Extracting Business Process Data using EntireX Process Extractor

Extracting Business Process Data using EntireX Process Extractor

This chapter covers the following topics:

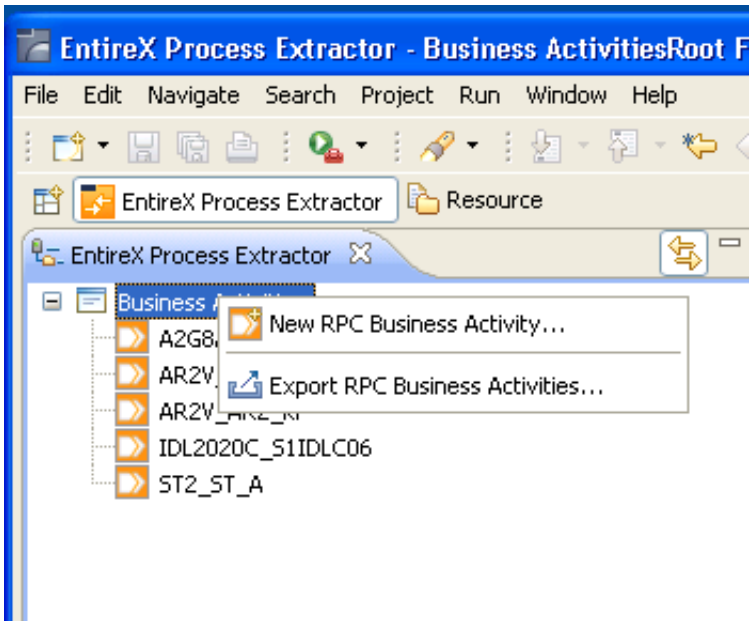
Introduction

Extracting business process data requires the design-time component of EntireX Process Extractor. During design time, the RPC calls to be extracted are defined. For each RPC program, an RPC business activity is defined, which specifies the content of the business process data for ARIS PPM.

Creating a New RPC Business Activity

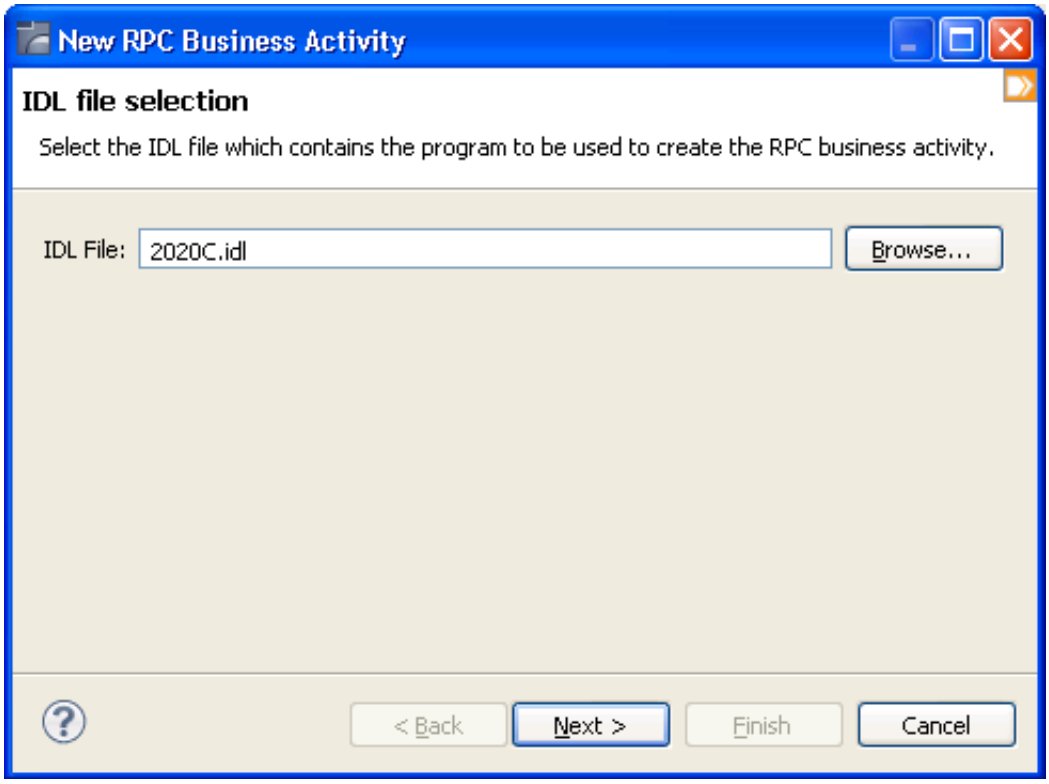
To create a new RPC business activity

1. Within the EntireX Process Extractor Perspective, right-click on the Business Activities container and choose **New RPC Business Activity...**.

