LABELSTAR OFFICE

User Manual

Version 5.00 Build 1050
October 15, 2016

Contents

Labelstar Office

With this program you can design and print your own labels.

- Simple operation by drag & drop
- Support for all the most common bar code types
- Direct database connection possible
- Individual label design through various printer and system variables
- Mark ups for flexible text formatting
- Print preview, logging, memory card support and other features

Variables

The purpose of variables is to insert certain changeable values on a label, for example the current date.

```
$DateTime ("dd.MM.yyyy HH:mm", UpdateInterval=1, MonthOffset=10)
```

Certain characters within a printout signify and separate individual segments and permit a dismantling and processing of the printout.

The following table describes the reserved characters.

Character	Designation
$	Indicates the start of a variable.
(	Indicates the start of parameter list.
)	Indicates the end of parameter list.
"	Text identification
,	Parameter separator
=	Parameter value separator

See also

- System Variables
- Printer Variables
- Custom Variables

System Variables

With the help of these variables variable field contents for flexible label creation can be defined.
In contrast to Printer Variables, system variables are managed and calculated by the application.

Supported System Variables

- Date/Time Variables
- Field Variables
- Database Field
- Path Variables
- Text Variables
- Counter
- User Input
- Math Variables
- Check Digit Calculations
- Misc Variables
- Execute VBScript

Date/Time Variables

With the help of these variables date and time values can be defined on the label.

Supported Variables

Current Date	$CurrentDate	Returns a value which contains the current date in accordance with the system settings.
Current Date/Time	$CurrentDate/Time	Returns a value which contains the current date and current time in accordance with the system settings.
Current Time	$CurrentTime	Returns a value which contains the current time in accordance with the system settings.
Date/Time	$DateTime	Defines a system date and time variable.
Day of Week	$DayOfWeek	Calculates the day of week.
Day of Year	$DayOfYear	Calculates the day of year.
Format Date/Time	$FormatDateTime	Formats a date/time.
Parse Date/Time	$ParseDateTime	Converts the specified string representation to its date and time equivalent using the specified format and culture-specific format information.
Calendar Week	$WeekOfYear	Calculates the calendar week.

Current Date

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns a value which contains the current date in accordance with the system settings.

Syntax

```
$CurrentDate
```

Return value

Current date according to the system settings.

Examples

```
$CurrentDate -> "10/15/2014"
$Format ($CurrentDate, "yyMMdd") -> "141015"
```

See also

- Current Date/Time
- Current Time
- Date/Time (System)
- Date/Time (Printer)

Current Date/Time

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns a value which contains the current date and time in accordance with the system settings.

Syntax

```
$CurrentDateTime
```

Return value

Current date and time according to the system settings.

Examples

```
$CurrentDateTime -> "10/15/2014 11:03:59 AM"
$Format ($CurrentDateTime, "yyMMdd") -> "141015"
$Format ($CurrentDateTime, "hhmmss") -> "110359"
```

See also

- Current Date
- Current Time
- Date/Time (System)
- Date/Time (Printer)

Current Time

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns a value which contains the current time in accordance with the system settings.

Syntax

```
$CurrentTime
```

Return value

Current time according to the system settings.

Examples

```
$CurrentTime -> "11:03:59 AM"
$Format ($CurrentTime, "hhmmss") -> "110359"
```

See also

- Current Date/Time
- Current Date
- Date/Time (System)
- Date/Time (Printer)

Date/Time

Required program variant LITE, BASIC, PROFESSIONAL

For more information, see Program Variants.

Defines a date and time variable and converts the value with the indicated format in the correpsonding character string presentation.

Syntax

```
$DateTime (format, [Prompt=prompt, UpdateInterval=updateInterval, MonthOffset=monthOffset,
DayOffset=dayOffset, MinOffset=minOffset, StartDate=startDate, Language=language])
```

Parameters

format
Indicates how the date and time is to be formatted.

The *format* parameter should either contain an individual format identifier (see Standard Date and Time Format Strings) or a customized format example (see Custom Date and Time Format Strings), which defines the format of the returned string. If *format* contains the value **null** or an empty string (""), the general format identifier 'G' is used.

prompt (optional, standard = empty)
If a prompt text is defined, the start date is queried at print start.

updateInterval (optional, standard = 0)
Indicates how often the variable is to be updated during a print order.
0: At print start
1: After each label
n: After n labels
-1: At record change

monthOffset (optional, standard = 0)
Month offset (is added to the current date)

dayOffset (optional, standard = 0)
Day offset (is added to the current date)

minOffset (optional, standard = 0)
Minute offset (is added to the current time)

startDate (optional, as default the current date and time according to the system settings is used)
Defines the start date and start time.

language (optional, as default the language set under Windows is used)
Language which is used for formatting the output. For more information, see Country Codes.

