LV
TEKLYNX
LABELVIEW
TEKLYNX
LABELVIEW
Barcode Better
powered by:
TEKLYNX
Barcode Better

# TUTORIAL

The information contained in this guide is not of a contractual nature and may be subject to change without prior notice.

The software described in this guide is sold under a license agreement. The software may be used, copied or reproduced only in accordance with the terms of the agreement.

No part of this guide may be copied, reproduced or transmitted in any form, by any means or for any purpose other than the purchaser’s own use without the written permission of TEKLYNX Corporation SAS.

©2021 TEKLYNX Corporation SAS,

All rights reserved.

# Table of Contents


About this manual ..... 5
Typographical conventions ..... 5
About your product ..... 5
Connecting to a database ..... 6
Overview ..... 6
Installing an ODBC data source ..... 7
Importing data ..... 7
Creating variable objects ..... 8
Active Query Builder ..... 9
Getting started ..... 9
Adding an object to the query ..... 10
Editing object properties ..... 11
Joining tables ..... 11
Sorting output fields ..... 12
Defining criteria ..... 12
Defining a parameterized query ..... 14
The Query result grid ..... 14
Database Manager ..... 16
Database Structure Window ..... 16
Choose a database from the list of connections ..... 16
Choose a table in a database ..... 16
Add a table to the active database ..... 17
Delete a table in the active database ..... 17
View/hide active table's data ..... 17
Define a key field ..... 17
Define a field's content type ..... 18
Define a field's maximum size ..... 18
Allow an empty field ..... 18
Edit Database window ..... 18
Select records according to their content ..... 19
Select all identical records ..... 19
Select an identical record ..... 19
Create a new record ..... 19
Modify a record ..... 19
Delete a record ..... 20
Database Query window ..... 20
Add a query ..... 20
Select/deselect one or more fields ..... 20
Modify the order of fields selected ..... 21
Create a filter using predefined data ..... 21
Apply a logical operator to several filters ..... 21
Remove a filter ..... 21
Modify a filter in SQL ..... 21
The Print window ..... 22
Select a document to be printed ..... 22
Select an existing label template ..... 22
Select a document from a field ..... 23
Select a printer ..... 23
Formulas ..... 24
The Formula data source ..... 24
About functions ..... 24

Operators ..... 24
Arithmetic operators ..... 25
Comparative operators ..... 25
Concatenation operator ..... 25
Logical operator ..... 25
Check character calculation functions ..... 26
Conversion functions ..... 28
ATA 2000 Conversion functions ..... 34
Date and time functions ..... 36
Logical functions ..... 42
Mathematical functions ..... 43
Text functions ..... 46
Defining the properties of a Formula data source ..... 51
Exercise: Creating a specific modulo ..... 51
To calculate the weight ..... 52
To add up the result of the weight calculation: ..... 52
To calculate the check character: ..... 52
To calculate the data to be encoded: ..... 53
To create the barcode: ..... 53
Installing the network version ..... 54
Description ..... 54
Network Installation procedure ..... 54
Network configuration ..... 54
Description of Network License Manager ..... 54
Installing the Network License Manager on the server ..... 55
Configuration ..... 55
Starting the License Service ..... 56
To start the License Service Controller ..... 56
Installing the software on the workstations ..... 56
To install the software on a workstation ..... 56

# About this manual

## Typographical conventions

This manual distinguishes between different types of information using the following conventions:

- Terms taken from the interface itself, such as commands, appear in **bold**.
- Key names appear in all caps. For example: "Press the SHIFT key."
- Numbered lists indicate that there is a procedure to follow.
- When the conjunction -or- appears next to a paragraph, it means there is the choice of another procedure for carrying out a given task.
- When a menu command contains submenus, the menu name followed by the command to select appears in bold. Thus, "Go to **File > Open**" means choose the **File** menu, then the **Open** command.

## About your product

Some of the functions described in this manual may not be available in your product.

For the complete list of specific features available in your software, refer to the specification sheet provided with the product.

# Connecting to a database

## Overview

In this chapter we are going to link a label (the container) with a database (the content). To do this, we will use ODBC (Open DataBase Connectivity) or OLE DB connections.

Databases allow you to store data. All data is organized into two-dimensional tables in what is called a relationship. Each row in a table is called a record. The purpose of a record is to manage an object, the properties of which are organized across the different columns of the table in the form of fields.

A database can contain a number of tables. To link the different tables within a given database, we use joins. A concrete example later in this chapter demonstrates how joins are created.

#### ODBC

ODBC data sources make it possible to access data belonging to a wide range of database management systems. ODBC makes it easy to link an application such as your label design software with a certain number of databases. The software comes with several ODBC drivers. These enable you to access the most common types of databases.

Some of the more common drivers are listed below:

- Microsoft Access Driver (*.mdb)
- Microsoft Excel Driver (*.xls)
- Microsoft FoxPro Driver (*.dbf), etc.

#### OLE DB

OLE DB is a set of interfaces that gives access to all data, regardless of type, format or location. It provides components such as access interfaces, query drivers, and so on. These components are called "providers".

The example below describes a connection process when a database is not connected to your software.

# Installing an ODBC data source

The process described below uses the direct creation mode. If you want, you can use the wizard by selecting **Wizard** in the context menu.

**Connecting to the TKTraining.mdb database**

1. In the program, choose **Tools > Administrator ODBC**.
2. Click on the **System Data Source** tab (DSN), and then click **Add**.

> **Note**: You can define data sources with system Data Sources Names (DSNs). These data sources are unique to a specific computer but not to a specific user. Any user with the necessary rights can access a system DSN.

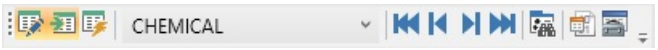
3. Select **Microsoft Access Driver** then click **Finish**.
4. Enter "TK Training Level 2" in the **Data source name** field.
5. Click **Select** and select the database TKTraining.mdb database, which can be found in the InstallDir\Samples\Forms\Tutorial directory.
6. Click the **Options** button. Check the **Read Only** option. This option makes it possible to open the database at the same time as your labelling software without causing any read/write problems.
7. Click **OK** in the **ODBC Microsoft Excel Setup** dialog box.

# Importing data

After the database is connected to your software, you have to connect it to your document.

1. Open the  PRODUCT_WS3 label.
2. Choose **Data sources > Database > Create/Modify query**.
3. Select TK Training Level 2 in the **Select data source** list.
4. Select "Fruits" in the **Select table** list.
   The database fields appear in the **Select fields** list.
5. Select the "ProdName", "Origin", "Weight", and "Reference" fields.
6. Click the button. It allows the selected records to be sorted into alphabetical or numeric order, ascending, or descending.
7. Select "Reference" as the **Sort Key** and "Ascending" as the **Sort Order**.
8. Save the query in the InstallDir\Samples\Forms\Tutorial\PRODUCT_WS4_ODBC.CSQ directory.
9. Click **OK**.
   The variables are created automatically and are listed in the **Database** branch in the **Data sources** view.

To view or print the different values that your object can take, use the navigation bar. You can also print from the **Query result** window.




## Creating variable objects

1. Select the created variables listed in the **Database** list in the **Data sources** view, then drag and drop it into the workspace.
2. Select **Text** in the context menu.

# Active Query Builder

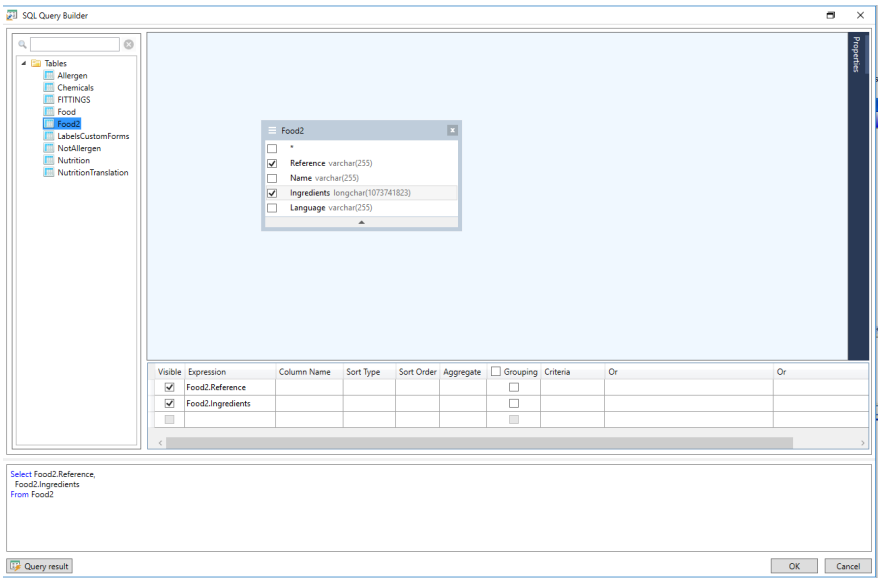
Active Query Builder is a visual query builder component that allows you to build complex SQL queries using an intuitive visual query building interface.

To work with Active Query Builder, you must have basic knowledge of SQL concepts. Active Query Builder will help you to write correct SQL code hiding technical details, but you must have an understanding of the SQL principles to achieve desired results.

## Getting started

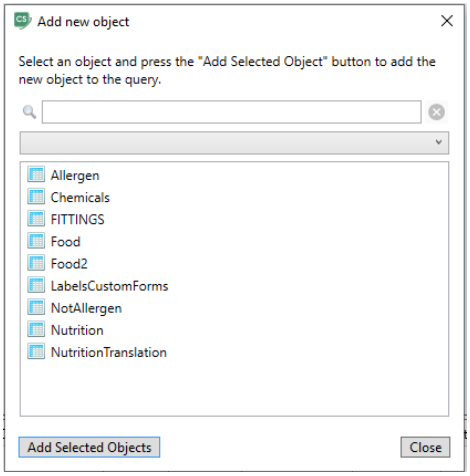
The main Active Query Builder window is divided into the following parts:

- The **Query Building area** is the main area where the visual representation of query will be displayed. This area allows you to define source database objects and derived tables, define links between them and configure properties of tables and links.
- The **Columns pane** is located below the Query Building area. It is used to perform all necessary operations with query output columns and expressions. Here you can define field aliases, sorting and grouping, and define criteria.
- The **Table view area** is located at the left. Here you may browse your query and quickly locate any part of it. Use the **Search** field at the top of the pane to specify your search.
- The page control above the Query Building area will allow you to switch between the main query and sub-queries.
- The small area in the corner of the Query Building area marked by the letter **Q** is the union sub-query handling control. Here you can add new union sub-queries and perform all necessary operations.