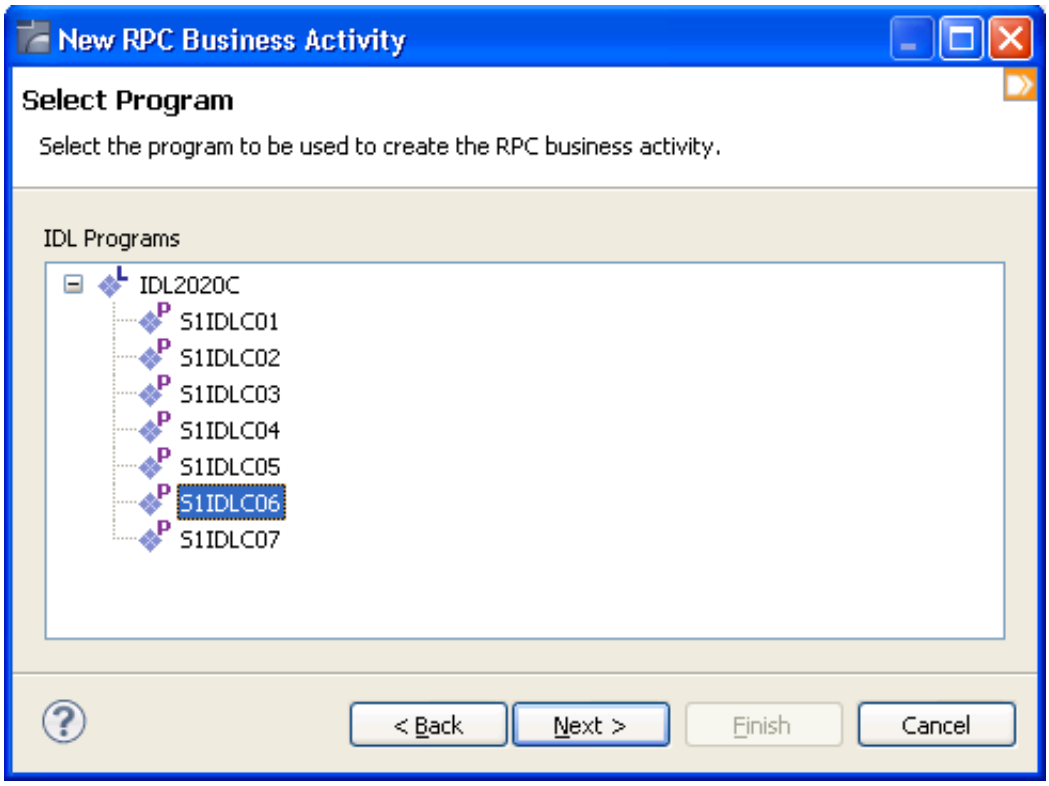


Or:
Choose **File > New > Other... > Software AG > EntireX Process Extractor > RPC Business Activity**.

The **New RPC Business Activity** dialog box is displayed.



2. Enter or select a relevant IDL file. Click **Next**. The **Select Program** dialog box is displayed.



3. Enter a name for the entity which is to be created. The default name provided is: *<library name>_<program name>*.
4. Enter a relevant description.
5. Select or enter a process type. The process type must be the same as the process type used in ARIS PPM.
6. Click **Finish**. The activity appears in the tree and the activity's editor is opened.

To edit an existing RPC business activity

- Within the EntireX Process Extractor Perspective, double-click on a business activity or right-click on it and choose **Open**.

To rename an RPC business activity

1. Within the EntireX Process Extractor Perspective, right-click on a business activity and choose **Rename...**, or from the **File** menu, choose **Rename...**.
2. Enter the new name. Click **Finish**.