Return value

Formatted text.

Examples

```
$DateTime ("dd.MM.yyyy") -> "11.09.2013"
$DateTime ("dd.MM.yyyy", StartDate="6/12/2009", MonthOffset=2) -> "15.08.2009"
$DateTime ("D", UpdateInterval=0, Language="fr", StartDate=$ParseDateTime ("131012", "yyMMdd")) ->
"samedi 12 octobre 2013"
$DateTime ("HH:mm:ss") -> "13:20:35"
```


`$DateTime ("hh:mm:ss")` -> "01:20:35"

ID01 = "260514"

`$DateTime ("D", UpdateInterval=0, StartDate=$ParseDateTime (<<ID01>>, "ddMMyy"))` -> "Monday, June 26, 2014
`$DateTime ("D", UpdateInterval=0, Language="de", StartDate=$ParseDateTime (<<ID01>>, "ddMMyy"))` -> "Montag, 26. Juni 2014"

See also

- Current Date/Time
- Current Date
- Current Time
- Date/Time (Printer)

Day of Week

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Calculates the day of week.

Syntax

```
$DayOfWeek (format, [UpdateInterval=updateInterval, MonthOffset=monthOffset, DayOffset=dayOffset, StartDate=startDate, Sunday=sunday])
```

Parameters

format

Indicates how the weekday is to be formatted.

Format Identifier	Description
d	Day of week, one digit
dd	Day of week, two digits
ddd	Day of week, three digits
dddd	Day of week, four digits
\	Escape character
All other characters	Literals. The character is copied to the result string unchanged.

updateInterval (optional, standard = 0)

Indicates how often the variable is to be updated during a print order.
0: At print start
1: After each label
n: After n labels
-1: At record change

monthOffset (optional, standard = 0)

Month offset (is added to the start date)

dayOffset (optional, standard = 0)

Day offset (is added to the start date)

startDate (optional, as default the current date and time according to the system settings is used)

Defines the start date.

sunday (optional, standard = 0)

integer or character: Defines which value is to be used for Sunday.
string: Defines a character for each weekday, starting with Sunday.

Return value

Formatted day of week.

Examples

Current date: 2/1/2014 (Saturday)


```
$DayOfWeek ("d") -> "6"
$DayOfWeek ("dd") -> "06"
$DayOfWeek ("dd", DayOffset=5) -> "04"
$DayOfWeek ("d", DayOffset=5, Sunday=A) -> "E"
$DayOfWeek ("Day of week: d", StartDate="3/5/2014", Sunday=10) -> "Day of week: 14"
$DayOfWeek ("d", Sunday="ABCDEFG") -> "G"
```


Day of Year

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Calculates the day of year.

Syntax

```
$DayOfYear (format, [UpdateInterval=updateInterval, MonthOffset=monthOffset, DayOffset=dayOffset, StartDate=startDate])
```

Parameters

format

Indicates how the day of year is to be formatted.

Format Identifier	Description
d	Day of year, one digit (1 to 366)
dd	Day of year, two digits (01 to 366)
ddd	Tag im Jahr, three digits (001 to 366)
dddd	Tag im Jahr, four digits (0001 to 0366)
\	Escape character
All other characters	Literals. The character is copied to the result string unchanged.

updateInterval (optional, standard = 0)

Indicates how often the variable is to be updated during a print order.
0: At print start
1: After each label
n: After n labels
-1: At record change

monthOffset (optional, standard = 0)

Month offset (is added to the start data)

dayOffset (optional, standard = 0)

Day offset (is added to the start date)

startDate (optional, as default the current date and time according to the system settings is used)

Defines the start date.

Return value

Formatted day of year.

Examples

Current date: 2/1/2014

```
$DayOfYear ("d") -> "5"
$DayOfYear ("dd") -> "05"
$DayOfYear ("ddd", DayOffset=5) -> "006"
$DayOfYear ("Day of year: dd", StartDate="01.03.2014") -> "Day of year: 09"
```


Format Date/Time

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Formats a date/time.

Syntax

```
$FormatDateTime (datetime, format, [Language=language])
```

Parameters

datetime
A string that contains a date and time to format.

format
A format specifier that defines the required format of *datetime*. For details about valid format specifiers, see Standard Date and Time Format Strings or Custom Date and Time Format Strings.

language (optional, as default the language set under Windows is used)
Language which is used for formatting the output. For more information, see Country Codes.