## Adding an object to the query

To add an object to the query, right-click the Query Building area and select the **Add Object** item in the drop-down menu.

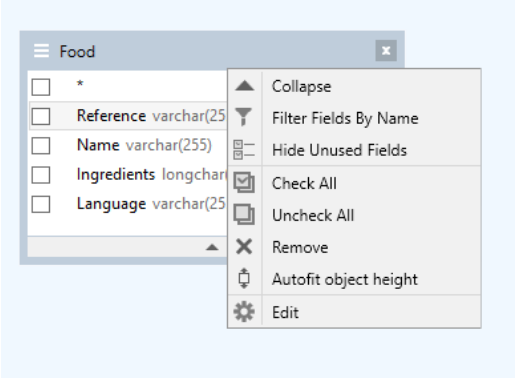


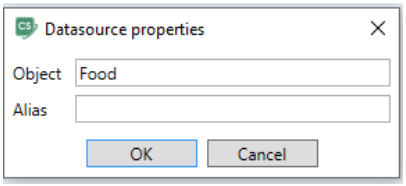

The **Add New Object** window allows you to add multiple objects at once. Table elements are listed in the combo box. **Show all objects** filter will show all objects available for selection. **Search field** at the top of the window is used to look for a needed object. You may select one or several objects by holding the **CTRL key** and then press the **Add Selected Objects** button to add these objects to the query. You may repeat this operation several times. After you finish adding objects, click the **Close** button to hide this window.

To remove an object from the query, click the **Close** button in the object header.

## Editing object properties

You can change the properties of each object added to the query by right-clicking the object and selecting the **Edit** option from the drop-down menu or simply by double-clicking the object header.

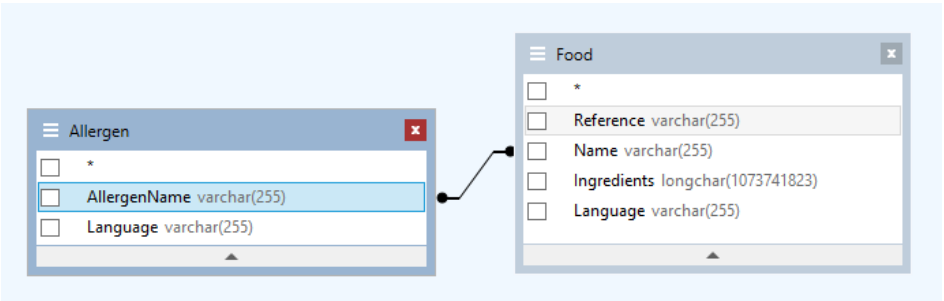



The **Datasource Properties** dialog may vary from server to server, but the **Alias** property is the same for all database servers.

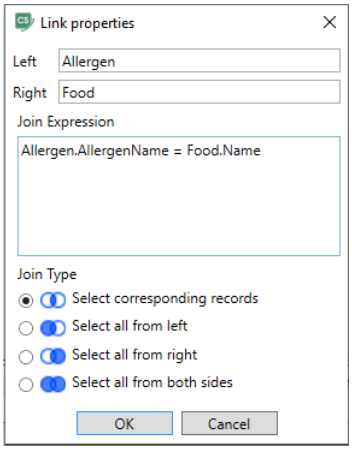
## Joining tables

To create a link between two objects (i.e., join them), you should select the field to link the objects and drag it to the corresponding field of the other object. After drag the field to the other object, a line connecting the linked fields will appear.




The default join type is INNER JOIN. This means that only matching records in both tables will be included in the resulting dataset. To define other types of joins, right-click the link and select the **Edit**

option in the drop down menu or double-click it to open the **Link Properties** window. This window allows you to define join type and other link properties.




To remove a link between objects, right-click the link line and select the **Remove** option in the drop-down menu, or select the link line and press the **Delete** button.

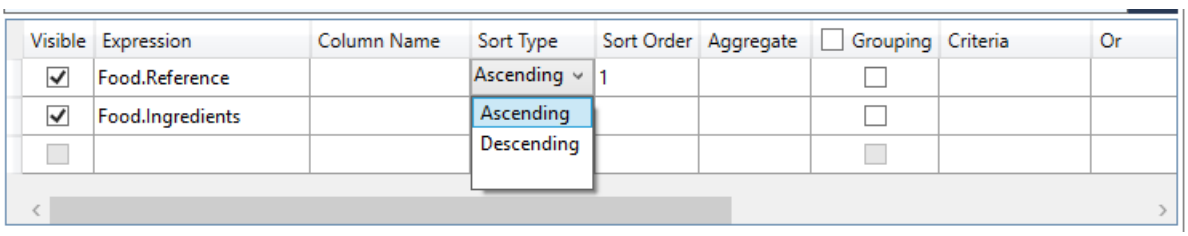
## Sorting output fields

To sort of output query fields, use the **Sort Type** and **Sort Order** columns in the **Columns Pane**.

The **Sort Type** column allows you to specify the way the fields will be sorted, either in the **Ascending** or **Descending** order.

The **Sort Order** column allows you to set up the order in which fields will be sorted if more than one field will be used for sorting..

To disable sorting by a field, clear the **Sort Type** column for the field.




## Defining criteria

To define criteria for the expression listed in the **Columns Pane**, use the **Criteria** column.

In this column, write the criterion omitting the expression itself. For example, to get the following criterion in your query:

**WHERE** (field >= 10) **AND** (field <= 20)
you should write
>= 10 **AND** <= 20  in the **Criteria** column.

You can also use the **dropdown list** in the **Criteria** column in order to paste common expressions and operators into the edit field. For advanced criteria creation use the **Expression editor** button in the **Criteria** column. **Expression Editor** dialog is displayed.

You can specify several criteria for a single expression using the **Or** columns. These criteria will be concatenated in the query using the **OR** operator.

# Defining a parameterized query

**Query Builder** lets you create a parameterized query where the value of the parameter is held in a variable.

**Note**: You must have created a variable beforehand.

1. Drag and drop the table on which your query is to be carried out.
2. Select the field(s) to which the criterion or criteria is/are to apply.
3. In the **Criteria** column or in the SQL-format edit field, specify the variable to be used as the object of a search criterion.

   **Example**: To look for the value of the variable Var0 which you created beforehand:

   - **In SQL:**

     ```
     SELECT [Table].*
     FROM [Table]
     WHERE [Table].Champ = APPLICATION.DOCUMENT.Var0
     ```

   - Criterion column
     = APPLICATION.DOCUMENT.Var0

4. Click the **Query result** button to display the result of your query.

# The Query result grid

To access the **Query result** grid, click the button in the Defining a query dialog box, in the **Merge database** browser toolbar or via the **Data sources > Database > View the query resulting data** menu.

This grid allows the result of a query to be displayed and allows you to search for a particular term and all its occurrences and print the corresponding labels.

**Note: "<Binary data>"** tag is shown in the Query result dialog for database fields that contain **BLOB data type** (e.g., images files).

The **Query result** grid contains:

- **Search functions** :

  **Search** field, which allows you to enter the field you want to search.

  Data to search for, which allows you to enter the value to search for.

  Search for the value anywhere in the field or at the start of the field .

- **Navigation functions for browsing query result records:**

  First record

  Previous record

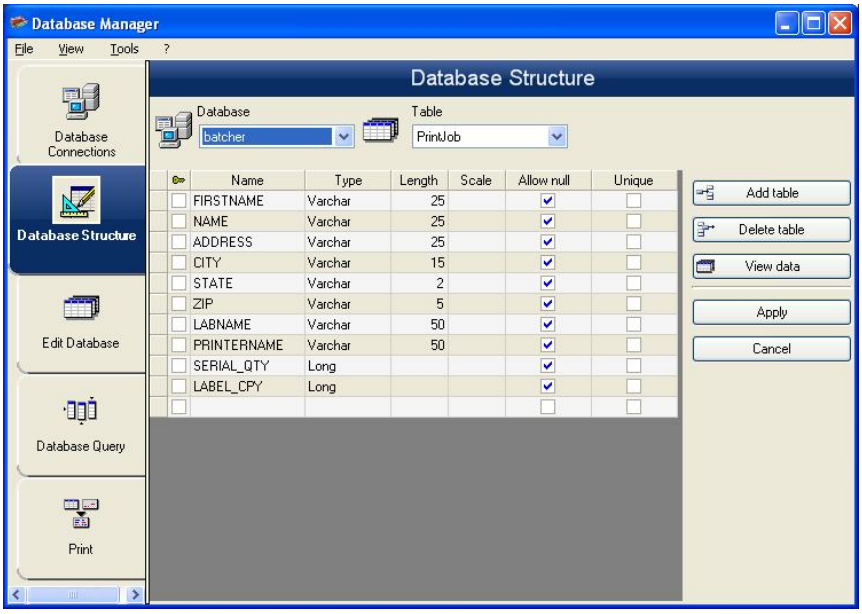
  Next record

  Last record

- **The results grid**
  Displays the search results.
- **Requery**
  Requery the request and update the grid.

# Database Manager

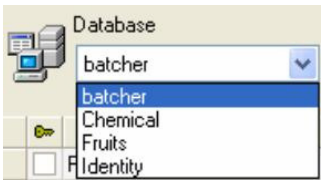
## Database Structure Window




The Database Structure window is used to manage the structure of the database file. From this window, you can add, modify or delete tables/fields, etc.

### Choose a database from the list of connections

1. Click the **Database** drop-down list.
2. Select the required database.




### Choose a table in a database

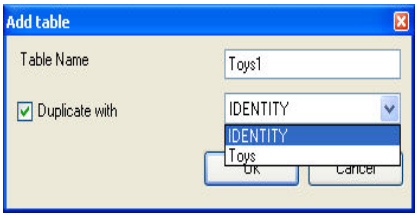
1. Click the **Table** drop-down list.
2. Select the required table.

## Add a table to the active database

1. Click **Add table**.
2. Enter the name of the new table.
3. Click **OK**.

You can also copy the structure of the table from a table that already exists in the selected database. To copy a table:

1. Select the **Duplicate with** check box.
2. Click on the drop-down list.
3. Select the required data.
4. Click **OK**.