To delete an RPC business activity

1. Within the EntireX Process Extractor Perspective, right-click on a business activity and choose **Delete**.

2. Confirm that you wish to delete the entity.

Creating User-defined Attributes

The output attributes consist of general PPM attributes and user-defined attributes. General PPM attributes cannot be deleted or edited. New user-defined attributes can be created, edited and deleted. In this section, we will explain how to create, edit and delete user-defined attributes.

User-defined attributes can be created in two ways: using the **User-defined Attributes** root (on the right side of the Mapper view), or using the "Quick Creation" method (on the left side of the Mapper view), initiated from the input that is to be mapped to the newly created user-defined attribute.

To create a user-defined attribute using the User-defined Attributes root node

1. In the Mapper area of the EntireX Process Extractor Perspective, right-click on the **User-defined Attribute** root node and choose **Add User-defined Attribute...**.
2. Enter a name for the new attribute.
3. Select a type.
4. Click **Finish**. The newly created attribute appears as a child node of the **User-defined Attributes** root node.
5. You can map inputs to this attribute by clicking on the relevant input and dragging the mouse to the newly added attribute.

To create a user-defined attribute using the "Quick Creation" method

1. In the Mapper area of the EntireX Process Extractor Perspective, select the Input attribute to which you want to map the user-defined attribute.
2. Either right-click on the attribute and choose **Map to Procedure Output**, or drag the mouse from this attribute to the user-defined attribute node.
3. A new attribute is created within the user-defined attribute root node. The name of the attribute is the name of the input, and the type of the attribute is the type of the input attribute. The input attribute is now mapped to the newly created user-defined attribute.

   The default mappings for the data types are listed below:

IDL Data Type	PPM Attribute Data Type
A*n*, AV, AV*n*	Text
U*n*, UV, UV*n*	Text
I1, I2, I4	long
F4	Float
F8	Double
D, T	Date
N, NU, P, PU	Double
L	Boolean

To edit an existing user-defined attribute

- In the Mapper area of the EntireX Process Extractor Perspective, double-click on the user-defined attribute or right-click on it and choose **Edit Attribute...**.

To delete a user-defined attribute

1. In the Mapper area of the EntireX Process Extractor Perspective, right-click on the user-defined attribute and choose **Delete Attribute**.
2. Confirm that you wish to delete the entity.

Mapping a Program’s Inputs and Outputs to PPM Attributes

In addition to mapping attributes, you can also determine the relevant process type. The process type must be the same as the process type used in ARIS PPM.

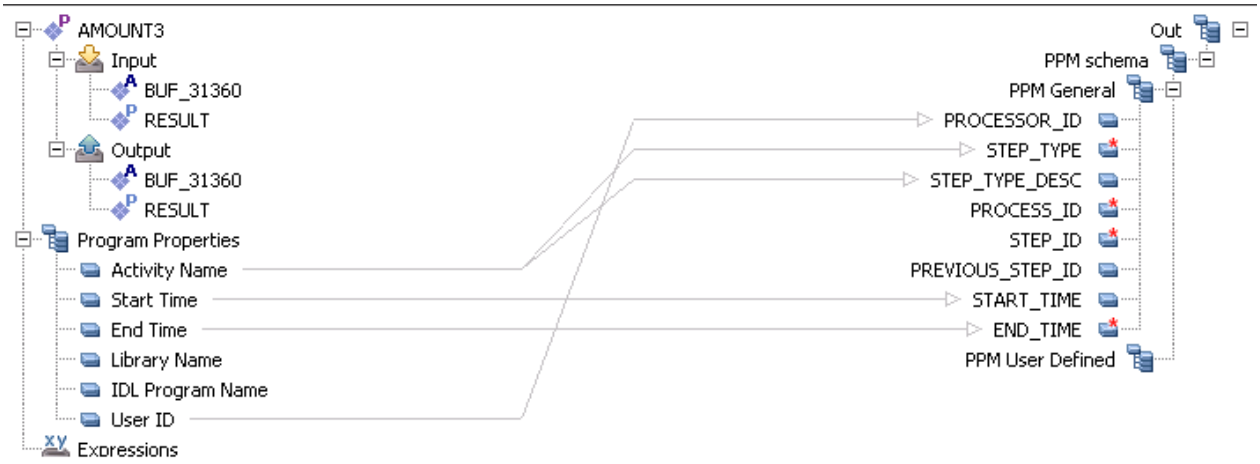
Within the Activity’s editor, you can map all the program attributes to PPM attributes. These will then be part of an XML file that will be analyzed using ARIS PPM.

