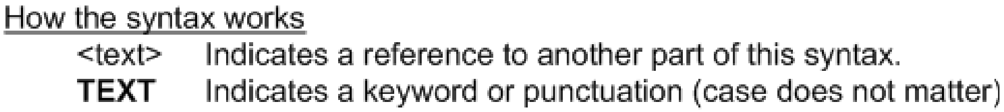
`IF ISSPACES({field1})`
`   THEN COLUMN = "DEFAULT"`
`   ELSE COLUMN = {field1}`
`ENDIF`
	If field1 for the current record is spaces, then set the current column to "DEFAULT", otherwise set the current column to field1.

Syntax: function MONTHSBETWEEN

How do I use MONTHSBETWEEN?

Use MONTHSBETWEEN to compare dates and give an different in months.

MONTHSBETWEEN can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

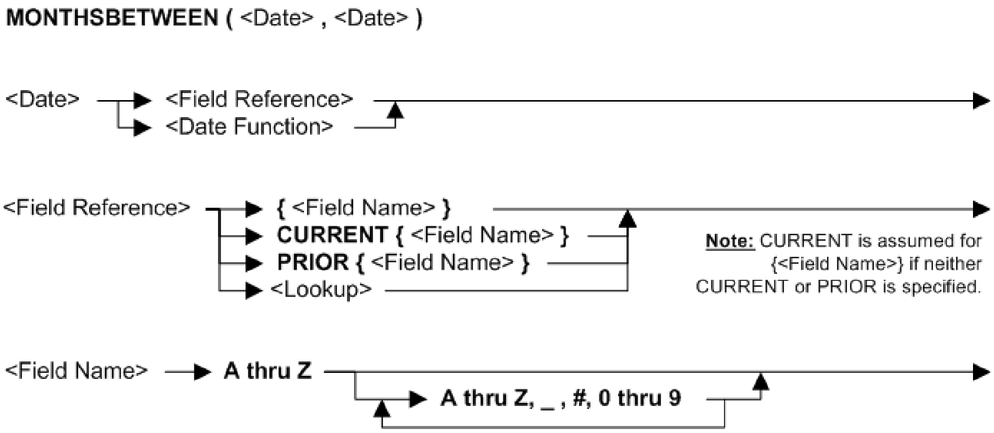


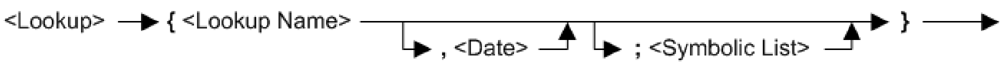
How the syntax works

<text> Indicates a reference to another part of this syntax.

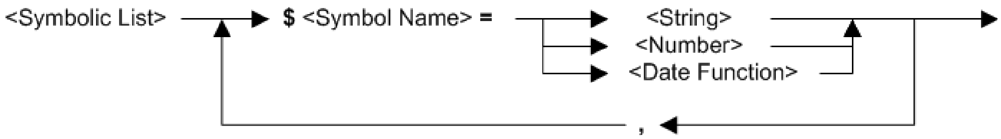
TEXT Indicates a keyword or punctuation (case does not matter)

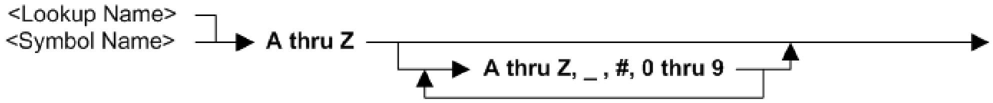
Syntax

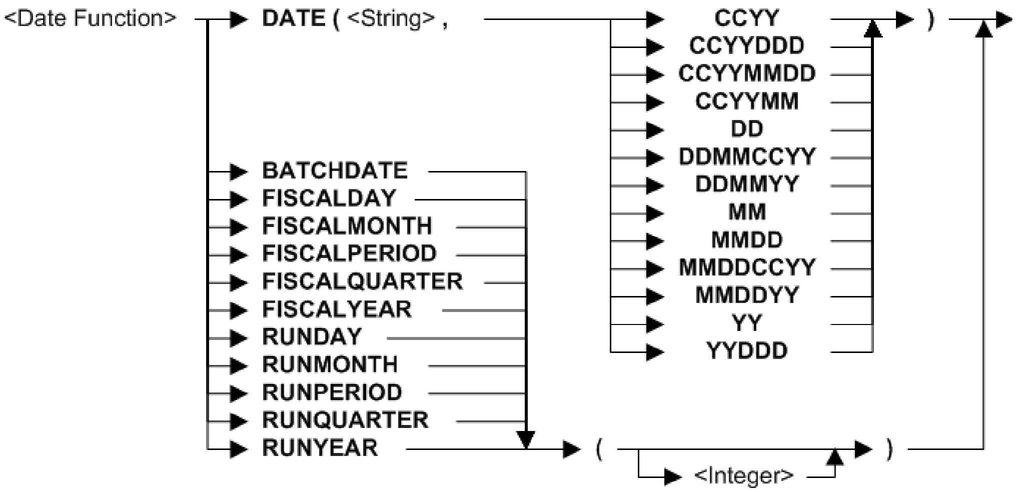




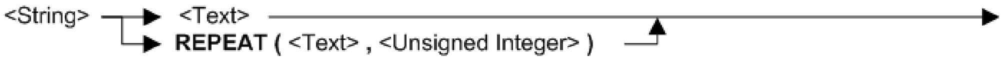
Note: if no date specified, effective date is RUNDAY().



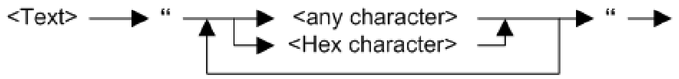


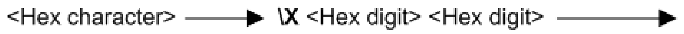


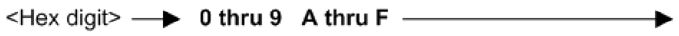
Note: In this situation, <Integer> means the number of units to add or subtract from the current value.

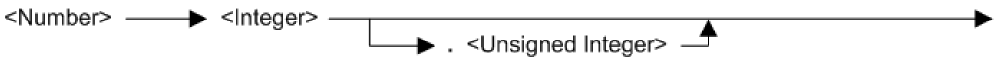


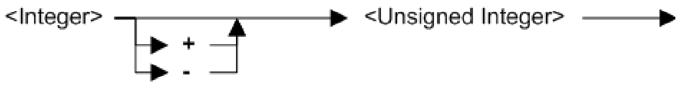
Note: <String> has maximum length 256 characters.

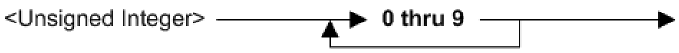












Rules for the syntax

MONTHSBETWEEN can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: function MONTHSBETWEEN in Extract Record Filter

Example logic text	Meaning

IF (MONTHSBETWEEN({field1},{field2})
    >= 3)
   THEN SELECT
ENDIF
	Select only records where there are at least 3 months between field1 and field2, and skip all other records. This example can also be written:
	SELECTIF(MONTHSBETWEEN({field1},{field2})
	           >= 3)

Examples: function MONTHSBETWEEN in Extract Column Assignment

Example logic text	Meaning
COLUMN = MONTHSBETWEEN({BUY_DATE},{SHIP_DATE})	Set the current column to the months between the transaction date and the shipping date.

IF (MONTHSBETWEEN({BUY_DATE},{SHIP_DATE})
     > 1)
   THEN COLUMN = {SHIP_DATE}
   ELSE COLUMN = {BUY_DATE}
ENDIF
	If there is more than one month between the transaction date and the shipping date, then set the current column to the shipping date, otherwise set the current column to the transaction date.

IF (MONTHSBETWEEN({BUY_DATE},{SHIP_DATE})
      > 6)
   THEN WRITE(SOURCE=VIEW,DEST=EXT=03)
ENDIF
	Write to extract 3 those records where there are more than 6 months between the transaction date and the shipping date.

Syntax: function PRIOR

How do I use PRIOR?

PRIOR means the previous input record. For any input record, you can compare the current value of a field with the value in the previous record.

PRIOR is typed before the name of the field, for example:

```
PRIOR {product_code}
```

If you type

```
{product_code}
```

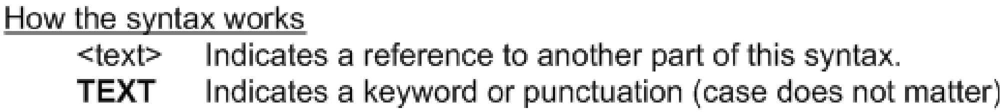
then this means the value in the current record.

If you use PRIOR, it is recommended you put CURRENT in front of all fields that refer to the current input record. As mentioned, this is not necessary - it is recommended because it makes the logic text much easier to understand. For example:

```
IF ((CURRENT {product_code} = PRIOR {product_code}) THEN
```

Notice how CURRENT makes the meaning very clear, even though if you omit the word CURRENT then the logic text works the same way. Normally, CURRENT is used only when a statement contains PRIOR.

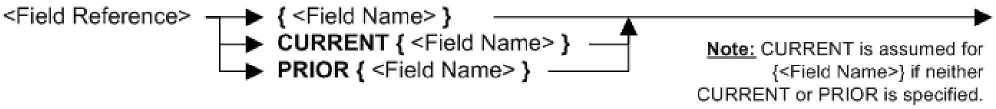
CURRENT and PRIOR can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

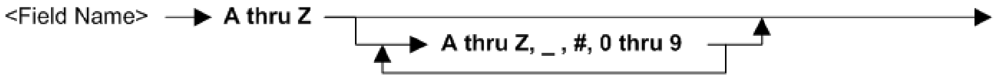


How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax


<Field Reference>
{ <Field Name> }
CURRENT { <Field Name> }
PRIOR { <Field Name> }
Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.



Rules for the syntax

- If neither CURRENT nor PRIOR is typed, then CURRENT is assumed.
- CURRENT and PRIOR can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: CURRENT and PRIOR in Extract Record Filter

Example logic text	Meaning

`IF (CURRENT({field1}) <>`
`    PRIOR({field1}))`
`   THEN SELECT`
`ENDIF`
	Select only records with unique values for field1. This assumes the input file is sorted into field1 order. This example can also be written:
	`SELECTIF(CURRENT({field1})<>`
	`         PRIOR({field1}))`

Examples: CURRENT and PRIOR in Extract Column Assignment

Example logic text	Meaning

IF (CURRENT({field2}) <>
    PRIOR({field2}))
    THEN COLUMN = "PRODUCT: "
    ELSE COLUMN = " "
ENDIF
	If the current record has a different value of field2 from the previous record, set the current column to "PRODUCT: " otherwise set the current column to blank. This assumes the input file is sorted into field2 order.