## Delete a table in the active database

1. Click the **Table** drop-down list.
2. Click on the data required.
3. Click **Delete** table.

## View/hide active table's data

1. Click **View data**.

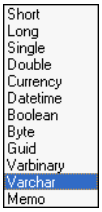
## Define a key field

1. Select the check box next to the required field.

2. Click **Apply**.

## Define a field's content type

1. Click the required field in the **Type** column.
2. Click the drop-down list button.
3. Select the required data.




4. Click **Apply**.

## Define a field's maximum size

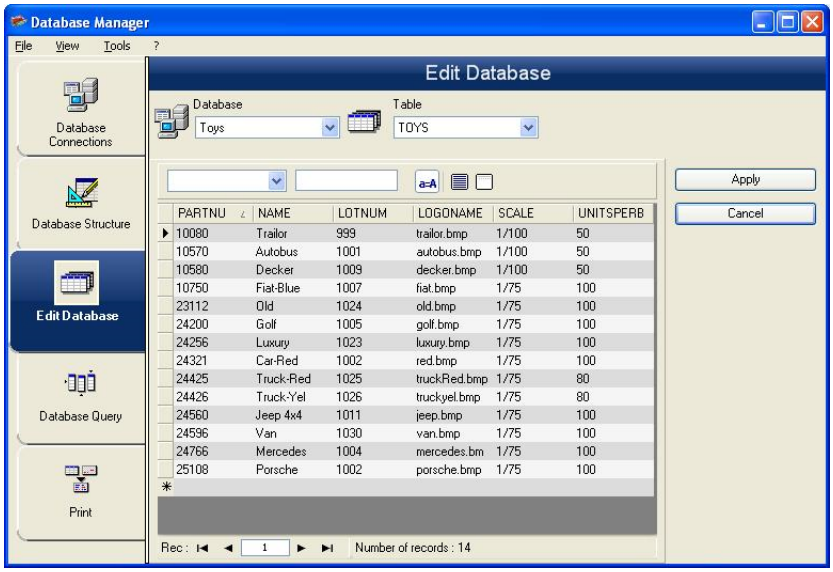
1. Click the required field in the **Length** column.
2. Enter the value required.

3. Click **Apply**.

## Allow an empty field

1. Select the **Allow Null** check box for the required field.
2. Click **Apply**.

# Edit Database window



The Edit Database window is used to manage the contents of the database file. In this window, you can add, modify, or delete data.

These actions depend on the type of database. Excel file records cannot be modified.

## Select records according to their content

Use the content of a field to find a record.

1. Click the table drop-down list button.
2. Click the data required.
3. Click the **Data Input** field.
4. Enter the value required in the **Data Input** field.

## Select all identical records

At least one record must have been found.

1. Click the drop-down list button.
2. Click the data required.
3. Click the **Data Input** field.
4. Enter the required data  in the **Data Input** field.
5. Click the **Select all** button ( ).

## Select an identical record

At least one record must have been found. There must be several identical records in the **Search** field.

To select a record, use the search tool:


## Create a new record

1. Click a field in the row marked with an asterisk.
2. Enter the values required in the corresponding fields.
3. Click **Apply**.

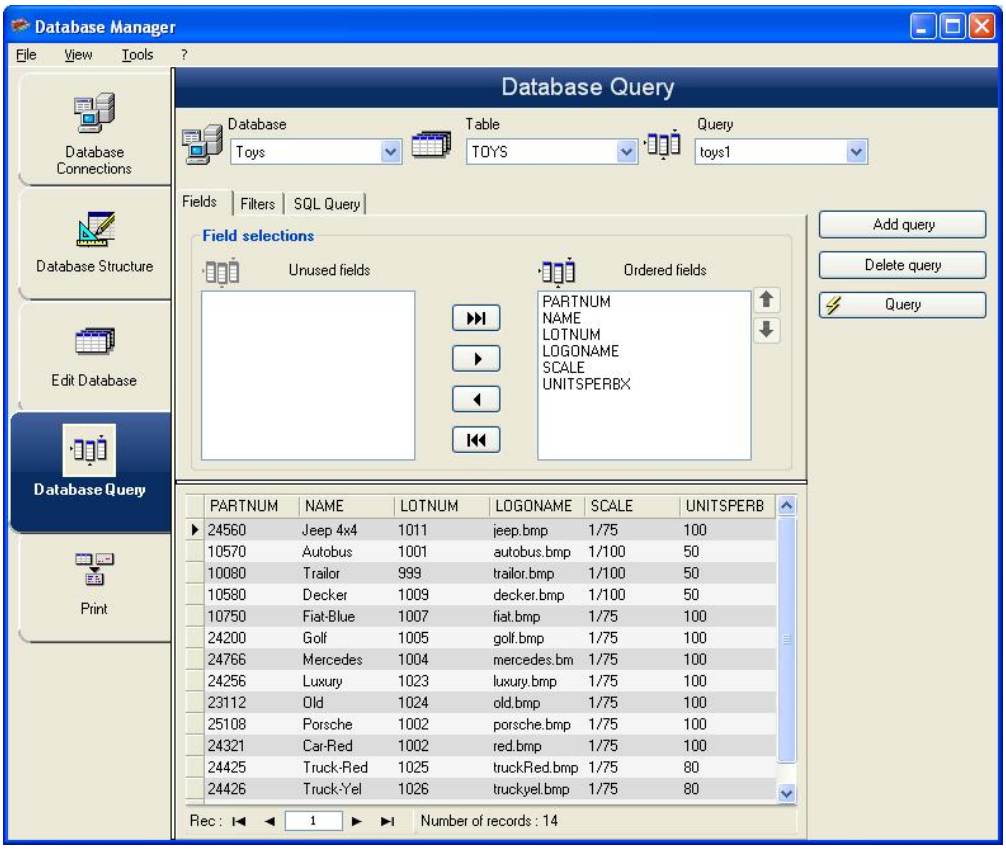
## Modify a record

1. Click the data you want to modifiy.
2. Enter the required data .
3. Click **Apply**.

## Delete a record

1. Click the database cursor for the required field.
2. Right-click the database cursor for the required field.
3. Click **Delete Record**.

# Database Query window




The Database Query window is used to create and apply various filters.

## Add a query

1. Click **Add query** on the **Fields** tab.
2. Enter a name for the query.
3. Click **OK**.

## Select/deselect one or more fields

1. Use to select the required (fields).
2. Click **Query** to refresh the database preview.

## Modify the order of fields selected

1. Click the required field in the **Ordered fields** window.
2. Click the up or down arrow to reorder the fields.
3. Click **Query** to refresh the database preview.

## Create a filter using predefined data

1. Select the **Filters** tab.
2. Click the **Add row** button ( ).
3. In the **Field** field, select in the drop-down list the data required.
4. In the **Operator** field, select the desired operator from the drop-down list..
5. In the **Value** field, enter the required value .
6. Click **Query** to view the result.

## Apply a logical operator to several filters

1. Click the **Add row** button ( ).
2. In the **Logical** field, select the desired logic (AND or OR) from the drop down list..
3. Create a filter (as described above).
4. Click **Query** to view the result.

## Remove a filter

**Note**: At least one filter must exist.

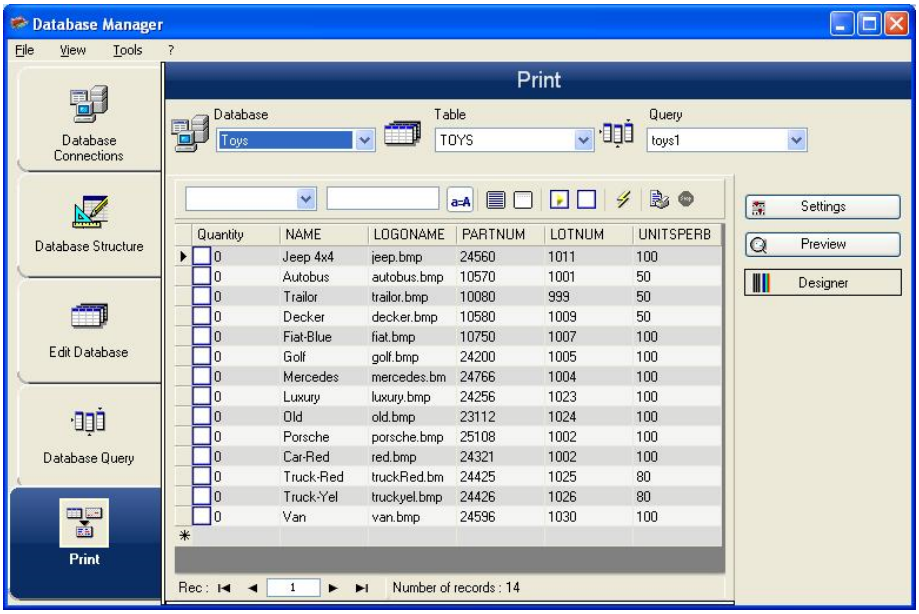
1. Click the database cursor for the required field.
2. Click the **Delete** row button ( ).

## Modify a filter in SQL

**Note**: At least one filter must exist.

1. Select the **SQL Query** tab.
2. Check **Modify the query in SQL language** to activate the SQL Query and make manual changes.
3. Click **Query** to view the result.

# The Print window




The Print window is used to select files for printing, assign printers, and define various parameters before printing is launched.

## Select a document to be printed

1. Select a document from a file

2. Select the **File** check box in the **Label** name group.
   - or -

1. Click the **Label Creation Wizard** button ( ).
2. Follow the on screen instructions.

**Note**: Creating a label in relation to the database allows you to define exactly which elements are required to position each database field.

## Select an existing label template

1. Click the **Open an existing document** button ( ).
2. Select a .lbl file.
3. Click **OK**.

**Note**: The Field radio buttons in the Label name and Printer name groups of options allow you to choose the required label or printer, when these options are defined in one of the fields of the active database.

## Select a document from a field

If your database contains the name of the label to be used for the printing job in one of the fields, you can define this field as the location where Database Manager is going to pick up the .lab .lbl file. For example, consider the following database:

| Ref | Designation | Qt | Code | Labname |
|---|---|---|---|---|
| 6574 | Ref1 | 1 | 9876546321 | Label1.lab |
| 6354 | Ref2 | 2 | 1236478855 | Label2.lab |
| 6987 | Ref3 | 3 | 6987456321 | Label1.lab |
| 3684 | Ref4 | 4 | 3698745632 | Label3.lab |

1. Check **Field** box in the **Label name** group.
2. Select the required field .

## Select a printer

1. Click on the **Add** or **Remove a printer** button ().
2. Select the required printer.

# Formulas

## The Formula data source

The Formula data source contains a list of data sources.