Return value

Formatted value

Examples

```
$FormatDateTime ("4/12/2016", "D") -> "Tuesday, April 12, 2016"
$FormatDateTime ("12.04.2016", "D", Language="fr") -> "mardi 12 avril 2016"
$FormatDateTime ("13.04.2016", "D", Language="zh-CN") -> "2016年4月13日"
```

See also

- Format Value
- Format Text
- Format Number

Parse Date/Time

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Converts the specified string representation to its date and time equivalent using the specified format and culture-specific format information. The format of the string representation must match the specified format exactly. Otherwise an error occured.

Syntax

```
$ParseDateTime (text, format, [Language=language])
```

Parameters

text
A string that contains a date and time to convert.

format
A format specifier that defines the required format of *text*.

The *format* parameter is a string that contains either a single standard format specifier, or one or more custom format specifiers that define the required format of *text*. For details about valid format specifiers, see Standard Date and Time Format Strings or Custom Date and Time Format Strings.

language (optional, as default the language set under Windows is used)
Language which indicates which culture-specific format information is to used. For more information, see Country Codes.

Return value

Date and time value.

Examples

ID01 = "091410"

```
$ParseDateTime ("130910", "yyMMdd") -> "9/10/2013 12:00:00 AM"
$ParseDateTime (<<ID01>>, "MMyydd") -> "9/10/2014 12:00:00 AM"
```

See also

- Date/Time (System)

Calendar Week

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Calculates the calendar week.

Syntax

```
$WeekOfYear (format, [UpdateInterval=updateInterval, MonthOffset=monthOffset, DayOffset=dayOffset,
StartDate=startDate, Language=language])
```

Parameters

format

Indicates how the calendar week is to be formatted.

Format Identifier	Description
w	Calendar week, one digit (1 to 53)
ww	Calendar week, two digits (01 to 53)
www	Calendar week, three digits (001 to 053)
wwww	Clanedar week, four digits (0001 to 0053)
\	Escape character
All other characters	Literals. The character is copied to the result string unchanged.

updateInterval (optional, standard = 0)

Indicates how often the variable is to be updated during a print order.
0: At print start
1: After each label
n: After n labels
-1: At record change

monthOffset (optional, Standard = 0)

Month offset (is added to the current date)

dayOffset (optional, Standard = 0)

Day offset (is added to the current date)

startDate (optional, as default the current date set in the system settings is used)

Defines the start date.

language (optional, as default the language set under Windows is used)

Language which indicates which culture-specific format information is to used. For more information, see Country Codes.

Return value

Formatted calendar week.

Examples

Current date: 2/1/2014


```
$WeekOfYear ("w") -> "5"
$WeekOfYear ("ww") -> "05"
$WeekOfYear ("www", DayOffset=5) -> "006"
$WeekOfYear ("Calendar \week: ww", StartDate="3/1/2014") -> "Calendar week: 09"
```

Current date: 3/4/2016

```
$WeekOfYear ("ww", Language="de") -> "09"
$WeekOfYear ("ww", Language="en") -> "10"
```

See also

- «Advanced» Tab

Field Variables

With the help of field variables, linkings between individual elements can be defined on the label.

Supported Variables

Field Link	$FieldLink	Returns the content of a field.
Field Name	$FieldName	Returns the field name.

Field Link

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the content of a field.

Syntax

```
$FieldLink (fieldName, [dummyText])
```

or

```
<<fieldName>>
```

Parameters

fieldName
Field name
Note: Upper and lower case is considered

dummyText (optional, Standard = empty)
Indicates whether another value for the screen display is to be used than the actual field contents.
Note: For the printout, the current field content is always used.

Return value

Field contents

Examples

ID01 = "12345"
ID02 = "abcABC"

```
$FieldLink (ID01) -> "12345"
$FieldLink (ID01, "00000") -> "00000"
$FieldLink (ID02) -> "abcABC"
$FieldLink (ID02, "XXXXXX") -> "XXXXXX"

<<ID01>> -> "12345"
<<ID02, "XXXXXX">> -> "XXXXXX"
```

See also

- Field Link (Printer)

Field Name

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the field name.

Syntax

```
$FieldName
```

Return value

Field name

Database Field

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Inserts a database field on the label.

Syntax

```
$DbField (dbName, columnName, [DBNullValue=dbNullValue, Format=format])
```

Parameters

dbName
Database name

columnName
Column name
Note: Upper and lower case is considered

dbNullValue (optional, Standard = empty)
Indicates which value is to be used if the appropriate database field is empty.

format (optional, Standard = empty)
Indicates how the contents of database field is to be formatted. For more information, see Formatting Types.

Return value

Contents of database field.