Syntax: functions Q1, Q2, Q3 and Q4

How do I use Q1 or Q2 or Q3 or Q4?

These functions return a range of dates that are part of a quarter year (a three month period). You can test if a date is inside that quarter.

For example, this logic text tests if field1 is in the first quarter of this year:

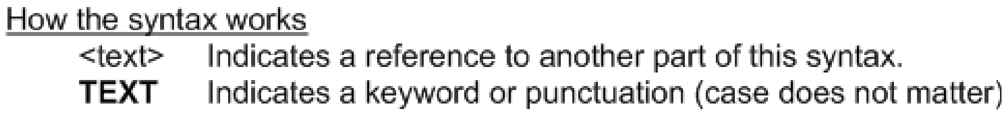
```
IF ({field1} = Q1()) THEN
```

Since there is no parameter for Q1, then the year that the view runs is the year for the quarter.

If you provide a year in CCYY format, then the quarter applies to that year. For example, this logic text tests if field2 is in the third quarter of 2008:

```
IF ({field2} = Q3(2008)) THEN
```

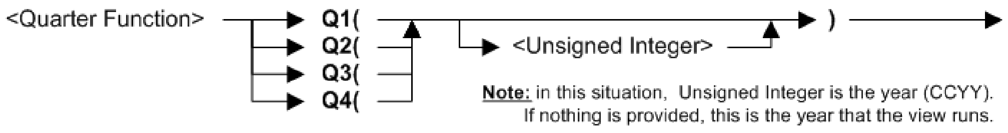
Q1, Q2, Q3 and Q4 can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax


<Quarter Function>
Q1(
Q2(
Q3(
Q4(
<Unsigned Integer>
)
Note: in this situation, Unsigned Integer is the year (CCYY).
If nothing is provided, this is the year that the view runs.

Rules for the syntax

- If there is no CCYY parameter, then the year that the view runs is the year for the quarter.
- Q1, Q2, Q3 and Q4 can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

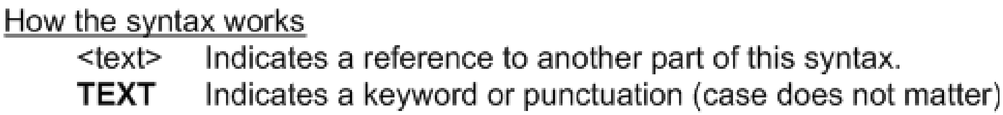
See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Syntax: function REPEAT

How do I use REPEAT?

If you provide a text string and an integer, then REPEAT can create a string that consists of the given text string with integer repetitions. REPEAT is different from ALL because REPEAT has a fixed number of repetitions, whereas ALL is flexible and compares with fields of different lengths.

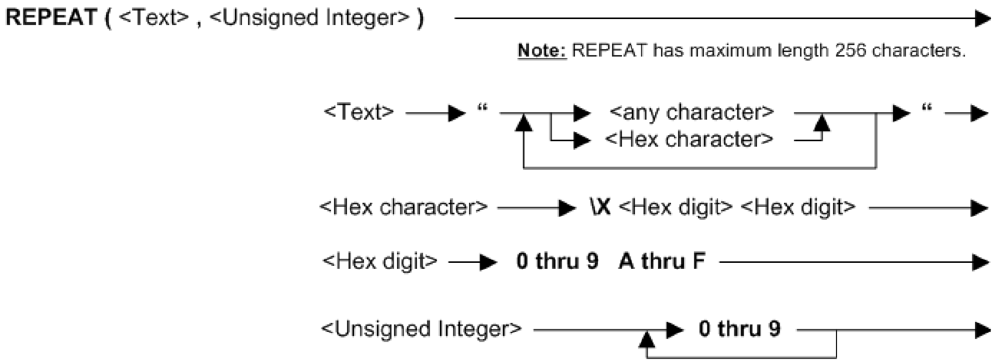
REPEAT can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax


REPEAT (<Text> , <Unsigned Integer>)
Note: REPEAT has maximum length 256 characters.
<Text> “ <any character> <Hex character> “
<Hex character> \X <Hex digit> <Hex digit>
<Hex digit> 0 thru 9 A thru F
<Unsigned Integer> 0 thru 9

Rules for the syntax

- REPEAT can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.
- REPEAT has a maximum length of 256 characters.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: REPEAT function in Extract Record Filter

Example logic text **Meaning**

```
IF NOT ({field7} =
        REPEAT("-", 13))
   THEN SELECT
ENDIF
```

Select for output those records with field7 is not equal to 13 dashes. Skip all other records. This example gives the same result as:

```
SKIPIF({field7} = REPEAT("-", 13))
```

```
IF ({field8} =
     REPEAT("-", 13))
   THEN SKIP
ENDIF
```

Skip for output those records with field8 is equal to 13 dashes. Select all other records. This example gives the same result as:

```
SKIPIF({field8} = REPEAT("-", 13))
```

Examples: REPEAT function in Extract Column Assignment

Example logic text	Meaning

IF ({field9} = "Total")
THEN COLUMN =
REPEAT("-", 13)
ENDIF
	If field9 is "Total" then set the current column to 13 dashes.

IF ({field10} =
REPEAT("-", 13))
THEN COLUMN =
{field1} + {field2}
ENDIF
	If field10 is 13 dashes, then set the current column to a total of fields 1 and 2.

Syntax: function RUNDAY

What is RUNDAY?

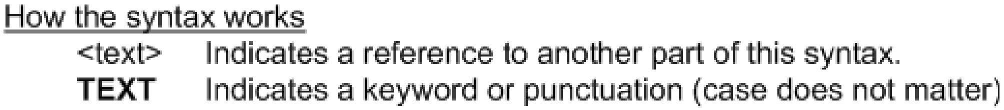
Normally, the date PE runs is the "run date".

RUNDAY returns a CCYYMMDD format date based on the run date. All views in the batch use the same base date for RUNDAY.

How do I use RUNDAY?

The parameter for RUNDAY is a number of days to add or delete from the default RUNDAY. For example, RUNDAY(-5) provides the day five days before the date the view is run.

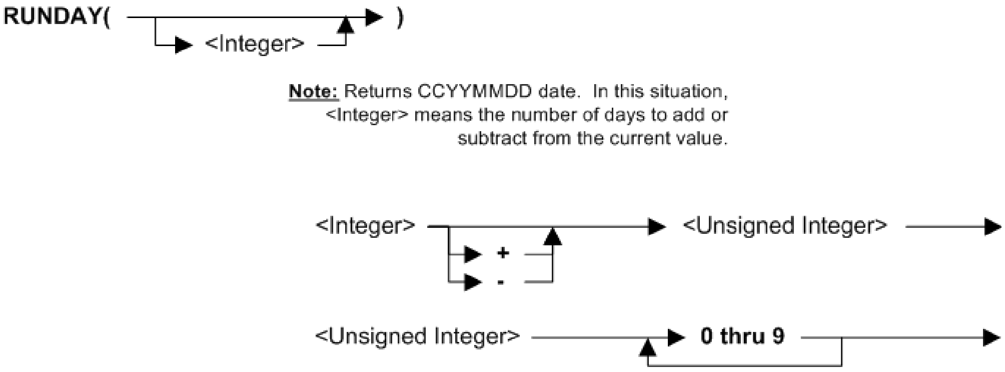
RUNDAY can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMMDD date. In this situation, <Integer> means the number of days to add or subtract from the current value.

Rules for the syntax

RUNDAY can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: RUNDAY function in Extract Record Filter

Example logic text	Meaning

`IF ({field1} < RUNDAY(-7))`
`   THEN SKIP`
`ENDIF`
	Skip any input records where field1 is more than 7 days before the date of running this view, and select all other records. This example assumes that field1 is a date. The code at left can also be written as:
	`SKIPIF({field1} < RUNDAY(-7))`

Examples: RUNDAY function in Extract Column Assignment

Example logic text	Meaning
`COLUMN = RUNDAY()`	Set the current column to the same day number as the view is run.

Syntax: function RUNMONTH

What is RUNMONTH?

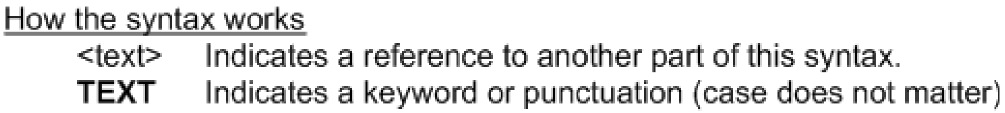
Normally, the date that PE runs is the "run date".

RUNMONTH returns a CCYYMM format date based on the run date. All views in a batch use the same base date for RUNMONTH.

How do I use RUNMONTH?

The parameter for RUNMONTH is a number of months to add or delete from the default RUNMONTH. For example, RUNMONTH(-5) provides the day five months before the date the view is run.

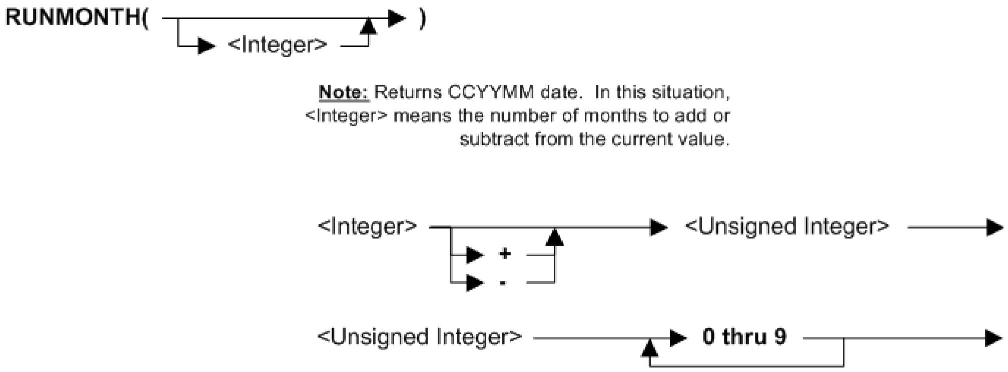
RUNMONTH can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMM date. In this situation, <Integer> means the number of months to add or subtract from the current value.

Rules for the syntax

RUNMONTH can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: RUNMONTH function in Extract Record Filter

Example logic text	Meaning

`IF ({field2} >= RUNMONTH(-1))`
`   THEN SELECT`
`ENDIF`
	Select any input records where field2 is the previous month or later, and skip all other records. The example at left assumes that field2 is a month number. The code at left can also be written as:
	`SELECTIF({field2} >= RUNMONTH(-1))`

Examples: RUNMONTH function in Extract Column Assignment

Example logic text	Meaning
`COLUMN = RUNMONTH()`	Set the current column to the current month number.

Syntax: function RUNPERIOD

What is RUNPERIOD?

RUNPERIOD is a similar concept to month. The difference is that there can be 13 periods in a year instead of 12. Periods are defined in the control record for the environment of the view.