These data sources are populated by combinations of operators, constants, data sources, control variables, formulas, and functions. Data can be numeric or alphanumeric.

In order to carry out a calculation within a document, you must first create a **Formula** data source.

This data source has a specific dialog box allowing you to define the required function(s) for a given formula.

## About functions

Functions are predefined formulas that carry out calculations by using values called arguments, placed in a certain order, called syntax.

Functions are used to return a numeric, character string, or logical value, which is the result of a calculation or an operation.

There are six groups of functions in the formula definition:

- Check character calculation functions
- Conversion functions
    - ATA 2000 conversion functions
- Date and time functions
- Logical functions
- Mathematical functions
- Text/Character string functions

## Operators

Operators are mathematical symbols indicating an operation to be carried out. There are different types of operators: arithmetic, comparative, concatenation, and logical.

The program contains mathematical, comparative, concatenation, and logical operators.

## Arithmetic operators

| Operator | Used |
|---|---|
| * | Multiply two numbers. |
| + | Add two numbers together. |
| - | Subtract one number from another, or assign a negative value to an operand. |
| / | Divide one number by another. |
| ^ | Raise a number to the power of the exponent. |
| % | Modulo. |

## Comparative operators

| Operator | Meaning |
|---|---|
| < | Less than. |
| <= | Less than or equal to. |
| > | Greater than. |
| >= | Greater than or equal to. |
| = | Equal to. |
| <> | Different from. |

## Concatenation operator

Used to combine two strings.

| Operator | Meaning |
|---|---|
| & | Concatenation of two strings. |

## Logical operator

(See also Logical Functions)

| Operator | Meaning |
|---|---|
| ! | Not logical. |

## Check character calculation functions

**addmodulo10 («string»)**: Returns the integer with the Modulo 10 check character added to the end.

**addmodulo10_212(«string»)**: Returns a modulo 10_212 check character (a variant of the standard modulo 10 check character), and includes the current text string and the checksum at the end of the string.

**addmodulo43(«string»)**: Returns a modulo 43 check character and includes the current text string and the checksum at the end of the string.

**bimodulo 11 («string»)**: Returns the two modulo 11 check characters (Code 11).

**canadacustomscd («string»)**: Returns the Canada Customs Standard check character.

**Check103(«string»)**: Calculates Mod103 checksum.

**check 128 («string»)**: Returns the Code 128 and EAN-128 check character.

**Check PZN(«string»)**: Calculates check digit for PZN barcode. PZN barcode is based on Code 39 used for pharmaceutical products in Germany.

**CheckSum2Mod47(«string»)**: Calculates Mod47 checksum.

**CheckSumMod10(«string»)**: Calculates Mod10 checksum.

**CheckSumMod10_Codabar(«string»)**: Calculates specific Mod10 checksum for Codabar.

**CheckSumMod10_MSI(«string»)**: Calculates Mod10 checksum for MSI barcode.

**CheckSumMod11_3Suisse(«string»)**: Calculates Mod10 checksum for 3Suisses.

**CheckSumMod34(«string»)**: Calculates Mod34 checksum.

**checkupce («string»)**: Returns the check character for UPC-E (6 mandatory characters).

**CRC16(«string»)**: Returns CRC-16 value.

Examples:
CRC16("123456789") = 47933
"0x" & hex(CRC16("123456789"), 4)  = 0xBB3D

**modulo 10 («string»)**: Returns a modulo 10 check character of Code 2 of 5 interlaced, EAN-13, EAN-8, EAN-128 K-Mart, VPCA.

**modulo10_212(«string»)**: Returns a modulo 10 check character variant.

**modulo10IBM(«string»)**: Returns an IBM modulo 10 check character (a variant of the standard modulo 10 check character).

**modulo10UPS(«string»)**: Returns an UPS modulo 10 check character (a variant of the standard modulo 10 check character).

**modulo 11 («string»)**: Returns a modulo 11 (code 11) check character.

**modulo11IBM(«string»)**: Returns an IBM modulo 11 check character (a variant of the standard modulo 11 check character).

**modulo16( string ):** Returns the modulo 16 check digit

**modulo 24 («string»)**: Returns a modulo 24 (Code PSA) check character.

**modulo 32 («string»)**: Returns a modulo 32 (Italian pharmaceutical industry) check character.

**modulo 43 («string»)**: Returns a modulo 43 (code 39) check character.

**modulo 47 («string»)**: Returns two modulo 47 (Code 93) check characters.

**Plessey («string»)**: Gives two check characters (Plessey Code).

**pricecd («string»)**: Returns 4 UPC Random Price check characters.

**pricecd5 («string»)**: Returns 5 UPC Random Price check characters.

**StringToExt39(«string»)**: Prepares string to be used in Extended 39 barcode.

**UPSCheckDigit («string»)**: Returns the UPS check characters. The input for the method is a string. This method leaves the rest of the tracking number up to you. It takes a 15 character sequence, and calculates the check digit using this sequence.

1. The first two characters must be "1Z".
2. The next 6 characters we fill with our UPS account number "XXXXXX".
3. The next 2 characters denote the service type:
    - "01" for Next Day Air shipments.
    - "02" for Second Day Air shipments.
    - "03" for Ground shipments.
4. The next 5 characters is our invoice number (our invoices are 6 digits; we drop the first digit, e.g., the 123456 invoice would yield 23456 characters).

5. The next 2 digits is the package number, zero filled. E.g., package 1 is "01", 2 is "02".
6. The last and final character is the check digit.

**Note**: The described sequence above gives 17 characters, where as only need 15 are needed to calculate the check digit. To do this, drop the "1Z" portion, and only use the last 15 characters in the method.

## Conversion functions

**AI253 («string»)**: Specific function for preparing string for application identifier 253.

**AI8003 («string»)**: Specific function for preparing string for application identifier 8003.

**ASCII («string»)**: Returns the ASCII code of the first character in the string argument.

Example:
ascii("A") = 65

**char («integer»)**:

- For values from 128 to 255: Gives the character corresponding to the integer argument in the ASCII table.
- For all negative values and values from 0 to 127 and greater than 255: Returns the Unicode character corresponding to the *integer* argument.

Example:
char (65) = "A"

**CodabarData («data», «start», «stop»):** Adjusts data for CodabarData.

**Code128CData («string»):** Adjusts data for Code 128 C.

**Currencytoeuro («value»)**: Converts the current national currency into Euros.

**DatabarData («data», «min», «max», «hasComposite»)**: Adjusts data for DatabarData.

**DBCSToUnicode («string»,«codepage»):** The formula takes two parameters - the data «*string*» to be converted, and the Codepage «*codepage*» used. The Codepage parameter is represented by the language name (Thai, Japanese, ChineseGBK, Korean, ChineseBig5, European, Eastern, Cyrillic, Greek, Turkish, Hebrew, Arabic, Baltic, Vietnamese, UTF-8, ACP) or numeric value (search code page identifiers documentation on Internet).

Example:

DBCSToUnicode(unicodetoDBCS("傍傍傍傍", "UTF-8"), "UTF-8") =傍傍傍傍

This function is needed for data that come from non-Unicode file or data sources (e.g. Database).

**Note**: This function has reverse action to UnicodeToDBCS method.

**dollar («value»)**: Modifies a field to represent a price.

Example:

If a field named PRICE displays a value of 199, then:

dollar(PRICE) will display $199.00.

**Ean128Data («string_1»,«string_2»)**: Adjust "string_1" data for EAN 128 barcode according to template "string_2"

**Eurotocurrency («value»)**: Converts the Euro into the national currency.

**Note**: The number of decimals to be displayed must be defined in the **Output** tab of the **Formula** data source by checking the Display decimals field. The conversion rate used is the one defined in the **Others** tab of the Options dialog box.

**FileToData(«fileName», «errorData», «maxSize»)**: Returns the data read from the file.

**fixed («value» , «num_decimals», «non_sep»)**: Returns a string of formatted characters with a number of decimals equal to *num_decimals* and with or without a thousands separator.

Examples:

fixed (1234.5678, 3, TRUE) = "1234.568"

fixed (1234.5678, 3, FALSE) = "1,234.568"

**Note**: The thousands separator to be displayed must be defined in the **Output** tab of the Formula data source by checking the **Display decimals** box.

**FormatDate(«date», «dateFormat», «localeID»)**: Converts **«date»** to a string formatted according to «dateFormat».

| | | |
|---|---|---|
| Return value | String | A date value in string format, formatted according to a format. |
| date | String | Date value in string format. |
| dateFormat | String / Number | Format of the base date value.<br>- Numbers encode predefined formats. (For example, 1"dd/mm", 2 "dd |

| | | |
|---|---|---|
| | | mmmm" and so on). Possible values are available here.<br>- 0 or negative values mean using the current system date format.<br>- String value encode the format directly. (For example, "yyyy/mm/dd") |
| LocaleID | Number | Locale used to show the date value. **LocaleID i**s optional (Default: 0, system locale will be used). Possible values (number) can be found on http://msdn.microsoft.com/en-us/goglobal/bb964664.aspx. |

**FormatMoney(«amount», «use separation characters(T/F)», «price characters», «force leading zero (T/F)», «location for price character», «# of decimal places»)**

Allows you to add currency symbol, dots, force leading zero, use separation characters

Example:

FormatMoney("123456.234", "T","$","F",0,2) = $123,456.23

FormatMoney("1234","F","$","T",8,2) = $1234.00

FormatMoney("0.234","F","$","F",0,2) = $.23

FormatMoney("0.234","F","$","T",0,2) = $0.23

**Note**: **Location for price character** is responsible for price character position, spaces are added between price character and amount . This value is ignored if it is less than result string length.

**FormatNumber(number)**: Allows you to format a numeric field where a pound sign (#) means only display if a value is there and zero (0) means always display.

Examples:

FormatNumber(123.45, "US$ #,###,###.00") = US$ 123.45

FormatNumber(123.45, "US$ 0,000,000.00") = US$ 0,000,123.45

FormatNumber(.45, "#,##0.00") = 0.45

FormatNumber(.45, "#,###.00") = 45

FormatNumber(7188302335, "(###) ###-####") = (718) 830-2335

FormatNumber(123.45, "00.00") = 23.45

FormatNumber(123.567, "###,##0.00") = 123.57

**GetEnv(«name»)**: Returns the value «name» (string) of environment variable.

Example:

GetEnv("OS") = " Windows_NT".