Examples

ID	Name	Capital	Area	Population	NativeName	Flag
12	United Kingdom	London	244820	60440000	United Kingdom	

```
$DbField ("Europe", "Area") -> "60440000"
$DbField ("Europe", "Area", Format="0000000000") -> "0000244820"
$DbField ("Europe", "Capital", Format="LLLL") -> "Lond"
$ToUpper ($DbField ("Europe", "Capital")) -> "LONDON"
```

Check whether a database field is empty

```
$If ($Length ($DBField (...)) = 0, "The database field is empty.", "The database field is not emptry.")
```

See also

- Databases
- Database Field (Printer)

Path Variables

With the help of the path variables, path strings can be read out and processed.

Supported Variables

Application Data Folder	$AppDataDir	Returns the full path to the file directory containing application data for all users.
Application Folder	$AppDir	Returns the current application folder.
Application Path	$AppPath	Returns the complete path name of the application.
Folder Name	$Dir	Returns the directory information for the specified path string.
File Extension	$Ext	Returns the extension of the specified path string.
File Name	$FileName	Returns the file name of the specified path string.
Image Folder	$ImageDir	Returns the current image folder.
Installation Folder	$InstallDir	Returns the full path to the directory in which **Labelstar Office** is installed.
Label Folder	$LabelDir	Returns the current label folder.
Label Path	$LabelPath	Returns the complete path of the current label file.

Application Data Folder

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns the full path to the file directory containing application data for all users.

Syntax

```
$AppDataDir
```

Return value

Windows XP: C:\Documents and Settings\All Users\Application Data\Labelstar Office
Windows 7/8: C:\ProgramData\Labelstar Office

Application Folder

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns the current application folder.

Syntax

```
$AppDir
```

Return value

The path for the executable file that started the application, not including the executable name (e.g. "C:\Programs\Carl Valentin GmbH\Labelstar Office").

See also

- Application Path
- Installation Folder

Application Path

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the complete path name of the application.

Syntax

```
$AppPath
```

Return value

The path for the executable file that started the application, including the executable name (e.g. "C:\Programs\Carl Valentin GmbH\Labelstar Office\LabelDesigner.exe").

See also

- Application Folder
- Installation Folder

Folder Name

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the directory information for the specified path string.

Syntax

```
$Dir (path)
```

Parameters

path
The path of a file or directory.

Return value

Directory information for *path*, or **null** if *path* denotes a root directory or is **null**. Returns an empty string if *path* does not contain directory information.

Examples

```
$Dir ("C:\Labels\Label.lbex") -> "C:\Labels"
$Dir ($AppPath) -> "C:\Programs\Carl Valentin GmbH"
```

See also

- File Name
- File Extension

File Extension

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the extension of the specified path string.

Syntax

```
$Ext (path)
```

Parameters

path
The path string from which to get the extension.

Return value

The extension of the specified path (including the period "."), or an empty string, if *path* does not have extension information.

Examples

```
$Ext ("C:\label.lbex") -> ".lbex"
$Ext ($AppPath) -> ".exe"
$Ext ("C:\label") -> ""
```

See also

- File Name
- Folder Name

File Name

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the file name of the specified path string.

Syntax

```
$FileName (path, [Ext=extension])
```

Parameters

path
The path string.

extension (optional, default = true)
true|1: Returns file name with extension
false|0: Returns file name without extension

Return value

The file name of the specified path string with or without extension.

Examples

```
$FileName ("C:\Labels\Label.lbex") -> "Label.lbex"
$FileName ($AppPath) -> "LabelDesigner.exe"
$FileName ($AppPath, Ext=false) -> "LabelDesigner"
$FileName ($LabelPath) -> "Label.lbex"
$FileName ($LabelPath, Ext=0) -> "Label"
```

See also

- Folder Name
- File Extension

Image Folder

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the current image folder.

Syntax

```
$ImageDir
```

Return value

Current image folder

See also

- Change image folder
- Label Folder

Installation Folder

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns the full path to the directory in which **Labelstar Office** is installed.

Syntax

```
$InstallDir
```

Return value

The installation folder of **Labelstar Office** (e.g. "C:\Programs\Carl Valentin GmbH\Labelstar Office" or "C:\Programs (x86)\Carl Valentin GmbH\Labelstar Office").

See also

- Application Folder
- Application Path

Label Folder

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the current label folder.

Syntax

```
$LabelDir
```

Return value

Current label folder

See also

- Change label folder
- Image Folder

Text Variables

With the help of this variables strings can be edited.