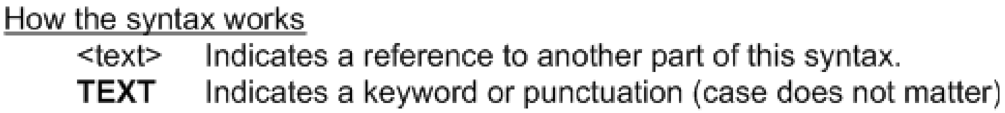
Normally, the date that PE runs is the "run date".

RUNPERIOD returns a CCYYMM format date based on the run date and the period definition in the control record. All views in a batch use the same base date for RUNPERIOD.

How do I use RUNPERIOD?

The parameter for RUNPERIOD is a number of periods to add or delete from the default RUNPERIOD. For example, RUNPERIOD(-5) provides the day five periods before the date the view is run.

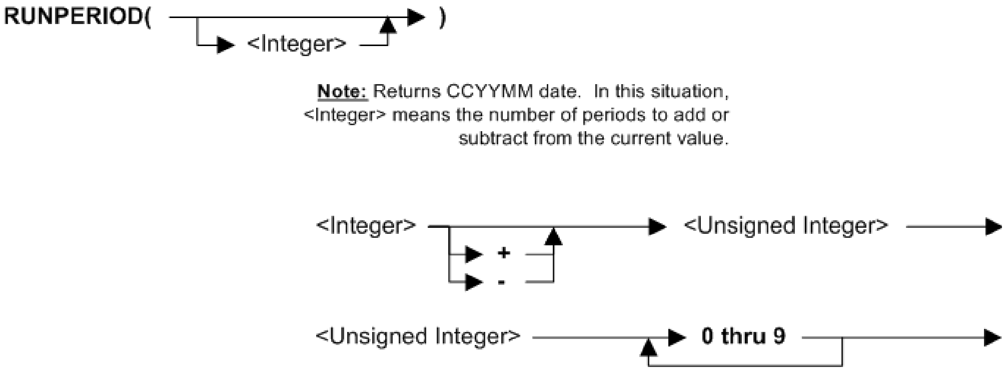
RUNPERIOD can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMM date. In this situation, <Integer> means the number of periods to add or subtract from the current value.

Rules for the syntax

RUNPERIOD can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: RUNPERIOD function in Extract Record Filter

Example logic text	Meaning

`IF ({field3} >= RUNPERIOD(-1))`
`   THEN SELECT`
`ENDIF`
	Select any input records where field3 is the previous period or later, and skip all other records. The example at left assumes that field3 is a period number. The code at left can also be written as:
	`SELECTIF({field3} >= RUNPERIOD(-1))`

Examples: RUNPERIOD function in Extract Column Assignment

Example logic text	Meaning
`COLUMN = RUNPERIOD()`	Set the current column to the current period number.

Syntax: function RUNQUARTER

What is RUNQUARTER?

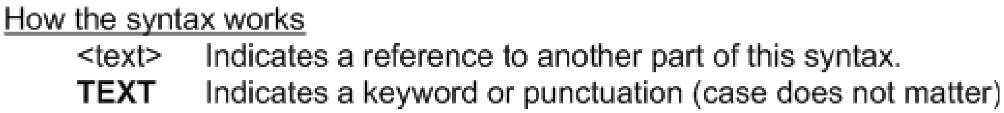
Normally, the date that PE runs is the "run date".

RUNQUARTER returns a CCYYMM format date (at the start of a quarter) based on the run date. All views in a batch use the same base date for RUNQUARTER.

How do I use RUNQUARTER?

The parameter for RUNQUARTER is a number of quarters to add or delete from the default RUNQUARTER. For example, RUNQUARTER(-5) provides the day five quarters before the date the view is run.

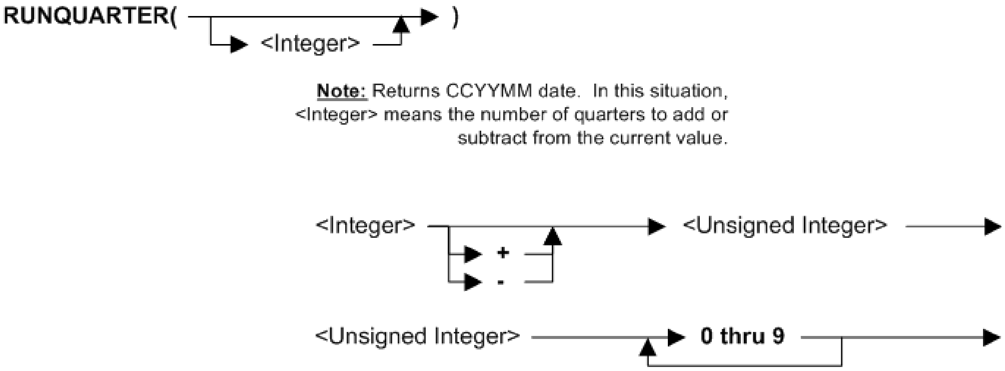
RUNQUARTER can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYYMM date. In this situation, <Integer> means the number of quarters to add or subtract from the current value.

Rules for the syntax

RUNQUARTER can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Syntax: function RUNYEAR

What is RUNYEAR?

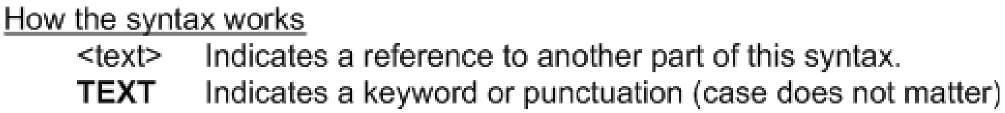
Normally, the date that PE runs is the "run date".

RUNYEAR returns a CCYY format date based on the run date. All views in a batch use the same base date for RUNYEAR.

How do I use RUNYEAR?

The parameter for RUNYEAR is a number of years to add or delete from the default RUNYEAR. For example, RUNYEAR(-5) provides a date five years before the date the view is run.

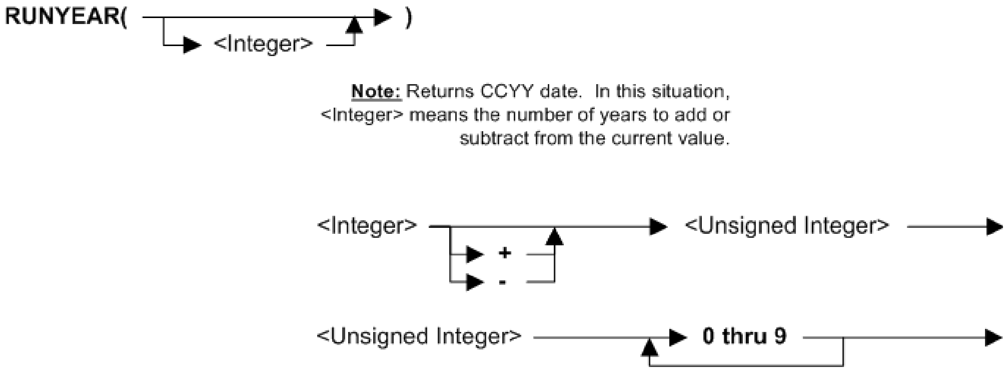
RUNYEAR can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: Returns CCYY date. In this situation, <Integer> means the number of years to add or subtract from the current value.

Rules for the syntax

RUNYEAR can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: RUNYEAR function in Extract Record Filter

Example logic text	Meaning

`IF ({field4} = RUNYEAR(-1))`
`   THEN SELECT`
`ENDIF`
	Select any input records where field4 is the previous year, and skip all other records. The example at left assumes that field4 is a year number. The code at left can also be written as:
	`SELECTIF({field4} = RUNYEAR(-1))`

Examples: RUNYEAR function in Extract Column Assignment

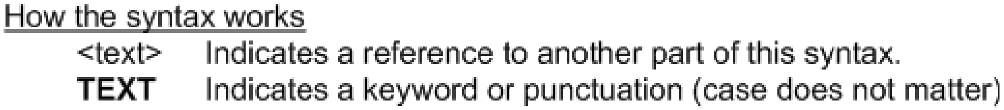
Example logic text	Meaning
`COLUMN = RUNYEAR()`	Set the current column to the current year number.

Syntax: function YEARSBETWEEN

How do I use YEARSBETWEEN?

Use YEARSBETWEEN to compare dates and give an different in years.

YEARSBETWEEN can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

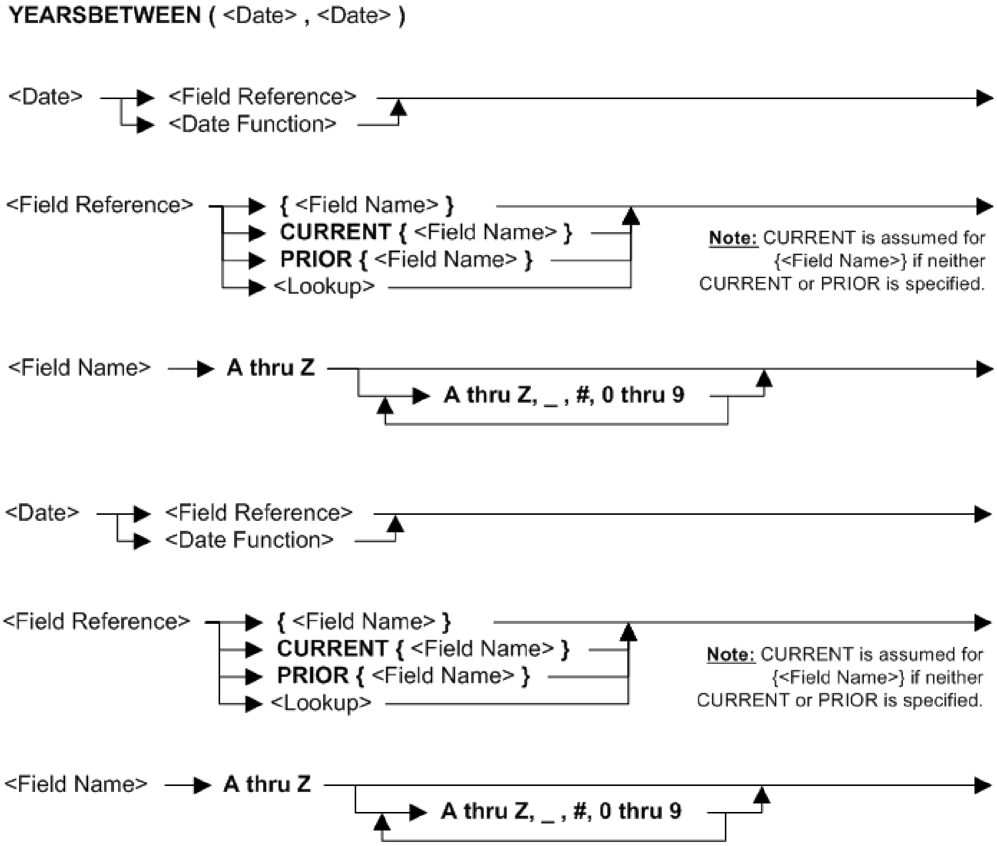


How the syntax works

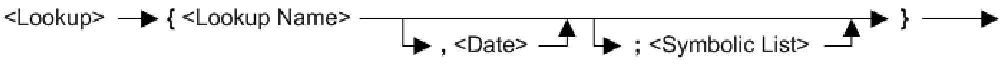
<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