To edit the business activity and map attributes

1. In the tree, double-click on the RPC business activity you created. The Editor will display the entity.
2. Check that the process type is correct, and if necessary select or type in the relevant type.
3. In the Mapper area, the left side of the map schema includes the IDL program inputs and outputs, and the right side of the map schema contains the PPM general and user-defined attributes. To map IDL program inputs and outputs from the left side of the map to outputs on the right side of the map, click on the desired input and drag the mouse across to the output on the right. When mapping an attribute that is an array, "link properties" is opened to allow defining the index (the default index is "0"). See Using the Mapper for more details.
4. Click **Add Expression** to add expressions as required. See *General Expressions* for more details.
5. Save the editor.

Using the Mapper

The Mapper tool enables mapping source elements to target output elements. The left-panel contains the entire source data elements that are available (these element may also be expressions). The right-panel contains the potential target elements. Use a drag-and-drop operation to define that when the Mapper is executed, the value/s in the source element will be copied to the target element. These definitions are indicated by lines, linking between the source element and the target element, and also are listed under the relevant node in the procedure nodes tree.



There are a number of different types of source and target elements that can be mapped:

- Mapping a Simple Type Element to a Simple Type Element
- Mapping an Array Type Element to a Simple Type Element
- Mapping Multiple Array Levels to a Simple Type Element

Mapping a Simple Type Element to a Simple Type Element

Drag the element in the source frame of the mapper to the relevant element in the target frame. When the mapping is executed, the source element will be copied to the target element.

Mapping an Array Type Element to a Simple Type Element

To map a value from an array type element to a simple type element, it is necessary to define the index of the relevant value in the array. The default index is [0].

Only one-dimensional arrays are allowed.

To place an array type element value in a simple type element

1. Drag the element in the source frame of the mapper to the relevant element in the target frame. When the Mapper is executed, the source element will be copied to the target element. A line will be displayed between the two, indicating that the value from the source element will be placed in the target element.

2. Right-click on the mapping to display the **Link Properties** dialog box. In the **Source Index** field, right-click on <exp> to set a value for the index. To set a static value, use the "Free Text" expression.

3. Click **OK** to close the dialog box.

Mapping Multiple Array Levels to a Simple Type Element

To map a value from within an array that has multiple levels to a simple type element, create a new "Value Of" expression, and select the element you wish to map. A dialog box is displayed with the expression and the required index numbers. Right-click on each index and choose "Free Text" and enter the relevant index.