Supported Variables

Format Text	$FormatText	Formats a text.
Convert HEX String to ASCII String	$HexToString	Converts a HEX string to an ASCII string.
Get Leftmost Characters of String	$Left	Returns a string containing a specified number of characters from the left side of a string.
String Length	$Length	Gets the number of characters in the current string.
Get Middle Characters of String	$Mid	Returns a string that contains a specified number of characters starting from a specified position in a string.
Pad String from Left	$PadLeft	Increases the length of the string by adding spaces or a specified character to the beginning. Strings with length greater than or equal to the required length will be unchanged.
Pad String from Right	$PadRight	Increases the length of the string by adding spaces or a specified character to the end. Strings with length greater than or equal to the required length will be unchanged.
Delete Characters	$Remove	Returns a string in which a specified number of characters, beginning at a specified position, have been deleted.
Replace String	$Replace	Returns a string in which all occurrences of a specified string are replaced with another specified string.
Replace Pattern	$ReplacePattern	Replaces all strings that match a specified regular expression with a specified replacement string. Specified options modify the matching operation.
Reverse String	$Reverse	Returns a string in which the character order of a specified string is reversed.
Get Rightmost Characters of String	$Right	Returns a string containing a specified number of characters from the right side of a string.
Convert ASCII String to HEX String	$StringToHex	Converts an ASCII string to a HEX string.
Convert String to Lowercase	$ToLower	Returns a copy of this string converted to lowercase.
Convert String to Uppercase	$ToUpper	Returns a copy of this string converted to uppercase.
Trim Leading and Trailing Characters	$Trim	Removes all leading and trailing occurrences of a set of characters specified in an array from the current string.
Trim Leading Characters	$TrimLeft	Removes all leading occurrences of a set of characters specified in an array from the current string.
Trim Trailing Characters	$TrimRight	Removes all trailing occurrences of a set of characters specified in an array from the current string.
Truncate String	$Truncate	Restricts the maximum length of a string and cuts the string if it is too long.

Format Text

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Formats a text.

Syntax

```
$FormatText (text, format)
```

Parameters

text
Text which is to be formatted.

format
Indicates how the text is to be formatted. For more information, see Text Format Strings.

Return value

Formatted text.

Examples

```
$FormatText ("1234", "0000-00") -> "1234-00"
$FormatText ("1234", "!0000-00") -> "0012-34"
$FormatText ($DbField ("Cookies", "ProductCode"), "!00.00.00") -> "03.25.00"
```

See also

- Format Value
- Format Number
- Format Date/Time

Convert HEX String to ASCII String

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Converts a HEX string to an ASCII string.

Syntax

```
$HexToString (text)
```

Parameter

text

The HEX string to be converted.
Note: An individual hexadecimal value consists always of two digits and can contain only numbers (0-9) and letters (a-f, A-F).

Return value

The ASCII string.

Examples

```
$HexToString ("3132333435") -> "12345"
$HexToString ("61626358595A") -> "abcXYZ"
```

See also

- Convert ASCII String to HEX String

Get Leftmost Characters of String

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns a string containing a specified number of characters from the left side of a string.

Syntax

```
$Left (text, length)
```

Parameters

text

String expression from which the leftmost characters are returned.

length

Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *text*, the entire string is returned.

Return value

A string containing a specified number of characters from the left side of a string.

Examples

```
$Left ("abcDEF", 0) -> ""
$Left ("abcDEF", 2) -> "ab"
$Left ("abcDEF", 4) -> "abcD"
$Left ("abcDEF", 10) -> "abcDEF"
$Left ("abcDEF", -2) -> Error
```

See also

- Get Rightmost Characters of String
- Get Middle Characters of String

String Length

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Gets the number of characters in the current string.

Syntax

```
$Length (text)
```

Parameter

text
The string whose length is to be calculated.

Rückgabewert

The number of characters in the current string.

Beispiele

```
$Length ("abcDEF") -> 6
$Length ("") -> 0
```

Check whether a database field is empty

```
$If ($Length ($DBField (...)) == 0, "The database field is empty.", "The database field is not empty.")
```


Get Middle Characters of String

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns a string that contains a specified number of characters starting from a specified position in a string.

Syntax

```
$Mid (text, startIndex, [length])
```

Parameters

text
String expression from which characters are returned.

startIndex
Starting position of the characters to return. If *startIndex* is greater than the number of characters in *text*, the function returns a zero-length string ("").
Note: The starting position is zero based.

length (optional)
Number of characters to return. If omitted or if there are fewer than *length* characters in *text* (including the character at position *index*), all characters from the start position to the end of the string are returned.

Return value

A string that consists of the specified number of characters starting from the specified position in the string.

Examples

```
$Mid ("abcDEF", 3) -> "DEF"
$Mid ("abcDEF", 3, Length=2) -> "DE"
```

See also

- Get Leftmost Characters of String
- Get Rightmost Characters of String

Pad String from Left

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Increases the length of the string by adding spaces or a specified character to the beginning. Strings with length greater than or equal to the required length will be unchanged.