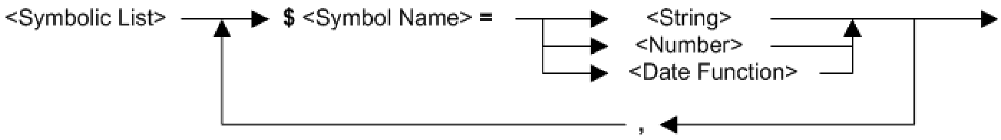
Syntax

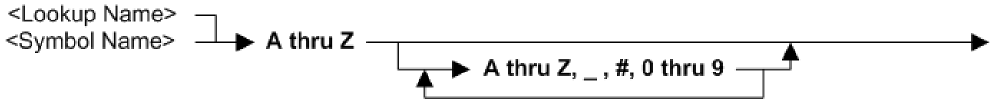


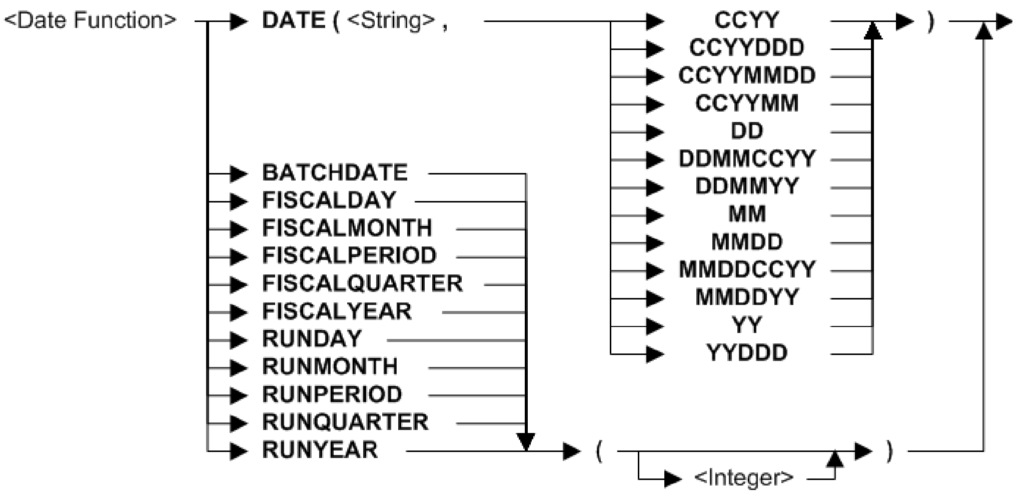
YEARSBETWEEN (<Date> **,** <Date> **)**
<Date>
<Field Reference>
<Date Function>
<Field Reference>
{ <Field Name> **}**
CURRENT { <Field Name> **}**
PRIOR { <Field Name> **}**
<Lookup>
Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.
<Field Name>
A thru Z
A thru Z, _ , #, 0 thru 9
<Date>
<Field Reference>
<Date Function>
<Field Reference>
{ <Field Name> **}**
CURRENT { <Field Name> **}**
PRIOR { <Field Name> **}**
<Lookup>
Note: CURRENT is assumed for {<Field Name>} if neither CURRENT or PRIOR is specified.
<Field Name>
A thru Z
A thru Z, _ , #, 0 thru 9



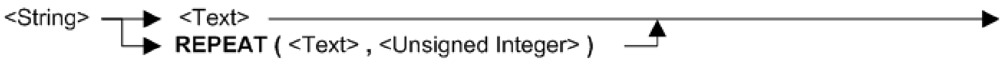
Note: if no date specified, effective date is RUNDAY().



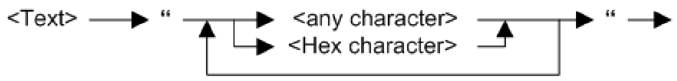


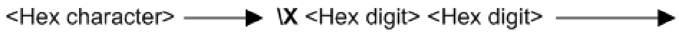


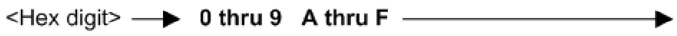
Note: In this situation, <Integer> means the number of units to add or subtract from the current value.

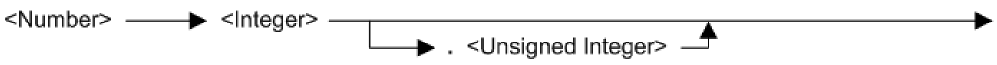


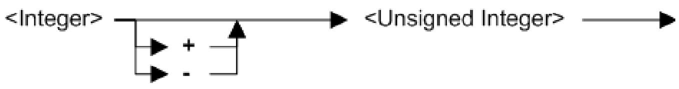
Note: <String> has maximum length 256 characters.

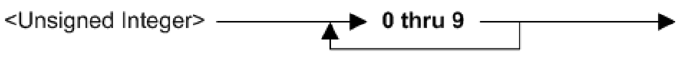












Rules for the syntax

YEARSBETWEEN can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: function YEARSBETWEEN in Extract Record Filter

Example logic text	Meaning

`IF (YEARSBETWEEN({field1},{field2})`
`    >= 1)`
`   THEN SELECT`
`ENDIF`
	Select only records where there is at least one year between field1 and field2, and skip all other records This example can also be written:
	`SELECTIF(YEARSBETWEEN({field1},{field2})`
	`         >= 1)`

Examples: function YEARSBETWEEN in Extract Column Assignment

Example logic text	Meaning
`COLUMN = YEARSBETWEEN({BUY_DATE},{SHIP_DATE})`	Set the current column to the years between the transaction date and the shipping date.

`IF (YEARSBETWEEN({BUY_DATE},{SHIP_DATE})`
`     >= 1)`
`   THEN COLUMN = {SHIP_DATE}`
`   ELSE COLUMN = {BUY_DATE}`
`ENDIF`
	If there is at least one year between the transaction date and the shipping date, then set the current column to the shipping date, otherwise set the current column to the transaction date.

`IF (YEARSBETWEEN({BUY_DATE},{SHIP_DATE})`
`     >= 1)`
`   THEN WRITE(SOURCE=VIEW,DEST=EXT=03)`
`ENDIF`
	Write to extract 3 those records where there is at least one year between the transaction date and the shipping date.

Chapter 4. Syntax: string comparison

Syntax: BEGINS_WITH

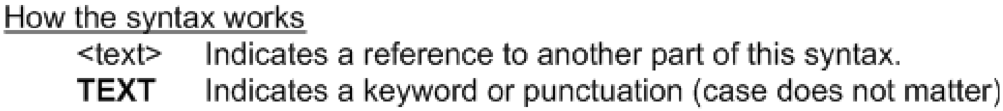
How do I use BEGINS_WITH?

BEGINS_WITH are keywords that are used as string comparison operators. You can check a string begins with certain characters.

For example, a field with "LONDON" begins with the string "L" and "LO" and even "LONDON".

BEGINS_WITH is an example of string comparisons that return a true or false value that can be part of an IF statement.

BEGINS_WITH can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

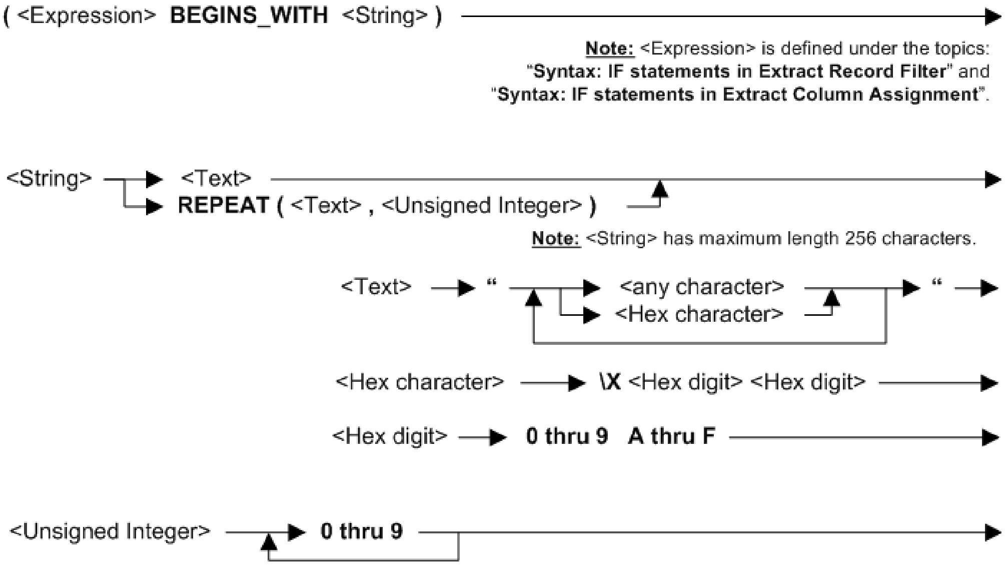


How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax



Note: <Expression> is defined under the topics: "**Syntax: IF statements in Extract Record Filter**" and "**Syntax: IF statements in Extract Column Assignment**".

Note: <String> has maximum length 256 characters.

Rules for the syntax

BEGINS_WITH can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: BEGINS_WITH in Extract Record Filter

Example logic text	Meaning
SELECTIF({field1} BEGINS_WITH "BBB")	Select input records where field1 begins with characters "BBB", and skip all other records.

IF ({field1}
BEGINS_WITH "BBB")
THEN SELECT
ENDIF
	Select input records where field1 begins with characters "BBB", and skip all other records. This example can be written:
	SELECTIF({field1} BEGINS_WITH "BBB")

Examples: BEGINS_WITH in Extract Column Assignment

Example logic text	Meaning

IF ({field1}
BEGINS_WITH "BBB")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	If field1 begins with characters "BBB" then set the current column to field1, otherwise set the current column to blank.

Syntax: CONTAINS

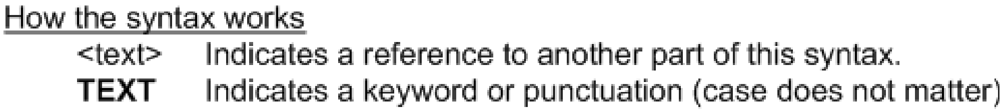
How do I use CONTAINS?