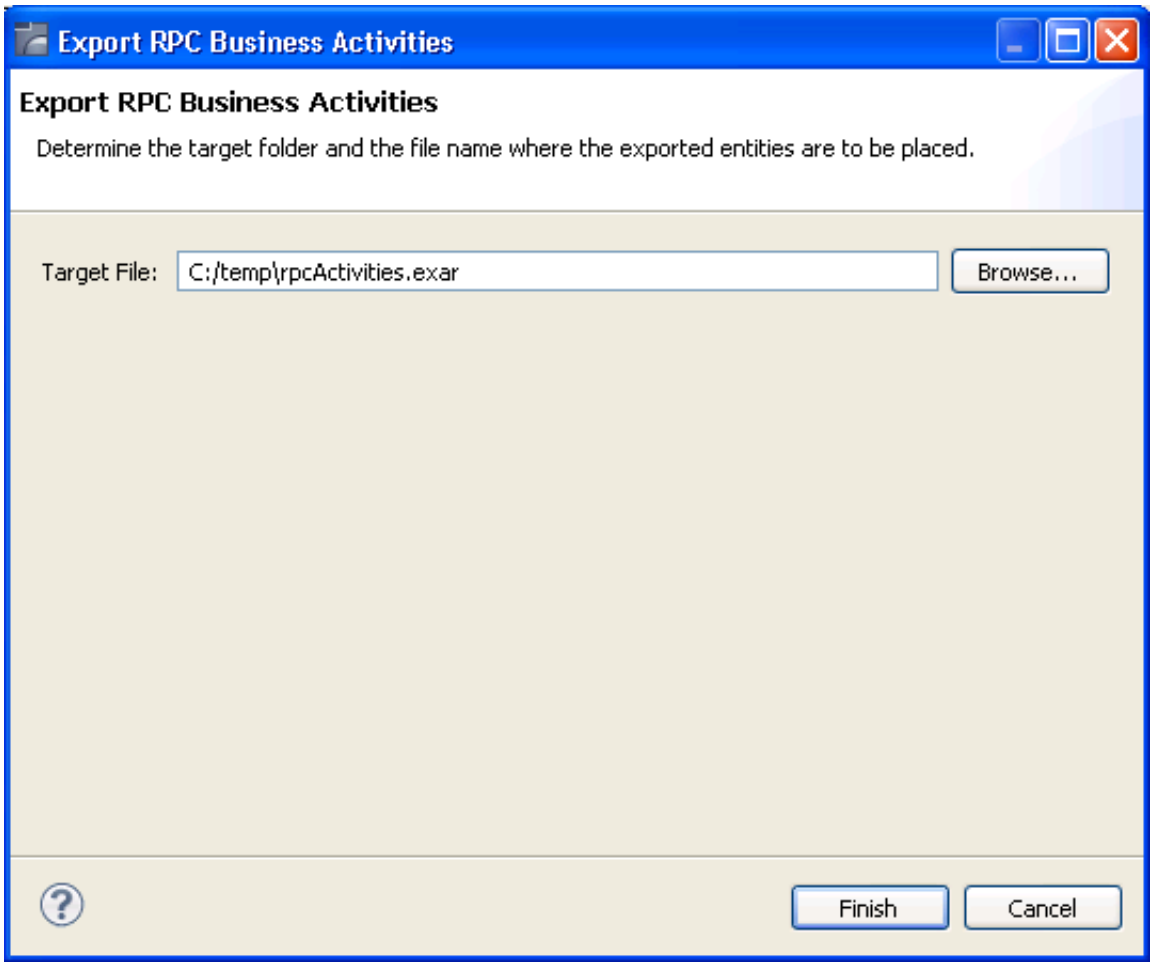
Exporting RPC Business Activities to an exar File

The exporting process exports all the RPC business activities in your application to a file with extension .exar. This file can then be used to create an XML file that can be analyzed by ARIS PPM.

To export business activities

1. Right-click on the RPC **Business Activities** node and choose **Export RPC Business Activities...**.

   Or:
   From the **File** menu, choose **File > Export > Software AG > RPC Business Activities**. The Export RPC Business Activities wizard is displayed.



2. Browse and select the location where the file is to be saved and enter a name for the file.
3. Click **Finish**.
4. The console area displays information as to the number of exported entities and also lists business activities that are not properly defined.

General Expressions

Expression Types

Expressions are used to map compound values to PPM attributes. See Using the Mapper. The object of an expression is to perform the computation indicated by the elements of the expression and to return a value that is the result of the computation. The expression types available vary according to the node you are defining.

- EmptyString, TRUE and FALSE Expressions
- Free Text
- Value Of
- Count Of
- Conditional Operator
- String Array
- Now
- Create Date
- To Date
- Date Part
- Compare
- Logical And/Or
- Is Null
- Calculate
- Ceil
- Floor
- Round
- Absolute
- Concat
- Trim
- StrIn
- SubString
- Replace String
- Change Case
- StringLength
- Reverse
- FormatDate
- Format Number
- Extract Number
- Character

EmptyString, TRUE and FALSE Expressions

Standard fixed syntax, used for these functions.

Free Text

In the **Free Text** dialog box, type in any text and add tokens in order to use values from the context. Click **Finish**. Click on the var link to define the variable.

Example

Enter text and replaceable tokens in the **Free Text** tab: "Your account number is - $(var)" . "$(var)" being a replaceable token. Click **Finish**. Use the Value Of expression to define a value for the token.

Refer to Mapper

Value Of

The **Value Of** expression returns the value of the selected object.