**Note**: This function works with user defined environment variables

**GS1AIData(«AIName», «AIData», «isLastAI», «calcCheckDigit»):** Adjusts data for GS1 AI.

**GS1HRData(«AIName», «AIData», «calcCheckDigit»):** Adjusts data for GS1 Human readable.

**GS1Norm(«string»):** Adjusts data for GS1 Norm.

**int («*value*»)**: Returns the largest integer less than or equal to the *value* argument.

Examples:

int (-5.863) = -6

int (5.863) = 5

**LmChar («integer») :** Returns the character corresponding to the integer argument in the ANSI table. This table does not depend on the locale system.
Value is between 0 and 255.

**MaxiCodeData(«Mode»,«PostalCode»,«CountryCode»,«ClassOfService»,«TrackingNumber»,«UpsShipperNumber»,«JulianDate»,«ShipmentID»,«PackageNumber»,«TotalPackages»,«PackageWeight»,«AdressValidation»,«ShipToAdress»,«ShipToCity»,«ShipToState»)**: Creates a maxicode data from input «string».

**text («*value*», «*format*»)**: Returns the *value* argument written in the format imposed by the *format* argument, which corresponds to the format described in the explanation of the When Printed variable.

Examples:

text (012345678,"###-##-####") = 012-24-5678

text (2125551212,"(###)###-####") = (212)555-1212

Example:

text (12 , "~~~~") = 0012

where "~" is replaced by 0 in result value if the input value length is less then format length.

Example:

text (12 , "****") or text (12 , "####") = 12

where "*" and "#" are the special symbols replaced by data.

**trunc («*value*»)**: Returns the integer part of the *value* argument.

Example:

trunc (123.45) = 123

**UnicodetoDBCS(«*string_1*», «*string_2*»)** : The formula takes two parameters - the data «*string_1*» to be converted, and the Codepage «*string_2*» used. The Codepage parameter can have any of the following as a value:

- Thai
- Japanese
- Chinese GBK
- Korean
- Chinese Big5
- European eastern
- Greek
- Turkish
- Hebrew
- Arabic
- Baltic
- Vietnamese

**Note**: The Codepage parameter is not case sensitive.

**value («*string*»)**: Gives the numeric value of a character string.

Example:
value("123") = 123
value("0.00:48:00")-value("12:00:00") equals "16:48:00"-"12:00:00" equals 0.00, the serial number for 4 hours and 48 minutes.

**Note**: It is not usually necessary to use the value function in a formula, since the application automatically converts the text to numbers if necessary.

**ValueEx(«string»)**: Converts «string» into a numerical value.

**VoiceCode(«string», «string», «string»)**: A VoiceCode is a 4 digit number computed using the GTIN, Lot, and optional Date from a PTI.

| | | |
|---|---|---|
| Return value | string | Calculates a VoiceCode value. |
| GTIN | string | 14 digit GTIN number. Only digits are allowed. Error will be returned in case of non-digit value. Left part 14 digits are accepted in case data length is more then 14. '0' are added to the left in case data length is less than 14 digits. |
| LotNumber | string | Up to 20 alphanumeric symbols data. Left part 20 symbols are accepted in case data length is more than 20 digits. |
| Date | string | This is an optional parameter. 6 digits data that represents the date in YYMMDD format. Formula returns error in case of wrong length, non-digit value or wrong date. |

This computation is performed as follows:

1. Compute PlainText.

    a. PlainText is the 14 digit GTIN appended by the Lot Code and the Date (where present).

    b. Do not include the application identifier prefixes or parentheses.

    c. There are no spaces between the GTIN, Lot and Date fields.

    d. Date if present is represented as YYMMDD with zero packing and no '/' characters.

2. Compute ANSI CRC-16 Hash of the PlainText ASCII bytes using the standard ANSI CRC-16 hash with the polynomial of X16 + X15 + X2 + 1

    See CRC-16 function.

3. Compute the VoiceCode by from the Hash by taking the 4 least significant digits in decimal form (Hash mod 10000).

Example:
GTIN = (01) 10850510002011
Lot = (10) 46587443HG234
PlainText = 10850510002011146587443HG234
CRC-16 Hash = 26359
VoiceCode = 6359
Large Digits = 59
Small Digits = 63

Examples:
VoiceCode("10850510002011", "46587443HG234") = 6359
VoiceCode("65457886676767", "2", "100126") = 5836

# ATA 2000 Conversion functions

**Bin2Hex(«value», «pad»)**

Converts a binary value to a hexadecimal output value. The ‘pad’ parameter is used to define the length of the whole output by adding 0 characters to the left of the result.

Example:

Bin2Hex("0010100111101001101",0)  returns 14F4D

Bin2Hex("0010100111101001101",0)  returns 00014F4D

**CRC16_CCITT(«string»)**

Calculates CRC16 CCITT checksum.

This is used in ATA2000 regulation for TOC for CRC calculation.

Example:

CRC16_CCITT ("A") returns B915

CRC16_CCITT ("AB") returns 4B74

CRC16_CCITT ("ABC") returns F508

**CRC32_CCITT(«string»)**

Calculates CRC32 CCITT checksum.

This is used in ATA2000 regulation for TOC for CRC calculation.

Example:

CRC32_CCITT ("A") returns D3D99E8B

CRC32_CCITT ("AB") returns 30694C07

CRC32_CCITT ("123456789") returns CBF43926

**Dec2Bin(«value», «pad»)**

Converts a decimal value to a binary output value. The ‘pad’ parameter is used to define the length of the whole output by adding 0 characters to the left of the result.

Example:

Dec2Bin("5", 0) returns 101

Dec2Bin("5", 4) returns 0101

**Dec2Hex(«value», «pad»)**

Converts a decimal value to a hexadecimal output value. The ‘pad’ parameter is used to define the length of the whole output by adding 0 characters to the left of the result.

Example:

Dec2Hex(510, 0) returns 1FE

Dec2Hex(510, 4) returns 01FE

### String2Hex(«value», «pad»)

Converts a string value to a hexadecimal output value. The ‘pad’ parameter is used to define the alignment in Bytes for the output by adding 0 characters to the right of the result.
In RFTAG, the data is very often encoded in hexadecimal with a word alignment (2Bytes).

Example:

String2Hex("A", 0) returns 41 ( **A** is AINSI code **65** in decimal which is **41** in hexadecimal)

String2Hex("ABC", 2) returns 414243

String2Hex("ABC", 4) returns 41424300

Here is a sample for a RFID ATA2000 Birth record:

String2Hex("MFR S0671*SEQ M37GXB92*PNO PQ7VZ4*PDT CONTROLLER*ICC 456789*UIC 2*DMF 20160701*UNT KG*HAZ UN0003*ESD 1*LOT 123456*CNT FR*PMLPML123456789", 2)
returns
4D46522053303637312A534551204D333747584239322A504E4F20505137565A342A50445420434F4E54524F4C4C45522A49

### String6BitsEncoding(«value», «pad»)

Converts a string value to a 6-Bit ASCII encoding output value.
The ‘pad’ parameter is used to define the alignment in Bytes for the output by adding 0 characters to the right of the result.
In ATA2000 data can be encoded in 6bits (instead of 8bits) for compression purposes.

Example:

String6BitsEncoding("11", 0) returns C710

### VDAString6BitsEncoding(«string»)

Converts a string value to a 6-Bit ASCII encoding output value using the German Association of the Automotive Industry (VDA) requirements.
In ATA2000 data can be encoded in 6bits (instead of 8bits) for compression purposes.
The table used for conversion is available here. For more details refer to specifications VDA5500 and VDA4994.

Example:

VDAString6BitsEncoding("11") returns C71860820820

# Date and time functions

The date variable represents the system date and time, and not the date variable defined by the program:

**Note**: 30/12/1899 = your application's reference date. If you want the formula to return the current date, the solution is to calculate the number of days elapsed since the reference date.

**BestBefore («*date*» , «dateFormat» , «offset» , «offsetUnit» , «changeMonth» , «outputFormat», «localeID»)**

Allows you to calculate a Best Before date based on keyboard input value (not based on current date).

| | | |
|---|---|---|
| Return value | String | A date value in string format, calculated by a base date and an offset. |
| date | String | Base date value in string format. |
| dateFormat | String / Number | Format of the base date value.<br>- Numbers encode predefined formats. (For example, 1"dd/mm", 2 "dd mmmm" and so on). Possible values are available here.<br>- 0 or negative values mean using the current system date format.<br>- String value encode the format directly. (For example, "yyyy/mm/dd") |
| offset | Number | Number of date units to be added to the base date. |
| offsetUnit | String / Number | Meaning of the offset parameter.<br>- 1 or "d" or "D" means days.<br>- 2 or "m" or "M" means months.<br>- 3 or "y" or "Y" means years. |
| changeMonth | Number | Optional (Default: True)<br>- 0 means False.<br>- 1 means True.<br>If calculated date does not exist in the month (For example, 30 of February).<br>True return the first day in the next month (For example, 1st March).<br>False return the last day in the current month (For example, 28 of February). |
| outputFormat | String / Number | Optional (Default: same as dateFormat)<br>This parameter specifies the output format of the calculated date.<br>Possible values are available here. |
| localeID | Number | Optional (Default: 0, system locale will be used)<br>This parameter specifies the locale used to show the date value. |

Example:

BestBefore("14/06/2012" ,"dd/mm/yyyy","18","m", 1, "mmmm dd, yyyy", 1033) will return December 14, 2013.

BestBefore("14/06/2012" ,"dd/mm/yyyy","18","m", 1, "mmmm dd, yyyy", 1036) will return Décembre 14, 2013.

Example: Formula with a date data source.
Create a date variable name date0 with the date of the day, for example 26/06/2012 (dd/mm/yy format).
BestBefore(date0 ,"dd/mm/yy","18","m", 1, "dd/mm/yyyy") will return 14/12/2013.

**CFIA_Month(«date»):**

Returns month name used by Canadian Food Inspection Agency (CFIA): JA, FE, MR, AL, MA, JN, JL, AU, SE, OC, NO, DE.