Syntax

```
$PadLeft (text, totalWidth, [PaddingChar=paddingChar])
```

Parameter

text
The string to be changed.

totalWidth
The number of characters in the resulting string, equal to the number of original characters plus any additional padding characters.

paddingChar (optional, standard = space character)
A padding character.

Return value

The changed string.

Examples

```
$PadLeft ("abcDEF", 10) -> "    abcDEF"
$PadLeft ("12345", 10, PaddingChar="0") -> "0000012345"
```

See also

- Pad String from Right

Pad String from Right

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Increases the length of the string by adding spaces or a specified character to the end. Strings with length greater than or equal to the required length will be unchanged.

Syntax

```
$PadRight (text, totalWidth, [PaddingChar=paddingChar])
```

Parameter

text
The string to be changed.

totalWidth
The number of characters in the resulting string, equal to the number of original characters plus any additional padding characters.

paddingChar (optional, standard = space character)
A padding character.

Return value

The changed string.

Examples

```
$PadRight ("abcDEF", 10) -> "abcDEF    "
$PadRight ("12345", 10, PaddingChar="0") -> "1234500000"
```

See also

- Pad String from Left

Delete Characters

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns a string in which a specified number of characters, beginning at a specified position, have been deleted.

Syntax

```
$Remove (text, startIndex, [length])
```

Parameters

text
The string to be changed.

startIndex
The zero-based position to begin deleting characters.

length (optional, standard = 0)
The number of characters to delete. If *length* is 0 a string is returned in which all characters, beginning at a specified position and continuing through the last position, have been deleted.

Return value

The changed string.

Examples

```
$Remove ("abcDEF", 3) -> "abc"
$Remove ("abcDEF", 3, Length=2) -> "abcF"
```


Replace String

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns a string in which all occurrences of a specified string are replaced with another specified string.

Syntax

```
$Replace (text, oldValue, newValue, [oldValue, newValue, ...])
```

Parameters

text
The string to be changed.

oldValue
The string to be replaced.

newValue
The string to replace all occurrences of *oldValue*.

Return value

The changed string.

Examples

```
$Replace ("abcDEFabcDEF", "abc", "") -> "DEFDEF"
$Replace ("abcDEFabcDEF", "abc", "ABC") -> "ABCDEFABCDEF"
$Replace ("abcDEFabcDEF", "ab", "AB", "EF", "ef") -> "ABcDefABcDef"
```

See also

- Replace Pattern

Replace Pattern

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Replaces all strings that match a specified regular expression with a specified replacement string. Specified options modify the matching operation.

Syntax

```
$ReplacePattern (text, pattern|filename, replacement, [IgnoreCase=ignoreCase,
RightToLeft=rightToLeft])
```

Parameters

text
The string to search for a match.

pattern|filename
The regular expression pattern to match or the file name of a text file that contains the pattern definition. The *pattern* parameter consists of regular expression language elements that symbolically describe the string to match. For more information about regular expressions, see Regular Expression Language.

replacement
The replacement string.

ignoreCase (optional, default = false)
true|1: Specifies case-insensitive matching.
false|0: Specifies case-sensitive matching.

rightToLeft (optional, default = false)
Change the search direction.
true|1: Specifies that the search will be from right to left.
false|0: Specifies that the search will be from left to right.

Return value

The changed string.

Examples

```
$ReplacePattern ("abcdefABCDEF", "abc|DEF", "<b>$0</b>") -> "abcdefABCDEF"
$ReplacePattern ("abcdefABCDEF", "abc", "<u><b>$0</b></u>", IgnoreCase=true) -> "abcdefABCDEF"
```

Strip Invalid Characters from a String

In this case, `$ReplacePattern` strips out all nonalphanumeric characters except periods (.), at symbols (@), and hyphens (-), and returns the remaining string.

```
$ReplacePattern ("<email>@example.com", "[^\w\.@-]", "") -> "email@example.com"
```

For more examples, see Food Allergen Labelling.

See also

- Replace String

Regular Expression Language

A regular expression is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs.

Character Escapes

The backslash character (\) in a regular expression indicates that the character that follows it either is a special character (as shown in the following table), or should be interpreted literally.