To define the Value Of expression

1. Select an item from the available scope.
2. Double-click to select this item as the expression's value.

Note:
When selecting an expression that has a complex input or output structure that includes arrays, it is possible to select a specific index.

Count Of

The **Count Of** expression returns a count of an array item.

To define the Count Of expression

1. Select an array item.
2. Double-click to select this item as the expression's value.

Conditional Operator

The **Conditional Operator** expression is short-hand for an if-else statement. The Conditional Operator returns `<expr1>` if `<condition>` is true or returns `<expr2>` if `<condition>` is false.

Format

```
If <condition> Then <expr1> Else <expr2>
```

Implementation

Click on each `<expr>` and define the expression.

Example

```
If (( In/AccountNumber ) = 23453) Then TRUE Else FALSE
```

String Array

Returns a string array.

Format

```
StringArray (expr, ...)
```

Implementation

Use the <expr> link to define the first string. To define additional strings click the "..." link.

Now

Now expression returns the current date and time according to the setting of your computer system’s date and time.

Create Date

Create Date returns a date for a specified year, month, day, hour, minutes and seconds.

Format

```
CreateDate (<year> , <month> , <day> , <hour> , <minute> ,
<second>)
```

Implementation

Click the links to define the expressions for the different parts of the date/time.

Example

`CreateDate (1982 , 07 , 19 , 09 , 20 , 13)` will return "1982-07-19 09:20:13:000"

To Date

To Date creates a date from a date/time string according to the given date format.

Format

```
ToDate (<datestring> , <format>)
```

Implementation

Click the links to define the expressions for the date string and format.

Example

`ToDate (19/07/1982 , dd/MM/yyyy)` will return "1982-07-19 00:00:00:000"

Date Part

Date Part extracts a part of the date (year, month, hour etc.) from a date expression.

Format

```
YearOf(<date>)

MonthOf (<date>)

DayOf(<date>)

HourOf(<date>)

MinuteOf(<date>)

SecondOf(<date>)
```

Implementation

Select the date part: year, month, day, hour, minute or second. Click on the date expression and define the Date expression.

Example

`YearOf (Now)` will return "2004"

Compare

Compare expression compares the values of two numeric or textual expressions.

Note:
When comparing two null expressions, the function will return "false".

Format

```
Is (<expr> = <expr>)

Is (<expr> > <expr>)

Is (<expr> < <expr>)

Is (<expr> >= <expr>)

Is (<expr> <= <expr>)
```

Implementation

Click `'Is'/'Is not'` to switch between the two options. Select the required comparison operator. Click the `<expr>` links to edit.

Logical And/Or

An expression that applies a logical AND or OR to several boolean expressions.

Format

```
Is (<expr> AND ...)

Is Not (<expr> AND ...)
```

Implementation

Click `'Is'/'Is not'` to switch between the two options. Click the `<expr>` or `"..."` links to add expressions. Select the required boolean operator (AND or OR).

Is Null

Is Null checks whether the selected object does not have an actual during runtime.

To define the Is Null expression:

1. Select an item from the available scope.
2. Click to select this item as the expression's value.

Calculate

Calculate returns a calculation and may include variables and arithmetic calculations.

Implementation

Type in the calculation formula using digits and operators. Click **Finish** . Click on the variable link to define an expression. A token representing this expression will appear in the calculation.

Ceil

Returns the smallest value that is not less than the argument and is equal to a mathematical integer. The value is displayed in double format.

Format

```
Ceil(<expr>)
```

Implementation

Click `<expr>` to define the relevant expression.

Example

`ceil(2.645);` will return "3.0"

Floor

Returns the largest value that is not greater than the argument and is equal to a mathematical integer. The value is displayed in double format.

Format

```
Floor(<expr>)
```

Implementation

Click `<expr>` to define the relevant expression.

Example

`floor(2.645);` will return "2.0" .

Round

Returns the closest integer to the argument.

Format

```
Round(<expr>)
```

Implementation

Click `<expr>` to define the relevant expression.

Example

```
round(2.500); will return 3.

round(2.499); will return 2.
```

Absolute

Returns the absolute value of the argument. The value is displayed in double format.

Format

```
Absolute(<expr>)
```

Implementation

Click `<expr>` to define the relevant expression.