Example:
CFIA_Month ("03/01/2021") returns MR
CFIA_Month (today()) returns month name based on today's date, if today is 01/01/2021 - the return value is JA
CFIA_Month ("06/01") returns JN

**Note**: «date» argument format depends on the environment settings applied for the system.

**DateOffset( «date» , «offset», «offsetUnit» , «changeMonth» )**

| Return value | Date | Adds a date interval to the specified base date. |
|---|---|---|
| date | Date | Base date value. |
| offset | Number | Number of date units to be added to the base date. |
| offsetUnit | String / Number | Meaning of the offset parameter.<br>- 1 or "d" or "D" means days.<br>- 2 or "m" or "M" means months.<br>- 3 or "y" or "Y" means years. |
| changeMonth | Number | Optional (Default: True)<br>- 0 means False.<br>- 1 means True.<br>If calculated date does not exist in the month (for example, 30 of February).<br>True return the first day in the next month (for example, 1st March).<br>False return the last day in the current month (for example, 28 of February). |

Example:
DateOffset("26/06/2012",1,"d",1) will return 27/06/2012
-OR-
DateOffset(today(),1,"D", 0)

Example: Formula with a date data source.
Create a date variable name Date0 with the date of the day, for example 26/06/2012. You should ensure that date variable is in default system format (otherwise you can use DateValue() function to get the date value from the date variable).

DateOffset(Date0,1,"D", 0) will return 27/06/2012

**DateValue( «formattedDate », «dateformat» )**

| | | |
|---|---|---|
| Return value | Date | Returns a date value of the formattedDate parameter. |
| formattedDate | String | A text format date, to be converted to a date value. |
| dateformat | String | Optional (Default: the system date format)<br>If specified, the direct format string. For example, "dd/mm/yy" |

Example:
DateValue(22062012,"ddmmyyyy") will return 22/06/2012.

**day («*date*»)**: Gives the day of the month of the *date* argument.

**FiscalDate(«fiscalStartDate», «outputFormat»)**

Allows you to calculate the date for the start of your fiscal year (yyyy.mm.dd).

Examples (if current date is 2009.11.24):

FiscalDate("2009.01.01", 1) = 09
FiscalDate("2009.01.01", "yyyy") = 2009
FiscalDate("2009.01.01", 3) = 47
FiscalDate("2009.01.01", "day") = 328

**hour («*date*»)**: Gives the hour of the *date* argument.

**minute («*date*»)**: Gives the minutes of the *date* argument.

**month («*date*»)**: Gives the month of the *date* argument.

**now ()**: Gives the current date and time.

**second («*date*»)**: Gives the second of the *date* argument.

**shiftcode («items», «defaultValue»)**

The Shift code is used as a data source for a field on the label that needs to change based upon the current time. A shift represents a span of time that is defined and named.

| | | |
|---|---|---|
| Return value | String | Returns the current shift code value. |
| items | String | Shift code items in the following format: startHour:startMin-stopHour:StopMin-value\|...\| startHour:startMin-stopHour:StopMin-value |

For example, shifts could be defined as follows:

- 7:00 to 15:00 = Day
- 15:00 to 23:00 = Evening
- 23:00 to 7:00 = Night

**Notes:**
1. The 24-hour clock is used to define time values. 0:00 is midnight; 12:00 is midday.
2. If there are any time overlaps in input then the first acceptable value is returned.
3. If there is no shift for the current time then an empty string is returned.

Example (if current time is 16:00):
shiftcode("7:00-15:00-Day|15:00-23:00-Evening|23:00-7:00-Night","Day") ="Evening"

**SpecificDateFormat ("[date format]", "+/[offset data][date interval]")**

Allows you to customize a date stamp by offsetting the date by a certain interval and use this customized date in a formula expression. The SpecificDateFormat function offsets the current date based on the system clock using either the default date format or a different format of your choosing.

- [date format] Specifies the date format to use. This parameter must be enclosed within parentheses. ***View table*** of valid date formats.

| Date Setting | Output | Description |
|---|---|---|
| D and d | 1 | date of month (non lead zero) |
| DD | 01 | 2 char date of month (lead zero) |
| dd | 1 | 2 char date of month (lead space) |
| DDD | 224 | days from begin of year (lead zero) |
| ddd | 224 | days from begin of year (lead space) |
| DDDDD and ddddd | 33482 | days from 1/1/1900 (5 to 9 chars available) |
| W and w | 1 | day of week (Sun=1, Sat=7) |
| WW | 34 | weeks from begin of year (lead zero) |

| | | |
|---|---|---|
| ww | 34 | weeks from begin of year (lead space) |
| WWW and www | 783 | weeks from 1/1/1900 (last 3 digits) |
| WWWW and wwww | 4783 | weeks from 1/1/1900 (4 to 9 chars available) |
| WWWWWWWWW | 000004783 | weeks from 1/1/1900 (lead zero) |
| wwwwwwwww | 4783 | weeks from 1/1/1900 (lead space) |
| M and m | 9 | month digit (non lead zero) |
| MM | 09 | 2 char month digit (lead zero) |
| mm | 9 | 2 char month digit (lead space) |
| MMM | 009 | 3 char month digit (lead zero) |
| mmm | 9 | 3 char month digit (lead space) |
| MMMM and mmmm | 1101 | months since 1/1/1900 |
| MMMMM | 01101 | 5 char months from 1900 (lead zero) |
| mmmmm | 1101 | 5 char months from 1900 (lead space) (5 to 9 chars available) |
| MMMMMMMMM | 000001101 | 9 char months from 1900 (lead zero) |
| mmmmmmmmm | 1101 | 9 char months from 1900 (lead space) |
| YY and yy | 91 | 2 digit year |
| Y and y | 1991 | full year |
| YYY and yyy | 991 | last 3 digits of year (3 to 9 chars available) |
| YYYYYYYYY | 000001991 | full year (leading zero) |
| yyyyyyyyy | 1991 | full year (leading space) |

- +/[offset data] This value specifies the amount by which to offset the date. The interval must be preceded by either a plus (+) or minus () sign, depending on if you want to add or subtract from the current date.
- [date interval] The date interval must be either d for days, w for weeks, m for months, or y for years.

Examples of the SpecificDateFormat function used in expressions in different formats:

**Note**: For these examples, assume the current date is July 29, 2013.

SpecificDateFormat( "mm dd yy" , "+1y" ) = 7 29 14 (1 year after the current date)
SpecificDateFormat( "ww/m/yyyy" , "30w) = 1/12/2012 (30 weeks before the current date)
SpecificDateFormat( "ddddd / wwww /mmmm" , "" ) = 41484/5928/1363 (days, weeks and months since 1900 )

**TimeOffset(FormatDate(Now(), "mm/dd/yyyy hh:nn:ss"),"time interval +/offset time")**The TimeOffset function allows you to customize a time stamp by offsetting the time by a certain interval and use this customized time in a formula expression. The TimeOffset function offsets the current time based on the system clock using the default time format.

When used in an expression, the TimeOffset function must be written using the following parameters:

- You can use only the default format of "mm/dd/yyyy hh:nn:ss". To enter a different format you need to wrap TimeOffset with the DateValue function and then wrap into FormatDate function where you can specify other time format.
- “+/offset time” This value specifies the amount by which to offset the time. The offset must be preceded by either a plus (+) or minus () sign, depending on if you want to add or subtract from the current time.
- "time interval" The time interval must be either s for seconds, m for minutes, or h for hours.

The following table shows examples of the TimeOffset function used in expressions in different formats. (Note: For these examples, assume the current time is 4:12:10 PM on July 30, 2013)

TimeOffset(FormatDate(Now(), "mm/dd/yyyy hh:nn:ss"),"h+5") = 07/30/2013 21:12:10 (5 hours after the current time)
FormatDate(DateValue(TimeOffset(FormatDate(Now(), "mm/dd/yyyy hh:nn:ss"), "h+10"), "mm/dd/yyyy hh:nn:ss"), "hhnn") = 16-22 (10 minutes after the current time)

**today ()**: Gives the current date.

**Week (*«date»*)**: Returns the number of the week from the *date* argument.

**weekday (*«date»*)**: Returns the day of the week from the *date* argument.

**Note**: Sunday is considered the first day of the week.

Example:
On "Tuesday November 20, 1999" the weekday(now()) = 3

**WeekISO8601(«Date», «DateFormat»)**

Allows you to create a formula that support the ISO8601. ISO 8601 is an international standard covering the exchange of date and time-related data.

| | | |
|---|---|---|
| Return value | date | Returns the week of the specified date |
| Date | string | A text format date, to be converted to a date value. |
| DateFormat | string | Format of the base date value.<br>- Numbers encode predefined formats.<br>(e.g. 1 dd/mm/yy, 2 dd/mm/yyyy and so on<br>- 0 or negative values means using the current system date format<br>- String value encode the format directly. (e.g. "yyyy/mm/dd") |

Example:

WeekISO8601(" 03/01/2010 ",0) = 53

WeekISO8601(" 02/01/2011 "," dd/mm/yyyy ") = 52

WeekISO8601(" 01/01/2011 ", " dd/mm/yyyy ") = 52

**Year («*date*»)**: Gives the year of the date argument.

Example:

Minute (now()) gives the current hour and minutes.

Year (today()) gives the year of the current date.

**Note**: For Minute and Hour, only the now() argument is allowed. For all other functions, now() and today() are allowed.

## Logical functions

Logical functions allow you to check whether one or more conditions have been fulfilled.

**Note**: TRUE equals 1 and FALSE equals 0.

**and («*expr_1*», «*expr_2*»)**: Returns TRUE if both arguments are true, FALSE if at least one is false. The arguments must be calculated from logical values.

Example:

and(exact("string","string"),exact("string","string")) = 0

and(exact("string","string"),exact("string","string")) = 1

**exact («*string_1*», «*string_2*»)**: Returns TRUE if the two strings are identical, FALSE if not. This function is case-sensitive.

Example:

exact("software","software") = 1

exact("software","software") = 0

**if («*expr*», «*Val_if_true*», «*Val_if_false*»)**: Returns the *Val_if_true* value if *Expr* is true and the *Val_if_false* argument if *Expr* is false.

Example:

if(exact("string", "string"), "true", "false") = false

if(exact("string", "string"), "true", "false") = true

**not («*logical*»)**: Gives the opposite of the *logical* argument.

Example:

not(exact("string", "string")) = 1

not(exact("string", "string")) = 0

not(False) = 1 or not(0) = 1

not(True) = 0 or not(1) = 0

not(1+1=2) = 0

**or («*expr_1*», «*expr_2*»)**: Returns TRUE if one of the two arguments is true and FALSE if both arguments are false. The arguments must be calculated from logical values.