CONTAINS is a keyword that is used as a string comparison operator. You can check a string contains with certain characters.

For example, a field with "LONDON" contains the string "ON" and "DO" and even "LONDON".

CONTAINS is an example of string comparisons that return a true or false value that can be part of an IF statement.

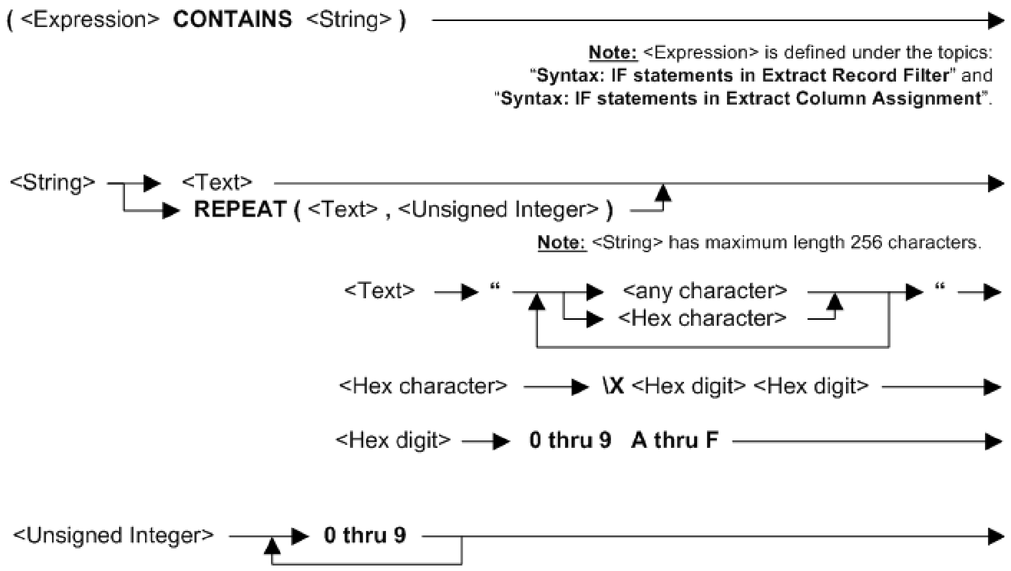
CONTAINS can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax


(<Expression> CONTAINS <String>)
Note: <Expression> is defined under the topics: "Syntax: IF statements in Extract Record Filter" and "Syntax: IF statements in Extract Column Assignment".
<String>
<Text>
REPEAT (<Text> , <Unsigned Integer>)
Note: <String> has maximum length 256 characters.
<Text>
"
<any character>
<Hex character>
"
<Hex character>
\X <Hex digit> <Hex digit>
<Hex digit>
0 thru 9 A thru F
<Unsigned Integer>
0 thru 9

Rules for the syntax

CONTAINS can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: CONTAINS in Extract Record Filter

Example logic text	Meaning
SELECTIF({field2} CONTAINS "CCC")	Select input records where field2 contains characters "CCC", and skip all other records.

IF ({field2}
CONTAINS "CCC")
THEN SELECT
ENDIF
	Select input records where field2 contains characters "CCC", and skip all other records. This example can be written:
	SELECTIF({field2} CONTAINS "CCC")

Examples: CONTAINS in Extract Column Assignment

Example logic text	Meaning

IF ({field2} |
CONTAINS "CCC")
THEN COLUMN = {field2}
ELSE COLUMN = " "
ENDIF
	If field2 contains characters "CCC" then set the current column to field2, otherwise set the current column to blank.

Syntax: ENDS_WITH

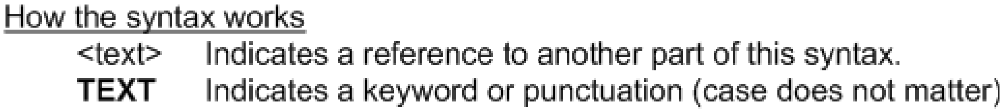
How do I use ENDS_WITH?

ENDS_WITH are keywords that are used as string comparison operators. You can check a string ends with certain characters.

For example, a field with "LONDON" begins with the string "N" and "ON" and even "LONDON".

ENDS_WITH is an example of string comparisons that return a true or false value that can be part of an IF statement.

ENDS_WITH can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

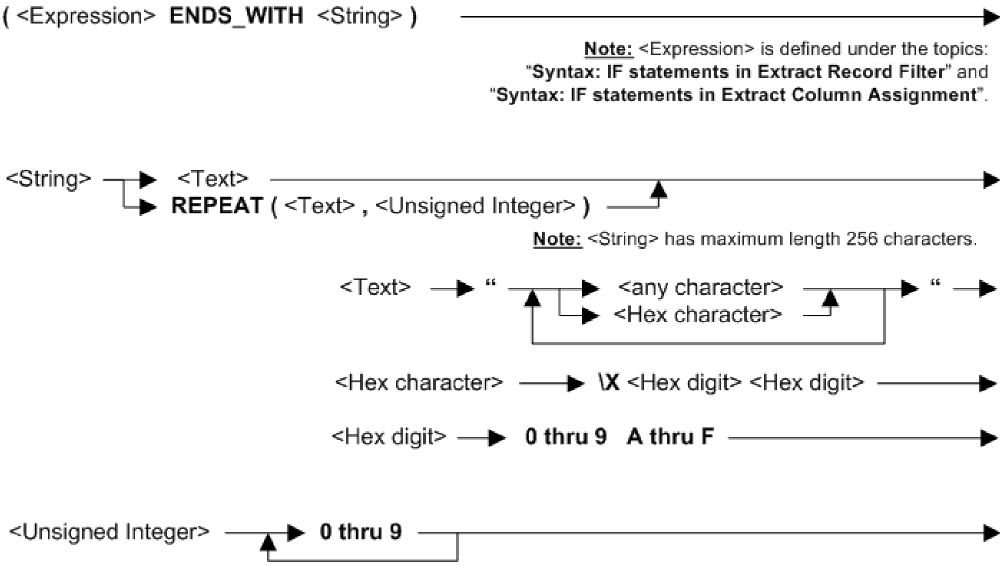


How the syntax works

<text> Indicates a reference to another part of this syntax.

TEXT Indicates a keyword or punctuation (case does not matter)

Syntax



Note: <Expression> is defined under the topics: "**Syntax: IF statements in Extract Record Filter**" and "**Syntax: IF statements in Extract Column Assignment**".

Note: <String> has maximum length 256 characters.

Rules for the syntax

ENDS_WITH can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: ENDS_WITH in Extract Record Filter

Example logic text	Meaning

`SELECTIF({field3}`
`       ENDS_WITH "EEE")`
	Select input records where field3 ends with characters "EEE", and skip all other records.

`IF ({field3}`
`   ENDS_WITH "EEE")`
`   THEN SELECT`
`ENDIF`
	Select input records where field3 ends with characters "EEE", and skip all other records. This example can be written:
	`SELECTIF({field3} ENDS_WITH "EEE")`

Examples: ENDS_WITH in Extract Column Assignment

Example logic text	Meaning

`IF ({field3}`
`   ENDS_WITH "EEE")`
`   THEN COLUMN = {field3}`
`   ELSE COLUMN = " "`
`ENDIF`
	If field3 ends with characters "EEE" then set the current column to field3, otherwise set the current column to blank.

Syntax: LIKE

How do I use LIKE?

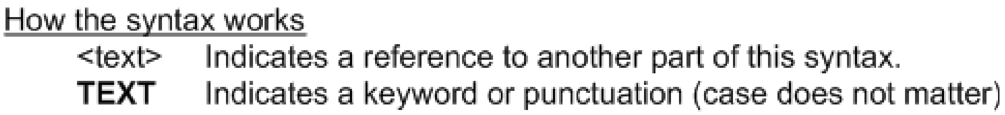
LIKE is a keyword that is used as a string comparison operator. You can check a string contains certain characters in certain positions.

For example, a field with "It is raining in London " starts with "I" and contains "rain" and ends with "on". All these things can be checked in one use of LIKE.

LIKE is an example of string comparisons that return a true or false value that can be part of an IF statement.

LIKE can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

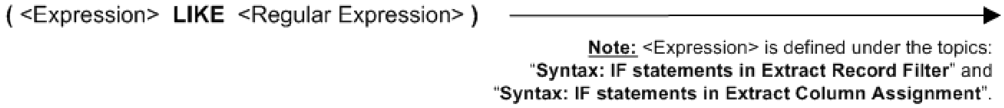
LIKE has exactly the same effect as MATCHES (another string comparison operator).



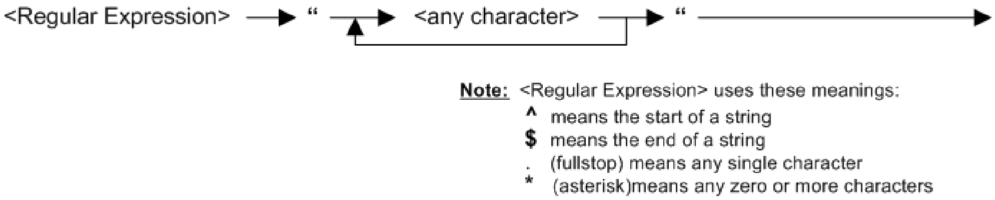
How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: <Expression> is defined under the topics: "**Syntax: IF statements in Extract Record Filter**" and "**Syntax: IF statements in Extract Column Assignment**".



Note: <Regular Expression> uses these meanings:
^ means the start of a string
$ means the end of a string
. (fullstop) means any single character
* (asterisk)means any zero or more characters

Rules for the syntax

- LIKE can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.
- LIKE has exactly the same effect as **MATCHES** (another string comparison operator).

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: LIKE in Extract Record Filter

Character "^" means the start of a string.

Character "$" means the end of a string.

Character "." means any single character.

Character "*" means zero or more characters at this point.

Example logic text	**Meaning**
`SELECTIF({field1} LIKE "MA...")`	Select input records where field1 is exactly 5 characters starting with "MA", and skip all other records.
`SELECTIF({field1} LIKE "..VA..")`	Select input records where field1 is exactly 6 characters with characters 3 and 4 as "VA", and skip all other records.
`SELECTIF({field1} LIKE ".....NA")`	Select input records where field1 is exactly 6 characters ending in "NA", and skip all other records.
`SELECTIF({field1} LIKE "^BBB*")`	Select input records where field1 begins with characters "BBB", and skip all other records. This example has the same effect as: `SELECTIF({field1} BEGINS_WITH "BBB")` It is better to use BEGINS_WITH because the logic text executes faster.

Example logic text	Meaning

`IF ({field1} LIKE "^BBB*")`
`THEN SELECT`
`ENDIF`
	Select input records where field1 begins with characters "BBB", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} BEGINS_WITH "BBB")`
	It is better to use BEGINS_WITH because the logic text executes faster.