Escaped character	Description	Pattern	Input string	Matches
\a	Matches a bell character, \u0007.	`\a`	"Error!" + '\u0007'	"\u0007"
\b	In a character class, matches a backspace, \u0008.	`[\b]{3,}`	"\b\b\b\b"	"\b\b\b\b"
\t	Matches a tab, \u0009.	`(\w+)\t`	"item1\titem2\t"	"item1\t", "item2\t"
\r	Matches a carriage return, \u000D. (**\r** is not equivalent to the newline character, **\n**.)	`\r\n(\w+)`	"\r\nThese are \ntwo lines."	"\r\nThese"
\v	Matches a vertical tab, \u000B.	`[\v]{2,}`	"\v\v\v"	"\v\v\v"
\f	Matches a form feed, \u000C.	`[\f]{2,}`	"\f\f\f"	"\f\f\f"
\n	Matches a new line, \u000A.	`\r\n(\w+)`	"\r\nThese are \ntwo lines."	"\r\These"
\e	Matches an escape, \u001B.	`\e`	"\x001B"	"\x001B"
**** *nnn*	Uses octal representation to specify a character (*nnn* consists of two or three digits).	`\w\040\w`	"a bc d"	"a b", "c d"
\x *nn*	Uses hexadecimal representation to specify a character (*nn* consists of exactly two digits).	`\w\x20\w`	"a bc d"	"a b", "c d"
\c *X* **\c** *x*	Matches the ASCII control character that is specified by *X* or *x*, where *X* or *x* is the letter of the control character.	`\cC`	"\x0003" (Strg-C)	"\x0003"
\u *nnnn*	Matches a Unicode character by using hexadecimal representation (exactly four digits, as represented by *nnnn*).	`\w\u0020\w`	"a bc d"	"a b", "c d"
****	When followed by a character that is not recognized as an escaped character in this and other tables in this topic, matches that character. For example, ***** is the same as **\x2A**, and **\.** is the same as **\x2**E. This allows the regular expression engine to disambiguate language elements (such as * or ?) and character literals (represented by * or \?).	`\d+[\+-x\*]\d+\d+[\+-x\*\d+`	"(2+2) * 3*9"	"2+2", "3*9"

Character Classes

A character class matches any one of a set of characters. Character classes include the language elements listed in the following table.

Character class	Description	Pattern	Input string	Matches
[*character_group* **]**	Matches any single character in *character_group*. By default, the match is case-sensitive.	`[ae]`	"gray" "lane"	"a" "a", "e"

[^ *character_group* **]**	Negation: Matches any single character that is not in *character_group*. By default, characters in *character_group* are case-sensitive.	`[^aei]`	"reign"	"r", "g", "n"
[*first* **-** *last* **]**	Character range: Matches any single character in the range from *first* to *last*.	`[A-Z]`	"AB123"	"A", "B"
.	Wildcard: Matches any single character except **\n**. To match a literal period character (. or **\u002E**), you must precede it with the escape character (**\.**).	`a.e`	"nave" "water"	"ave" "ate"
\p{ *name* **}**	Matches any single character in the Unicode general category or named block specified by *name*.	`\p{IsCyrillic}`	"ДЖem"	"Д", "Ж"
\P{ *name* **}**	Matches any single character that is not in the Unicode general category or named block specified by *name*.	`\P{IsCyrillic}`	"ДЖem"	"e", "m"
\w	Matches any word character.	`\w`	"ID A1.3"	"I", "D", "A", "1", "3"
\W	Matches any non-word character.	`\W`	"ID A1.3"	" ", "."
\s	Matches any white-space character.	`\w\s`	"ID A1.3"	"D "
\S	Matches any non-white-space character.	`\s\S`	"int __ctr"	" _"
\d	Matches any decimal digit.	`\d`	"4 = IV"	"4"
\D	Matches any character other than a decimal digit.	`\D`	"4 = IV"	" ", "=", " ", "I", "V"

Assertion

Atomic zero-width assertions, cause a match to succeed or fail depending on the current position in the string, but they do not cause the engine to advance through the string or consume characters.

Assertion	Description	Pattern	Input string	Matches
^	The match must start at the beginning of the string or line.	`^\d{3}`	"901-333-"	"901"
$	The match must occur at the end of the string or before **\n** at the end of the line or string.	`-\d{3}$`	"-901-333"	"-333"
\A	The match must occur at the start of the string.	`\A\d{3}`	"901-333-"	"901"
\Z	The match must occur at the end of the string or before **\n** at the end of the string.	`-\d{3}\Z`	"-901-333"	"-333"
\z	The match must occur at the end of the string.	`-\d{3}\z`	"-901-333"	"-333"
\G	The match must occur at the point where the previous match ended.	`\G\(\d\)`	"(1)(3)(5)[7](9)"	"(1)", "(3)", "(5)"
\b	The match must occur on a boundary between a **\w** (alphanumeric) and a **\W** (nonalphanumeric) character.	`\b\w+\s\w+\b`	"them theme them them"	"them theme", "them them"
\B	The match must not occur on a **\b** boundary.	`\Bend\w*\b`	"end sends endure lender"	"ends", "ender"

Grouping Constructs

Grouping constructs delineate subexpressions of a regular expression and typically capture substrings of an input string. Grouping constructs include the language elements listed in the following table.