Example:

or(exact("string", "string"),exact("string", "string")) = 0

or(exact("string", "string"),exact("string", "string")) = 1

or(true,true) = 1 or or(1,1) = 1

or(true,false) = 1 or or(1,0) = 1

or(false,false)= 0 or or(0,0) = 0

## Mathematical functions

**Abs(data)**: Returns the absolute (positive) value of data.

Examples:

Abs(-5) = 5

Abs(5) = 5

**base10tobaseX(«*string_1*»,«*string_2*»)**: Converts *string_2* from base 10 to base *string_1.*

Examples:

If the field named Base 16 contains the string "0123456789ABCDEF"

BASE10TOBASEX(Base16, 12) produces C

BASE10TOBASEX(Base16,10) produces A

BASE10TOBASEX("012345","9") produces 13

**Note**: This formula cannot accept negative decimal numbers for *string_2* parameter.

**baseXtobase10(«*string_1*»,«*string_2*»)**: Converts *string_2* from base *string_1* to base 10.

Examples:

If the field named Base 16 contains the string "0123456789ABCDEF"

BASEXTOBASE10(Base16, "E") produces 14

BASEXTOBASE10(Base16,10) produces A

BASEXTOBASE10("012345","9") produces 13

**Ceil(data)**: Rounds data up to the next whole number.

Example:

Ceil(3.234) = 4

Ceil(7.328) = 8

**Decimals(data1, data2)**: Uses data2 decimal places in data1.

Example:

Decimals(4, 2) = 4.00

Decimals(3.524, 1) = 3.5

**eval_add(«string»,«string»)**: Returns the sum of parameters.

Example:

eval_add(5,5)=10

**eval_div(«string»,«string»)**:Returns the division of parameters.

Example:

eval_div(20,2)=10

**eval_mult(«string»,«string»)**: Returns the multiplication of parameters.

Example:

eval_mult(5,2)=10

**eval_sub(«string»,«string»)**: Returns the substraction of parameters.

Example:

eval_sub(20,10)=10

**Floor(data)**: This function rounds data down to the next whole number.

Example:

Floor(3.234)= 3

Floor(7.328)= 7

**hex(«val_1»,«val_2»):** Converts the **val_1** decimal number to hexadecimal format with a total value of **val_2.**

**Note**: This formula cannot accept negative decimal numbers for *val_1* parameter.

Example:

hex(2, 8) = 00000002

**int («value»)**: Returns the largest integer less than or equal to the value argument.

Examples:

int (-5.863) = -6

int (5.863) = 5

**max(data1, data2)**: Displays the largest value in the data series.

Example:

Max(5, 12) = 12

**min(data1, data2)**: Displays the lowest value in the data series.

Example:

Min(5, 12) = 5

**mod («val_1», «val_2»)**: Returns the remainder of the division of the val_1 argument by the val_2 argument. The result has the same sign as the divisor.

Examples:

mod (7,2) = 1

mod (-7,2) = -1

mod (7,-2) = 1

mod (-7,-2) = -1

**quotient («val_1», «val_2»)**: Returns the integer result of the division of the val_1 argument by the val_2 argument.

Example:

quotient(10, 2) = 5

**round («val_1», «val_2»)**: Returns the argument val_1 rounded to the number of figures indicated by val_2.

- If *val_2* is greater than 0, *val_1* is rounded to the number of decimals indicated.
- If *val_2* is equal to 0, *val_1* is rounded to the closest integer.
- If *val_2* is less than 0, *val_1* is rounded off to the left of the decimal point.

Examples:
round (4.25,1) = 4.3
round (1.449, 1) = 1.4
round (42.6,-1) = 40

## Text functions

A character string can be assimilated into a table if each fields contains a character. It is defined by its length (total number of characters in the string, including spaces). The position of a character in the string corresponds to its place in the table. For example, the first character is at position one, position 3 corresponds to the third character in the string.

**cyclebasex ( )**: Allows counting to take place in any kind of database counting system. The numbering system must be defined within the linked expression. The start value, the value of each increment and the number of copies must also be specified for each number. All these values can be linked to other fields in the label, but the field names must not be enclosed in quotation marks.

Example:
If a field named Base 16 contains the character string 0123456789ABCDEF, then:
cyclebasex(base16, "8", 1 ,1) = 8,9,A,B,C
cyclebasex(base16, "F", -1,1) = F,E,D,C,B,A 9,8,7
cyclebasex(base16, "B0 ", 1,1) = B0, B1, B2
cyclebasex("012345", "4",1,2) = 4,4,5,5,10,10,11,11

**cyclechar ( )**: Creates a user-defined set of characters for a complete cycle.

Examples:
cyclechar("A", "C") = A B C A B C A B C
cyclechar("A", "C", 1,2) = A A B B C C A A B B

**cyclenumber ( )**: Allows you to set your own sequence of numbers, instead of using the normal sequence of numbers or letters (0,1,2; or A,B,C).

Examples:

cyclenumber(1,3) will produce labels in the following sequence: 1 2 3 1 2 3 1 2 3...

cyclenumber(1,3,1,2) will produce labels in the following sequence: 1,1,2,2,3,3,1,1,2,2,3,3,1,1…

**cyclestring ( )**: Allows you to create a group of words or characters using a complete cycle as an increment field. The whole string must be enclosed in quotation marks (" ") and each word or group of characters must be separated from the others by a semicolon ( ; ).

Example:

cyclestring("Mon ; Tue ; Wed ; Thu ; Fri ; Sat ; Sun") = Mon Tue Wed Thu Fri Sat Sun

The following example is for labels that use all letters of the alphabet except for O and I.

cyclestring("A;B;C;D;E;F;G;H;J;K;L;M;N;P;Q;R;ST;U;V;W;X;Y;Z")

**exact («*string_1*», «*string_2*»)**: Returns TRUE if the two strings are identical, FALSE if not.

Examples:

exact("software","software") = 1

exact("sftware","software") = 0

**extract («string», «sep», «pos»)**: Returns the substring from the character string «string» at the specified position «pos» that contains data separated by string «sep».

Example:

Extract("1;2;3;4", ";", 3) = 3

**find («string», «*key*», «*start*»)**: Returns the position of the first occurrence of the *key* argument in the *string* argument. The search in the *string* argument starts from the position returned by the *start* argument (*start* >= 1). The function resets to zero if no occurrence of the *key* argument is found. The function distinguishes between upper and lower case letters.

Example:

find("Peter McPeepert","P",1) = 1

find("Peter McPeepert","p",1) = 12

**left («*string*», «*num_char*»)**: Returns the character string extracted from the *string* argument. This string starts at position one of the *string* argument and has a length equal to the *num_char* argument.

Example:

left("Peter McPeepert",1) = P

left("Peter McPeepert ",5) = Peter

**len («*string*»)**: Gives the length of the *string* argument. Spaces are counted as characters.

Example:

len("Paris, New York") = 15

len("") = 0

len(" ") = 1

**lower («*string*»)**: Converts all upper case letters in a text string into lower case letters.

Example:

lower("Paris, New York") = paris, new york

**LTrim(«*string*»)**: Will automatically trim off any trailing spaces or leading spaces to the **left** data.

Example:

LTrim(" No."): No

**mid («*string*», «*start*», «*num_char*»)**: Returns the character string extracted from the *string* argument. This string starts at the position corresponding to the value of the *start* argument (*start* >=1) and has a length equal to the *num_char* argument.

Example:

mid("Paris, New York",8,8) = New York

**Output («variable»)**: Returns the formatted value of the variable.

Example:

Counter0 has the prefix "number:". The formula "counter0 has a value of " & counter0 & " and a formatted value of " & output(Counter0) gives the result : counter0 has a value of 1 and a formatted value of number:1

**Note**: The variable formatting is defined in the Output tab.

**pad ( «*string*», «*length*», «*char*»)***:* Adds characters to the left of the field to assign a predefined length to the whole input. Any character can be selected as a padding character.

Example:

If a field named GREETING displays a value HELLO, then:

pad(GREETING,8,0) = 000HELLO

pad(5,3,0) = 005

pad("Nine",6,"a") = aaNine

**replace («*string*», «*start*», «*num_char*», «*new_string*»)**: Returns the converted *string* argument. A number (equal to the *num_char* argument) of characters from the position defined in the *start* argument has been replaced by the *new_string* argument.

Example:

replace("Paris, New York",8,8,"Singapore") = Paris, Singapore

**replaceString («string», «old_string», «new_string»)**: Replaces all occurrences of a specified «old_string» in character string «string» with another specified «new_string».

Example:

ReplaceString( "abc12def12", "12", "") = abcdef

**rept («*string*», «*num_char*»)**: Returns the character string where the *string* argument is repeated the number of times in the *num_char* argument.

Example:

rept("Ah Paris! ",2) = Ah Paris! Ah Paris!

**right («*string*», «*num_char*»)**: Gives the character string composed of the last characters of the *string* and has a length equal to the *num_char* argument.

Example:

right("Purchase order",5) = order

**RTrim («*string*»)**: Will automatically trim off any trailing spaces or leading spaces to the **right** data.

Example**:**

RTrim("Part ") :Part

**search («*string*», «*key, start*»)**: Gives the position of the first occurrence of the *key* argument in the *string* argument. The search starts from the position defined by the *start* argument (*start* >= 1). The function resets to zero if no occurrence of the *key* argument is found.

Examples:

search("Purchase order","order,1) = 10

search("Purchase order","c",1) = 4

**StrAfter*(«data»,«start after», «length»)***: This function results in a string that is exactly length characters long after a specified start after character.

Examples:

StrAfter("1234-5678", '-', 3)= Takes the 3 characters after the dash (567)

StrAfter("1234-5678", '-')= Takes all characters after the dash (5678)

**StrBefore*(«data»,«start before», «length»)***: This function results in a string that is exactly length characters long before a specified start before character.

Examples:

StrBefore("1234-5678", '-', 2)= Takes the 2 characters immediately before the dash (34)

StrBefore("1234-5678", '-')= Takes all characters before the dash (1234)

**SuppressBlankRows («string»)**: Returns a string with skipped empty lines. It allows you to create an object and place the fields you want within and suppress the ones that are blank.