`SELECTIF({field1} LIKE "*CCC*")`	Select input records where field1 contains characters "CCC", and skip all other records. This example has the same effect as: `SELECTIF({field1} CONTAINS "CCC")` It is better to use CONTAINS because the logic text executes faster.

`IF ({field1} LIKE "*CCC*")`
`THEN SELECT`
`ENDIF`
	Select input records where field1 contains characters "CCC", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} CONTAINS "CCC")`
	It is better to use CONTAINS because the logic text executes faster.

`SELECTIF({field1} LIKE "*EEE$")`	Select input records where field1 ends with characters "EEE", and skip all other records. This example has the same effect as: `SELECTIF({field1} ENDS_WITH "EEE")` It is better to use ENDS_WITH because the logic text executes faster.

`IF ({field1} LIKE "*EEE$")`
`THEN SELECT`
`ENDIF`
	Select input records where field1 ends with characters "EEE", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} ENDS_WITH "EEE")`
	It is better to use ENDS_WITH because the logic text executes faster.

`SELECTIF({field1}`
`LIKE "^B*C*E$")`
	Select input records where field1 begins with "B", contains "C" and ends with "E", and skip all other records.

`IF ({field1}`
`LIKE "^B*C*E$")`
`THEN SELECT`
`ENDIF`
	Select input records where field1 begins with "B", contains "C" and ends with "E", and skip all other records.

Examples: LIKE in Extract Column Assignment

Character "^" means the start of a string.

Character "$" means the end of a string.

Character "." means any single character.

Character "*" means zero or more characters at this point.

Example logic text	Meaning

IF ({field1} LIKE "MA...")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	Select input records where field1 is exactly 5 characters starting with "MA", and skip all other records.

IF ({field1} LIKE "..VA..")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	Select input records where field1 is exactly 6 characters with characters 3 and 4 as "VA", and skip all other records.

IF ({field1} LIKE ".....NA")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	Select input records where field1 is exactly 6 characters ending in "NA", and skip all other records.

IF ({field1} LIKE "^BBB*")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	If field1 begins with characters "BBB" then set the current column to field1, otherwise set the current column to blank.

IF ({field1} LIKE "*CCC*")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	If field1 contains characters "CCC" then set the current column to field1, otherwise set the current column to blank.

IF ({field1} LIKE "*EEE$")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	If field1 ends with characters "EEE" then set the current column to field1, otherwise set the current column to blank.

IF ({field1}
LIKE "^B*C*E$")
THEN COLUMN = {field1}
ELSE COLUMN = " "
ENDIF
	If field1 begins with "B", contains "C" and ends with "E" then set the current column to field1, otherwise set the current column to blank.

Syntax: MATCHES

How do I use MATCHES?

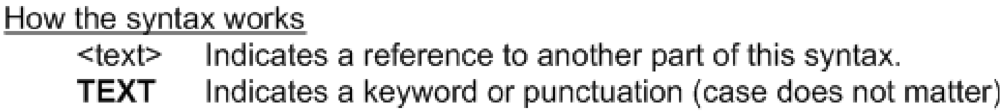
MATCHES is a keyword that is used as a string comparison operator. You can check a string contains certain characters in certain positions.

For example, a field with "It is raining in London " starts with "I" and contains "rain" and ends with "on". All these things can be checked in one use of MATCHES.

MATCHES is an example of string comparisons that return a true or false value that can be part of an IF statement.

MATCHES can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.

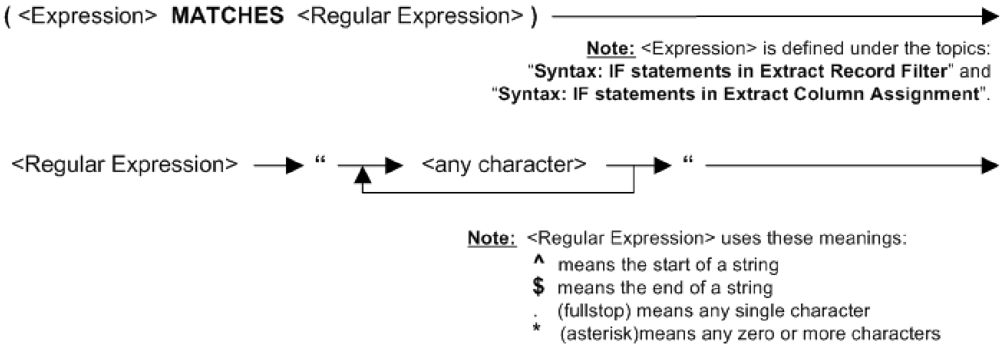
MATCHES has exactly the same effect as LIKE (another string comparison operator).



How the syntax works

<text>	Indicates a reference to another part of this syntax.
TEXT	Indicates a keyword or punctuation (case does not matter)

Syntax



Note: <Expression> is defined under the topics: "**Syntax: IF statements in Extract Record Filter**" and "**Syntax: IF statements in Extract Column Assignment**".

Note: <Regular Expression> uses these meanings:
^ means the start of a string
$ means the end of a string
. (fullstop) means any single character
* (asterisk)means any zero or more characters

Rules for the syntax

- MATCHES can only be used in **Extract Record Filter** or **Extract Column Assignment** logic text.
- MATCHES has exactly the same effect as **LIKE** (another string comparison operator).

See also topic "**Rules for all logic text**". To find that topic in a PDF, see chapter "**Cross reference of topics and PDF files**".

Examples: MATCHES in Extract Record Filter

Character "^" means the start of a string.

Character "$" means the end of a string.

Character "." means any single character.

Character "*" means zero or more characters at this point.

Example logic text	Meaning
`SELECTIF({field1} MATCHES "MA...")`	Select input records where field1 is exactly 5 characters starting with "MA", and skip all other records.
`SELECTIF({field1} MATCHES "..VA..")`	Select input records where field1 is exactly 6 characters with characters 3 and 4 as "VA", and skip all other records.
`SELECTIF({field1} MATCHES ".....NA")`	Select input records where field1 is exactly 6 characters ending in "NA", and skip all other records.

Example logic text	Meaning
`SELECTIF({field1} MATCHES "^BBB*")`	Select input records where field1 begins with characters "BBB", and skip all other records. This example has the same effect as: `SELECTIF({field1} BEGINS_WITH "BBB")` It is better to use BEGINS_WITH because the logic text executes faster.

`IF ({field1} MATCHES "^BBB*")`
`   THEN SELECT`
`ENDIF`
	Select input records where field1 begins with characters "BBB", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} BEGINS_WITH "BBB")`
	It is better to use BEGINS_WITH because the logic text executes faster.

`SELECTIF({field1} MATCHES "*CCC*")`	Select input records where field1 contains characters "CCC", and skip all other records. This example has the same effect as: `SELECTIF({field1} CONTAINS "CCC")` It is better to use CONTAINS because the logic text executes faster.

`IF ({field1} MATCHES "*CCC*")`
`   THEN SELECT`
`ENDIF`
	Select input records where field1 contains characters "CCC", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} CONTAINS "CCC")`
	It is better to use CONTAINS because the logic text executes faster.

`SELECTIF({field1} MATCHES "*EEE$")`	Select input records where field1 ends with characters "EEE", and skip all other records. This example has the same effect as: `SELECTIF({field1} ENDS_WITH "EEE")` It is better to use ENDS_WITH because the logic text executes faster.

`IF ({field1} MATCHES "*EEE$")`
`   THEN SELECT`
`ENDIF`
	Select input records where field1 ends with characters "EEE", and skip all other records. This example has the same effect as:
	`SELECTIF({field1} ENDS_WITH "EEE")`
	It is better to use ENDS_WITH because the logic text executes faster.

`SELECTIF({field1}`
`        MATCHES "^B*C*E$")`
	Select input records where field1 begins with "B", contains "C" and ends with "E", and skip all other records.

`IF ({field1}`
`   MATCHES "^B*C*E$")`
`   THEN SELECT`
`ENDIF`
	Select input records where field1 begins with "B", contains "C" and ends with "E", and skip all other records. T

Examples: MATCHES in Extract Column Assignment

Character "^" means the start of a string.

Character "$" means the end of a string.

Character "." means any single character.

Character "*" means zero or more characters at this point.

Example logic text **Meaning**

```
IF ({field1}
   MATCHES "MA...")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

Select input records where field1 is exactly 5 characters starting with "MA", and skip all other records.

```
IF ({field1}
   MATCHES "..VA..")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

Select input records where field1 is exactly 6 characters with characters 3 and 4 as "VA", and skip all other records.

```
IF ({field1} |
   MATCHES ".....NA")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

Select input records where field1 is exactly 6 characters ending in "NA", and skip all other records.

```
IF ({field1}
   MATCHES "^BBB*")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

If field1 begins with characters "BBB" then set the current column to field1, otherwise set the current column to blank.

```
IF ({field1}
   MATCHES "*CCC*")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

If field1 contains characters "CCC" then set the current column to field1, otherwise set the current column to blank.

```
IF ({field1}
   MATCHES "*EEE$")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

If field1 ends with characters "EEE" then set the current column to field1, otherwise set the current column to blank.

```
IF ({field1}
   MATCHES "^B*C*E$")
   THEN COLUMN = {field1}
   ELSE COLUMN = " "
ENDIF
```

If field1 begins with "B", contains "C" and ends with "E" then set the current column to field1, otherwise set the current column to blank.

Chapter 5. Cross reference of topics and PDF files

How to download a PDF

Go to SAFR Information Center, select **About this Information Center** and select **PDF**. Follow the instructions on that page.

Alphabetical list of topics

Note the following:

- "**InfoCtr4150**" means the PDF called "**SAFR Information Center 4.15.00**" which contains all help topics.
- "**Top 3**" means the PDF called "**Top 3 - Admin Guide, General Users Guide and Overviews**".

Find the required topic in the first column below. The columns to the right show the PDFs that contain that topic.