Example

```
Absolute(2.300); will return 2.0.
```

Concat

Returns a string value containing the concatenation of two or more supplied strings.

Format

```
Concat(“<expr>”,…)
```

Implementation

Use the <expr> link to define the first string. To define additional strings click the "..." link.

Example

`Concat(“John“, “Smith”,…)` will return “ `JohnSmith` ”.

Trim

Trim expression returns a string containing a copy of a specified string with no leading or trailing spaces.

Format

```
Trim(<expr>)
```

Implementation

Click the `<expr>` to define the string expression to trim.

Example

`Trim(“ John “)` will return “ `John` ”.

StrIn

StrIn expression returns the position of the first occurrence of one string within another.

Format

```
StrIn (<string> , <substring>, <case sensitive>)

StrIn (<string> , <substring>, <case insensitive>)
```

Implementation

Click `<string> , <substring>` to define the string in which to search and the string to search for. The expression will search for the first occurrence of the second string within the first string. Toggle between **case insensitive** and **case sensitive** to determine case sensitivity.

Example

`StrIn (“Catwalk”, “Cat”)` will return "0"

`StrIn (“John”, “Smith”)` will return "–1"

`StrIn (“Caterpillar”, “pillar”)` will return "6"

SubString

SubString expression returns a substring that begins at a specified location, and has a specified length.

Format

```
SubString (<string> , <start> , <length>)
```

Implementation

Click the links to define the original string, the start index and the required length of the substring.

Example

`SubString (“Caterpillar”, 6, 6)` will return "pillar" .

Replace String

Replaces the first substring in this string that matches the given pattern, with the defined replacement.

Format

```
ReplaceString( <string> , <patternToReplace> , <replacement> )
```

Implementation

Click the links to define the original string, the regular expression pattern to be replaced, and the replacement string.

Example

`ReplaceString("elephant", "e..a", "ega")` will yield the string "elegant" .

Change Case

Change Case expression returns a string that has been converted to a specified case (lowercase or uppercase).

Format

```
ToLowerCase (<expr>)

ToUpperCase (<expr>)
```

Implementation

Select the relevant option to transform the expression to upper or lower case. Use the link to define the expression.

Example

`ToLowerCase (“JOHN”)` will return "john" .

`ToUpperCase (“john”)` will return "JOHN" .

StringLength

StringLength expression returns the length of a string.

Format

```
StrLen (<expr>)
```

Implementation

Click the `<expr>` to define the string.

Example

`StrLen (“John”)` will return "4" .

Reverse

Reverse expression returns the reverse of a string expression.

Format

```
Reverse (<expr> )
```

Implementation

Click the `<expr>` to define the string expression.

Example

`Reverse (“caterpillar”)` will return "rallipretac" .

FormatDate

FormatDate expression converts a date/time object into a date/time string, according to the given date format.

Format

```
FormatDate (<date> , <format>)
```

Implementation

Click the links to define the expressions for the date/time object and format.

Example

`FormatDate (Now, “dd/MM/yyyy”)` , Now expression being the current date and time, will return "19/07/1982" .

Format Number

Formats a number according to the given format number.

Format

```
FormatNumber( <number> , <format> )
```

Implementation

Click the links to define the expressions for the number and format.

Example

For example if the number 18734573.07 is required as 18,734,573.07, use the format "#,##0.00". Refer to Number Format for further explanation about the format syntax.

Extract Number

Extract Number expression extracts a numeric value from a textual source number. When there is more than one number, it extracts the first number it locates. This expression may be used when needing to perform calculations on the source number.

Format

```
ExtractNumber (<expr>, Decimal:dot)
```

Implementation

Select the relevant decimal symbol: dot or comma. The separator that you do not select will be recognized as the thousand separator and will be removed. Click the link to define the source expression.

Example

When selecting the dot separator, `ExtractNumber ("1,000,876.321")` will return "1000876.321" When selecting the comma separator, `ExtractNumber ("1.000.876,321")` will return "1000876.321"

Character

Character expression returns an ASCII or Unicode Character according to the decimal representation.

Implementation

Insert the character's ASCII code or Unicode value.

Example

Enter "13" , the text in the **Value** field will display "carriage return" indicating the functionality.