Example:

SuppressBlankRows({Var0} & char(10) & {Var1} & char(10) & {Var2}) (Var0, Var1 and Var2 are variables, char(10) is "\n" symbols that mean new line)

| Variables/Values | Formula | Result |
|---|---|---|
| Var0 = "Doris"<br>Var1 = "Bull Run Ranch"<br>Var2 = "Aurora" | *SuppressBlankRows({Var0} & char(10) & {Var1} & char(10) & {Var2})* | Doris<br>Bull Run Ranch<br>Aurora |
| Var0 = "Craig"<br>Var1 = ""<br>Var2 = "Chicago" | *SuppressBlankRows({Var0} & char(10) & {Var1} & char(10) & {Var2})* | Craig<br>Chicago |
| Var0 = "Craig"<br>Var1 = ""<br>Var2 = "Chicago" | *{Var0} & char(10) & {Var1} & char(10) & {Var2}* | Craig<br><br>Chicago |

**trim («*string*»)**: Returns the converted *string* argument. All spaces encountered at the beginning and end of the string are deleted. The number of spaces included between two words is reduced to one.

Example:

trim(" Purchase order") = Purchase order

**trimall («*string*»)**: Returns the converted *string* argument. All spaces encountered are deleted.

Example:

trimall("Paris / New York / Rome") = Paris/NewYork/Rome

**upper («*string*»)**: Gives the *string* converted into upper case.

Example:

upper("Purchase order") = PURCHASE ORDER

**ztrim *(«value»)***: Deletes the zeros from the left side of the numerical value. Fields are entirely numerical.

Example:

If a field named WEIGHT displays a value of 000200, then:

ztrim(weight) = 200

# Defining the properties of a Formula data source

Command: **Data sources > Formula >** formulaname > **Properties**

1. Enter the formula directly in the **Edit** field.

   - or -

   Select the elements, and click **Insert.**

2. Click **OK**.

**Hint**: You can insert an element by double-clicking on it.

**Note**: If a variable used in the formula has a name containing one of the following characters &+-*/<>=^%,!\", it must be enclosed in brackets {}.

**Note**:Dynamic preview represents current formula calculation result, including the formatting defined in Output page. In case of error, the preview is displayed in red. If the value obtained is truncated, you must modify the maximum length specified in the **Output** tab.

# Exercise: Creating a specific modulo

Your software comes with integrated modulo calculation functions for generating barcode check characters.

However, it may be that you need to calculate a specific check character, and, as a result, the integrated functions do not meet your requirements.

You will thus need to create the function yourself. In the next sequence we will see how it is possible to use formulas to create any kind of check character calculations.

This example illustrates how a control character can be calculated for a 2/5 interleaved barcode.

### *Method for calculating a check character*

The method involves multiplying the first character of the data string by 1, the second by 2, the third by 1, and so on.

• Open the label named CHECK01.LAB located in the TUTORIAL folder.

The check character calculation will be carried out on the LOT_NUMBER (26053) data.

## To calculate the weight

1. Click on the **Formula** branch of the **Data Sources** view, and add a new Formula.
2. Enter the following formula:
   mid (LOT_NUMBER,1,1) &
   mid (LOT_NUMBER,2,1) * 2 &
   mid (LOT_NUMBER,3,1) &
   mid (LOT_NUMBER,4,1) * 2 &
   mid (LOT_NUMBER,5,1)

   The result is 2120103, as we have:
   2*1 = 2
   6*2 = 12
   0*1 = 0
   5*2 = 10
   3*1 = 3

   The concatenation of the results gives 2120103.
3. Name the formula ”WEIGHTED”.

## To add up the result of the weight calculation:

The next step involves adding together the figures resulting from the previous formula, bearing in mind that the maximum permitted length of this character string is 2.

1. Create a second Formula, and name it ”SUM”.
2. Enter the following expression in the text box:
   mid (WEIGHTED,1,1) + mid (WEIGHTED,2,1) +
   mid (WEIGHTED,3,1) + mid (WEIGHTED,4,1 )+
   mid (WEIGHTED,5,1) + mid (WEIGHTED,6,1) +
   mid (WEIGHTED,7,1)

   The result is 9; as we have:
   2+1+2+0+1+0+3= 9.

## To calculate the check character:

Using the previous result, we will calculate the value of the check character.

1. Create a third Formula, and name it ”CHECK_DIGIT”.
2. Enter the following expression in the text box:
   if ((SUM % 10) > 0,10 - SUM % 10,0)

   The result is 1; as we have:

SUM % 10 = 9 (% = modulo).
9 being greater than 0, we carry out the subtraction: 10- 9 = 1.

**Note**: If the result had been equal to 0, the value of the check character would have been 0.

## To calculate the data to be encoded:

When creating the barcode, you must include the data to be encoded; for example, the value of the LOT_NUMBER variable concatenated with the value of the check digit (CHECK_DIGIT).

1. Create a fourth formula and name it DATA.
2. Enter the following expression in the text box:
   LOT_NUMBER & CHECK_DIGIT.

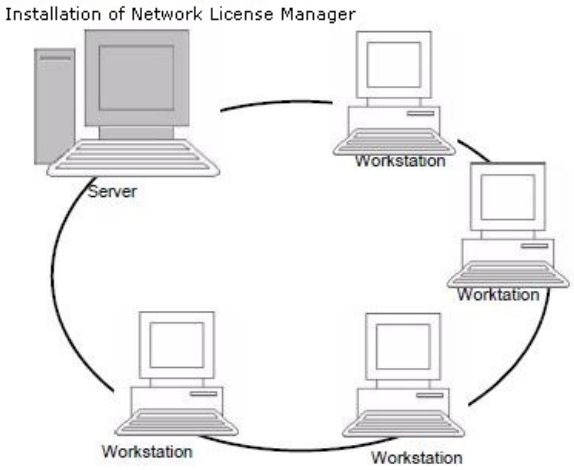
## To create the barcode:

1. Click the **Barcode** tool on the design toolbar. Next, select he DATA formula, then drag and drop it onto the label.
2. Select the barcode, right-click on it and choose **Object properties**. from the context menu. The Barcode dialog box appears.
3. Configure the barcode as follow:
   **Symbology**: 2/5 Interleaved
   **Check Digit**: None

# Installing the network version

## Description

To use the network/multi-user version of your Label Design Software, you must first install Network License Manager, either on the server or on a workstation that will act as the server, then install your labeling software on each workstation.




## Network Installation procedure

### Network configuration

Before you install the software, the network administrator must first define the structure of the network for the group of users:

- Define the license server on which the **Network License Manager** and dongle will be installed.
- Define the workstations, or the client workstations that will use the labeling software.

### Description of Network License Manager

The **Network License Manager** lets you define the network configuration of your labeling software. The **Network License Manager** includes:

- The **Network License Manager** (License Service).

- **Network Settings Wizard**: The Network Settings Wizard helps you define the network configuration.

## Installing the Network License Manager on the server

Before installing the labeling software on all the workstations that will use it, you must install the **Network License Manager** on the server to configure the network.

1. Insert the DVD for the installation in the appropriate drive.
   The Installation window is displayed.

   If the DVD for the installation does not run automatically:
   Go to Windows explorer and expand the letter of the DVD drive. Double click on **index.hta**.
2. Select **Network License Manager**, which includes **License Service**. Then click the **Install** button.
3. Follow the on-screen instructions..
4. If you want to define settings for your network configuration, start the **Network Settings Wizard** on the server. By default, if you do not modify the configuration, each workstation will have its own settings.

## Configuration

All the necessary tools to configure the network version are available from the **Network License Manager toolbar**, which can be accessed from the Windows taskbar (Systray).

From the Window taskbar, double-click on the icon to have the **Network toolbar** displayed.

The **Network Settings Wizard** helps you define the settings for your network version.

1. To start the **Network Settings Wizard**, click on the icon .
2. Select a settings mode: **Generic, By user** or **By station**.

   - **Generic**: All users will use the same settings on all workstations. (user.ini).
   - **By user**: Each user can access his or her own settings on any workstation. (user name.ini).
   - **By station**: Each workstation has its own settings (station.ini).

3. Specify the location in which you want to store these settings. If you want to share these settings between various workstations, specify a network path that is accessible to all workstations (for example TKDongle).

4. Specify the location in which you want to store the shared data (variables, lists, printing logfile, etc.).

## Starting the License Service

Before installing the labeling software on all workstations, you must be sure the License Service is started.

The License Service is installed as a service. This service, referred to as **TkxWebLicenseServer**, is enabled automatically when the server is started.

### To start the License Service Controller

Click the icon available on the **Network License Manager** toolbar.

-or -

Double-click the TkxWebLicenseServerController.exe file in the installation folder.

-or -

Right-click on the **Network License Manager** icon from the Windows taskbar and select **License Service Controller**.

## Installing the software on the workstations

The labeling software must be installed on all the workstations on which it will be used.

### To install the software on a workstation

1. Insert the DVD in the appropriate drive.
   The Installation window is displayed.

   If the DVD does not run automatically:
   Go to Windows explorer and expand the letter of the DVD drive. Double click on **index.hta**.
2. Select the product to be installed, click the **Install** button and follow the instructions on the screen.
3. Start the labeling software. The **License Manager** is displayed. Click **Try** to launch the software. From the **Tools** menu, choose **Network Administration**.
   - or -

From Windows Start menu, select **Network Administration** shortcut in the labeling software group.

4. Enable **Use Network License**.
5. Select the **Type of network license**.

   - **Shared folder license** uses the file sharing feature of Windows to communicate between the software and the License Manager.

   - **Web license** uses http/https communication between the software and the License Manager.

   6. Specify the **server port** if you have selected the **Web license** type.
7. Click **Modify** to select the server on which the license manager and dongle are installed.
   - or -
   Click **Browse** to automatically search for the server on which the license manager is installed.

   If the network has already been configured, a message asking if you want to use the current network configuration is displayed.
8. If you want to modify or configure the network settings, click the **Network Settings Wizard** button.
9. Click **OK**.
10. Restart the program.

**France**
+33 (0) 562 601 080

**Germany**
+49 (0) 2103 2526 0

**Singapore**
+65 6908 0960

**United States**
+1 (414) 837 4800

Copyright 2021 TEKLYNX Corporation SAS. All rights reserved. LABEL MATRIX, LABELVIEW, CODESOFT, LABEL ARCHIVE, SENTINEL, PRINT MODULE, BACKTRACK, TEKLYNX CENTRAL, TEKLYNX, and Barcode Better are trademarks or registered trademarks of TEKLYNX Corporation SAS or its affiliated companies. All other brands and product names are trademarks and/or copyrights of their respective owners.

www.teklynx.com

Microsoft Partner
Gold Independent Software Vendor (ISV)