Topic	**PDF**	**PDF**	**PDF**	**PDF**
Add View Source errors	Troubleshooting			InfoCtr4150
Add View Source screen help	Screens			InfoCtr4150
Admin Guide	Admin Guide		Top 3	InfoCtr4150
Administrators START HERE	Admin Guide		Top 3	InfoCtr4150
Basics of using the SAFR Workbench	Admin Guide	General Users Guide	Top 3	InfoCtr4150
Batch activate lookup paths	Admin Guide		Top 3	InfoCtr4150
Batch Activate Lookup Paths errors	Troubleshooting			InfoCtr4150
Batch Activate Lookup Paths screen help	Screens			InfoCtr4150
Batch activate views	Admin Guide		Top 3	InfoCtr4150
Batch Activate Views errors	Troubleshooting			InfoCtr4150
Batch Activate Views screen help	Screens			InfoCtr4150
Change Log Path screen help	Screens			InfoCtr4150
Changing Log Path	Admin Guide	General Users Guide	Top 3	InfoCtr4150
Checking metadata dependencies	Admin Guide		Top 3	InfoCtr4150
Clear environment	Admin Guide		Top 3	InfoCtr4150
Clear environment messages	Screens			InfoCtr4150
Column Source Properties errors	Troubleshooting			InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Column Source Properties screen help	Screens			InfoCtr4150
Common Key Buffers overview	Overviews		Top 3	InfoCtr4150
Control records overview	Overviews		Top 3	InfoCtr4150
Copying metadata	Admin Guide		Top 3	InfoCtr4150
Create New Extract Column Assignment errors	Troubleshooting			InfoCtr4150
Create New Extract Column Assignment screen help	Screens			InfoCtr4150
Create New Extract Record Filter errors	Troubleshooting			InfoCtr4150
Create New Extract Record Filter screen help	Screens			InfoCtr4150
Create New Format-Phase Calculation errors	Troubleshooting			InfoCtr4150
Create New Format-Phase Calculation screen help	Screens			InfoCtr4150
Create New Format-Phase Record Filter errors	Troubleshooting			InfoCtr4150
Create New Format-Phase Record Filter screen help	Screens			InfoCtr4150
Creating control records	Admin Guide		Top 3	InfoCtr4150
Creating environments	Admin Guide		Top 3	InfoCtr4150
Creating global fields	Admin Guide		Top 3	InfoCtr4150
Creating groups	Admin Guide		Top 3	InfoCtr4150
Creating logical files	Admin Guide		Top 3	InfoCtr4150
Creating logical records	Admin Guide		Top 3	InfoCtr4150
Creating lookup paths	General Users Guide		Top 3	InfoCtr4150
Creating physical files	Admin Guide		Top 3	InfoCtr4150
Creating user-exit routines	Admin Guide		Top 3	InfoCtr4150
Creating users	Admin Guide		Top 3	InfoCtr4150
Creating view folders	Admin Guide		Top 3	InfoCtr4150
Creating views	General Users Guide		Top 3	InfoCtr4150
Deleting metadata	Admin Guide		Top 3	InfoCtr4150
Deleting metadata messages	Troubleshooting			InfoCtr4150
Dependency Checker errors	Troubleshooting			InfoCtr4150
Dependency Checker screen help	Screens			InfoCtr4150
Do SAFR batch processes run sequentially or in parallel?				InfoCtr4150
Edit Control Record errors	Troubleshooting			InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Edit Control Record screen help	Screens			InfoCtr4150
Edit Environment errors	Troubleshooting			InfoCtr4150
Edit Environment screen help	Screens			InfoCtr4150
Edit Extract Column Assignment errors	Troubleshooting			InfoCtr4150
Edit Extract Column Assignment screen help	Screens			InfoCtr4150
Edit Extract Record Filter errors	Troubleshooting			InfoCtr4150
Edit Extract Record Filter screen help	Screens			InfoCtr4150
Edit Format-Phase Calculation errors	Troubleshooting			InfoCtr4150
Edit Format-Phase Calculation screen help	Screens			InfoCtr4150
Edit Format-Phase Record Filter errors	Troubleshooting			InfoCtr4150
Edit Format-Phase Record Filter screen help	Screens			InfoCtr4150
Edit Global Field errors	Troubleshooting			InfoCtr4150
Edit Global Field screen help	Screens			InfoCtr4150
Edit Group errors	Troubleshooting			InfoCtr4150
Edit Group screen help	Screens			InfoCtr4150
Edit Logical File errors	Troubleshooting			InfoCtr4150
Edit Logical File screen help	Screens			InfoCtr4150
Edit Logical Record errors	Troubleshooting			InfoCtr4150
Edit Logical Record (Assoc. Log. Files tab) screen help	Screens			InfoCtr4150
Edit Logical Record (LR Fields tab) screen help	Screens			InfoCtr4150
Edit Logical Record (LR Properties tab) screen help	Screens			InfoCtr4150
Edit Lookup Path errors	Troubleshooting			InfoCtr4150
Edit Lookup Path (General tab) screen help	Screens			InfoCtr4150
Edit Lookup Path (Lookup Path Definition tab) screen help	Screens			InfoCtr4150
Edit Physical File errors	Troubleshooting			InfoCtr4150
Edit Physical File screen help	Screens			InfoCtr4150
Edit User errors	Troubleshooting			InfoCtr4150
Edit User screen help	Screens			InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Edit User-Exit Routine errors	Troubleshooting			InfoCtr4150
Edit User-Exit Routine screen help	Screens			InfoCtr4150
Edit View (View Editor tab) screen help	Screens			InfoCtr4150
Edit View (View Properties, Extract Phase tab) screen help	Screens			InfoCtr4150
Edit View (View Properties, Format Phase tab) screen help	Screens			InfoCtr4150
Edit View (View Properties, General tab) screen help	Screens			InfoCtr4150
Edit View (View Properties, Header/ Footer tab) screen help	Screens			InfoCtr4150
Edit View errors	Troubleshooting			InfoCtr4150
Edit View Folder errors	Troubleshooting			InfoCtr4150
Edit View Folder screen help	Screens			InfoCtr4150
Empty "Deleted Views" folder	Admin Guide		Top 3	InfoCtr4150
Environment Checker errors	Troubleshooting			InfoCtr4150
Environment Checker screen help	Screens			InfoCtr4150
Environments - advanced overview	Overviews		Top 3	InfoCtr4150
Environments overview	Overviews		Top 3	InfoCtr4150
Examples: COLUMN and COL.nnn statements	Logic text types			InfoCtr4150
Examples: COLUMN statements	Logic text types			InfoCtr4150
Examples: IF with COLUMN and COL.nnn statements	Logic text types			InfoCtr4150
Examples: IF with COLUMN statements	Logic text types			InfoCtr4150
Examples: IF with SELECT	Logic text types			InfoCtr4150
Examples: IF with SKIP	Logic text types			InfoCtr4150
Examples: SELECTIF statements	Logic text types			InfoCtr4150
Examples: SKIPIF statements	Logic text types			InfoCtr4150
Examples: WRITE statements	Logic text types			InfoCtr4150
Export metadata overview	Overviews		Top 3	InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Export Utility errors	Troubleshooting			InfoCtr4150
Export Utility screen help	Screens			InfoCtr4150
Exporting metadata	Admin Guide		Top 3	InfoCtr4150
FAQ				InfoCtr4150
Finding and replacing logic text	Admin Guide		Top 3	InfoCtr4150
Find/Replace Logic Text errors	Troubleshooting			InfoCtr4150
Find/Replace Logic Text screen help	Screens			InfoCtr4150
Finding all environments for a particular a metadata item name	Admin Guide	General Users Guide	Top 3	InfoCtr4150
Finding screen help	Admin Guide	General Users Guide	Top 3	InfoCtr4150
General users guide	General Users Guide		Top 3	InfoCtr4150
General users START HERE	General Users Guide		Top 3	InfoCtr4150
Generating reports on metadata	General Users Guide		Top 3	InfoCtr4150
Global fields overview	Overviews		Top 3	InfoCtr4150
Glossary				InfoCtr4150
Group Membership errors	Troubleshooting			InfoCtr4150
Group Membership screen help	Screens			InfoCtr4150
Group Permissions By Environment (Component Security section) screen help	Screens			InfoCtr4150
Group Permissions By Environment (Environment, Assoc. Groups sections) screen help	Screens			InfoCtr4150
Group Permission By Environment messages	Troubleshooting			InfoCtr4150
Group Permissions (Component Security section) screen help	Screens			InfoCtr4150
Group Permissions (Groups, Assoc. Environs sections) screen help	Screens			InfoCtr4150
Group Permission messages	Troubleshooting			InfoCtr4150
Groups - advanced overview	Overviews		Top 3	InfoCtr4150
Groups overview	Overviews		Top 3	InfoCtr4150
How do I generate a Dependency Checker Report?				InfoCtr4150

Topic	PDF	PDF	PDF	PDF
How do I generate an Environment Checker Report?				InfoCtr4150
How do I generate an Environment Security Report?				InfoCtr4150
How do I generate a Logical Record Report?				InfoCtr4150
How do I generate a Lookup Path Report?				InfoCtr4150
How do I generate a View Column PIC Report?				InfoCtr4150
How do I generate a View Column Report?				InfoCtr4150
How do I generate a View Properties Report?				InfoCtr4150
How many address spaces does SAFR use?				InfoCtr4150
How many processes per address space?				InfoCtr4150
How many SAFR passes will there be?				InfoCtr4150
How scalable is SAFR in z/OS?				InfoCtr4150
I want to see more logic text for all rows in the Find/Replace Logic Text screen				InfoCtr4150
Import metadata overview	Overviews		Top 3	InfoCtr4150
Import Utility screen help	Screens			InfoCtr4150
Import Utility errors	Troubleshooting			InfoCtr4150
Importing metadata	Admin Guide		Top 3	InfoCtr4150
Logging into the SAFR Workbench	General Users Guide		Top 3	InfoCtr4150
Logic text 1: Extract Record Filter	Logic text types			InfoCtr4150
Logic text 1: Extract Record Filter overview	Overviews		Top 3	InfoCtr4150
Logic text 2: Extract Column Assignment	Logic text types			InfoCtr4150
Logic text 2: Extract Column Assignment overview	Overviews		Top 3	InfoCtr4150
Logic text 3: Format Column Calculations	Logic text types			InfoCtr4150
Logic text 3: Format Column Calculations overview	Overviews		Top 3	InfoCtr4150
Logic text 4: Format Record Filter	Logic text types			InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Logic text 4: Format Record Filter overview	Overviews		Top 3	InfoCtr4150
Logic text diagrams 1: Extract Record Filter	Logic text diagrams			InfoCtr4150
Logic text diagrams 2: Extract Column Assignment	Logic text diagrams			InfoCtr4150
Logic text diagrams 3: Format Column Calculation	Logic text diagrams			InfoCtr4150
Logic text diagrams 4: Format Record Filter	Logic text diagrams			InfoCtr4150
Logic Text Helper errors	Troubleshooting			InfoCtr4150
Logic Text Helper screen help	Screens			InfoCtr4150
Logic text overview	Overviews		Top 3	InfoCtr4150
Logic Text Validation Errors message help	Troubleshooting			InfoCtr4150
Logic Text Validation Errors screen help	Screens			InfoCtr4150
Logic text: summary diagrams	Logic text diagrams			InfoCtr4150
Logic text: syntax	Logic text syntax			InfoCtr4150
Logical files overview	Overviews		Top 3	InfoCtr4150
Logical records overview	Overviews		Top 3	InfoCtr4150
Lookup paths overview	Overviews		Top 3	InfoCtr4150
Metadata overview	Overviews		Top 3	InfoCtr4150
Metadata - advanced overview	Overviews		Top 3	InfoCtr4150
Migrate metadata overview	Overviews		Top 3	InfoCtr4150
Migrating metadata	Admin Guide		Top 3	InfoCtr4150
Migration Utility errors	Troubleshooting			InfoCtr4150
Migration Utility screen help	Screens			InfoCtr4150
Modifying control records	Admin Guide		Top 3	InfoCtr4150
Modifying Environments	Admin Guide		Top 3	InfoCtr4150
Modifying global fields	Admin Guide		Top 3	InfoCtr4150
Modifying group membership	Admin Guide		Top 3	InfoCtr4150
Modifying group permissions by environment	Admin Guide		Top 3	InfoCtr4150
Modifying group permissions by group	Admin Guide		Top 3	InfoCtr4150
Modifying groups	Admin Guide		Top 3	InfoCtr4150
Modifying logical files	Admin Guide		Top 3	InfoCtr4150
Modifying logical records	Admin Guide		Top 3	InfoCtr4150
Modifying lookup paths	General Users Guide		Top 3	InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Modifying own user account	General Users Guide		Top 3	InfoCtr4150
Modifying physical files	Admin Guide		Top 3	InfoCtr4150
Modifying user-exit routines	Admin Guide		Top 3	InfoCtr4150
Modifying users	Admin Guide		Top 3	InfoCtr4150
Modifying view folders	Admin Guide		Top 3	InfoCtr4150
Modifying views	General Users Guide		Top 3	InfoCtr4150
New Control Record errors	Troubleshooting			InfoCtr4150
New Control Record screen help	Screens			InfoCtr4150
New Environment errors	Troubleshooting			InfoCtr4150
New Environment screen help	Screens			InfoCtr4150
New Global Field errors	Troubleshooting			InfoCtr4150
New Global Field screen help	Screens			InfoCtr4150
New Group errors	Troubleshooting			InfoCtr4150
New Group screen help	Screens			InfoCtr4150
New Logical File errors	Troubleshooting			InfoCtr4150
New Logical File screen help	Screens			InfoCtr4150
New Logical Record errors	Troubleshooting			InfoCtr4150
New Logical Record (Assoc. Log. Files tab) screen help	Screens			InfoCtr4150
New Logical Record (LR Fields tab) screen help	Screens			InfoCtr4150
New Logical Record (LR Properties tab) screen help	Screens			InfoCtr4150
New Lookup Path errors	Troubleshooting			InfoCtr4150
New Lookup Path (General tab) screen help	Screens			InfoCtr4150
New Lookup Path (Lookup Path Definition tab) screen help	Screens			InfoCtr4150
New Physical File errors	Troubleshooting			InfoCtr4150
New Physical File screen help	Screens			InfoCtr4150
New User errors	Troubleshooting			InfoCtr4150
New User screen help	Screens			InfoCtr4150
New User-Exit Routine errors	Troubleshooting			InfoCtr4150
New User-Exit Routine screen help	Screens			InfoCtr4150
New View (View Editor tab) screen help	Screens			InfoCtr4150

Topic	PDF	PDF	PDF	PDF
New View (View Properties, Extract Phase tab) screen help	Screens			InfoCtr4150
New View (View Properties, Format Phase tab) screen help	Screens			InfoCtr4150
New View (View Properties, General tab) screen help	Screens			InfoCtr4150
New View (View Properties, Header/ Footer tab) screen help	Screens			InfoCtr4150
New View errors	Troubleshooting			InfoCtr4150
New View Folder errors	Troubleshooting			InfoCtr4150
New View Folder screen help	Screens			InfoCtr4150
Opening a log file	General Users Guide		Top 3	InfoCtr4150
Overviews	Overviews		Top 3	InfoCtr4150
Parallelism overview	Overviews		Top 3	InfoCtr4150
Performance Engine (PE) overview	Overviews		Top 3	InfoCtr4150
Physical files overview	Overviews		Top 3	InfoCtr4150
Pipes overview	Overviews		Top 3	InfoCtr4150
Return to login	General Users Guide		Top 3	InfoCtr4150
Rules for all logic text	Logic text diagrams	Logic text syntax	Logic text types	InfoCtr4150
SAFR Connection Manager errors	Troubleshooting			InfoCtr4150
SAFR Connection Manager screen help	Screens			InfoCtr4150
SAFR Login errors	Troubleshooting			InfoCtr4150
SAFR Login screen help	Screens			InfoCtr4150
SAFR optimization overview	Overviews		Top 3	InfoCtr4150
SAFR overview - START HERE	Overviews		Top 3	InfoCtr4150
SAFR phases overview	Overviews		Top 3	InfoCtr4150
Searching lists of metadata	Admin Guide	General Users Guide	Top 3	InfoCtr4150
Sort Key Properties errors	Troubleshooting			InfoCtr4150
Sort Key Properties screen help	Screens			InfoCtr4150
Sort Key Titles errors	Troubleshooting			InfoCtr4150
Sort Key Titles screen help	Screens			InfoCtr4150
Syntax: BEGINS_WITH	Logic text syntax			InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Syntax: COL.nnn in Format Record Filter	Logic text syntax			InfoCtr4150
Syntax: COLUMN and COL.nnn in Extract Column Assignment	Logic text syntax			InfoCtr4150
Syntax: COLUMN and COL.nnn in Format Column Calculations	Logic text syntax			InfoCtr4150
Syntax: COLUMN and COL.nnn statements	Logic text syntax			InfoCtr4150
Syntax: CONTAINS	Logic text syntax			InfoCtr4150
Syntax: ENDS_WITH	Logic text syntax			InfoCtr4150
Syntax: function ALL	Logic text syntax			InfoCtr4150
Syntax: function BATCHDATE	Logic text syntax			InfoCtr4150
Syntax: function CURRENT	Logic text syntax			InfoCtr4150
Syntax: function DATE	Logic text syntax			InfoCtr4150
Syntax: function DAYSBETWEEN	Logic text syntax			InfoCtr4150
Syntax: function FISCALDAY	Logic text syntax			InfoCtr4150
Syntax: function FISCALMONTH	Logic text syntax			InfoCtr4150
Syntax: function FISCALPERIOD	Logic text syntax			InfoCtr4150
Syntax: function FISCALQUARTER	Logic text syntax			InfoCtr4150
Syntax: function FISCALYEAR	Logic text syntax			InfoCtr4150
Syntax: function ISFOUND	Logic text syntax			InfoCtr4150
Syntax: function ISNOTFOUND	Logic text syntax			InfoCtr4150
Syntax: function ISNOTNULL	Logic text syntax			InfoCtr4150
Syntax: function ISNOTNUMERIC	Logic text syntax			InfoCtr4150
Syntax: function ISNOTSPACES	Logic text syntax			InfoCtr4150
Syntax: function ISNULL	Logic text syntax			InfoCtr4150
Syntax: function ISNUMERIC	Logic text syntax			InfoCtr4150
Syntax: function ISSPACES	Logic text syntax			InfoCtr4150
Syntax: function MONTHSBETWEEN	Logic text syntax			InfoCtr4150
Syntax: function PRIOR	Logic text syntax			InfoCtr4150
Syntax: function REPEAT	Logic text syntax			InfoCtr4150
Syntax: function RUNDAY	Logic text syntax			InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Syntax: function RUNMONTH	Logic text syntax			InfoCtr4150
Syntax: function RUNPERIOD	Logic text syntax			InfoCtr4150
Syntax: function RUNQUARTER	Logic text syntax			InfoCtr4150
Syntax: function RUNYEAR	Logic text syntax			InfoCtr4150
Syntax: function YEARSBETWEEN	Logic text syntax			InfoCtr4150
Syntax: functions	Logic text syntax			InfoCtr4150
Syntax: functions Q1, Q2, Q3 and Q4	Logic text syntax			InfoCtr4150
Syntax: IF statements	Logic text syntax			InfoCtr4150
Syntax: IF statements in Extract Column Assignment	Logic text syntax			InfoCtr4150
Syntax: IF statements in Extract Record Filter	Logic text syntax			InfoCtr4150
Syntax: IF statements in Format Column Calculations	Logic text syntax			InfoCtr4150
Syntax: IF statements in Format Record Filter	Logic text syntax			InfoCtr4150
Syntax: LIKE	Logic text syntax			InfoCtr4150
Syntax: lookup paths	Logic text syntax			InfoCtr4150
Syntax: MATCHES	Logic text syntax			InfoCtr4150
Syntax: SELECT and SELECTIF statements in Extract Record Filter	Logic text syntax			InfoCtr4150
Syntax: SELECT and SELECTIF statements in Format Record Filter	Logic text syntax			InfoCtr4150
Syntax: SELECT, SELECTIF, SKIP and SKIPIF statements	Logic text syntax			InfoCtr4150
Syntax: SKIP and SKIPIF statements in Extract Record Filter	Logic text syntax			InfoCtr4150
Syntax: SKIP and SKIPIF statements in Format Record Filter	Logic text syntax			InfoCtr4150
Syntax: string comparison	Logic text syntax			InfoCtr4150
Syntax: WRITE statements	Logic text syntax			InfoCtr4150
Syntax: WRITE statements in Extract Column Assignment	Logic text syntax			InfoCtr4150
Syntax: WRITE statements in Extract Record Filter	Logic text syntax			InfoCtr4150
Tokens overview	Overviews		Top 3	InfoCtr4150
User-exit routines overview	Overviews		Top 3	InfoCtr4150

Topic	PDF	PDF	PDF	PDF
Users overview	Overviews		Top 3	InfoCtr4150
View Activation Errors message help	Troubleshooting			InfoCtr4150
View Activation Errors screen help	Screens			InfoCtr4150
View folders overview	Overviews		Top 3	InfoCtr4150
View Source Properties errors	Troubleshooting			InfoCtr4150
View Source Properties screen help	Screens			InfoCtr4150
Views - advanced overview	Overviews		Top 3	InfoCtr4150
Views overview	Overviews		Top 3	InfoCtr4150
WE log file overview	Overviews		Top 3	InfoCtr4150
WE Security overview	Overviews		Top 3	InfoCtr4150
What are the keyboard shortcuts?	Admin Guide	General Users Guide	Top 3	InfoCtr4150
What does SAFR stand for?				InfoCtr4150
What does WE stand for?				InfoCtr4150
What metadata do I want to see?	General Users Guide		Top 3	InfoCtr4150
What's new in this Information Center				InfoCtr4150
Where is the WE log file?	Admin Guide	General Users Guide	Top 3	InfoCtr4150
Why use SAFR?				InfoCtr4150
Workbench overview	Overviews		Top 3	InfoCtr4150
XML structure for metadata overview	Overviews		Top 3	InfoCtr4150

Appendix: Notices

This information was developed for products and services offered in the U.S.A.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA

3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Index

Printed in USA