Grouping construct	Description	Pattern	Input string	Matches

(*subexpression* **)**	Captures the matched subexpression and assigns it a one-based ordinal number.	`(\w)\1`	"deep"	"ee"
(?< *name* **>** *subexpression* **)**	Captures the matched subexpression into a named group.	`(?<double>\w)\k<double>`	"deep"	"ee"
(?< *name1* **-** *name2* **>** *subexpression* **)**	Defines a balancing group definition.	`(((?'Open'\()[^\(\)]*)+((?'Clo Open'\))[^\(\)]*)+)*(?(Open)(?!))$`	"3+2^((1-3)*(3-1))"	"((1-3)*(3-1))"
(?: *subexpression* **)**	Defines a noncapturing group.	`Write(?:Line)?`	"Console.WriteLine()"	"WriteLine"
(?imnsx-imnsx: *subexpression* **)**	Applies or disables the specified options within *subexpression*.	`A\d{2}(?i:\w+)\b`	"A12xl A12XL a12xl"	"A12xl", "A12XL"
(?= *subexpression* **)**	Zero-width positive lookahead assertion.	`\w+(?=\.)`	"He is. The dog ran. The sun is out."	"is", "ran", "out"
(?! *subexpression* **)**	Zero-width negative lookahead assertion.	`\b(?!un)\w+\b`	"unsure sure unity used"	"sure", "used"
(?<= *subexpression* **)**	Zero-width positive lookbehind assertion.	`(?<=19)\d{2}\b`	"1851 1999 1950 1905 2003"	"99", "50", "05"
(?<! *subexpression* **)**	Zero-width negative lookbehind assertion.	`(?<!19)\d{2}\b`	"1851 1999 1950 1905 2003"	"51", "03"
(?> *subexpression* **)**	Nonbacktracking (or "greedy") subexpression.	`[13579](?>A+B+)`	"1ABB 3ABBC 5AB 5AC"	"1ABB", "3ABB", "5AB"

Quantifiers

A quantifier specifies how many instances of the previous element (which can be a character, a group, or a character class) must be present in the input string for a match to occur. Quantifiers include the language elements listed in the following table.

Quantifier	Description	Pattern	Input string	Matches
*	Matches the previous element zero or more times.	`\d*\. \d`		".0", "19.9", "219.9"
+	Matches the previous element one or more times.	`"be+"`	"been" "bent"	"bee" "be"
?	Matches the previous element zero or one time.	`"rai? n"`		"ran", "rain"
{ *n* }	Matches the previous element exactly *n* times.	`",\d{3}"`	"1,043.6" "9,876,543,210"	",043" ",876", ",543", ",210"
{ *n* ,}	Matches the previous element at least *n* times.	`"\d{2,}"`		"166", "29", "1930"
{ *n* , *m* }	Matches the previous element at least *n* times, but no more than *m* times.	`"\d{3,5}"`	"193024"	"19302"
*?	Matches the previous element zero or more times, but as few times as possible.	`\d*? \. \d`		".0", "19.9", "219.9"
+?	Matches the previous element one or more times, but as few times as possible.	`"be+?"`	"been" "bent"	"be" "be"
??	Matches the previous element zero or one time, but as few times as possible.	`"rai?? n"`		"ran", "rain"
{ *n* }?	Matches the preceding element exactly *n* times.	`",\d{3}?"`	"1,043.6" "9,876,543,210"	",043" ",876", ",543", ",210"

{ *n* **,}?**	Matches the previous element at least *n* times, but as few times as possible.	`"\d{2,}?"`		"166", "29", "1930"
{ *n* **,** *m* **}?**	Matches the previous element between *n* and *m* times, but as few times as possible.	`"\d{3,5}?"`	"193024"	"193", "024"

Backreference Constructs

A backreference allows a previously matched subexpression to be identified subsequently in the same regular expression. The following table lists the backreference constructs supported by regular expressions.

Backreference construct	Description	Pattern	Input string	Matches
**** *number*	Backreference. Matches the value of a numbered subexpression.	`(\w)\1`	"seek"	"ee"
\k< *name* **>**	Named backreference. Matches the value of a named expression.	`(?<char>\w)\k<char>`	"seek"	"ee"

Alternation Constructs

Alternation constructs modify a regular expression to enable either/or matching. These constructs include the language elements listed in the following table.

Alternation construct	Description	Pattern	Input string	Matches
|	Matches any one element separated by the vertical bar (|) character.	`th(e|is|at)`	"This is the day."	"the", "this"
(?(*expression* **)** *yes* **|** *nein* **)**	Matches *yes* if the regular expression pattern designated by expression matches; otherwise, matches the optional *no* part. expression is interpreted as a zero-width assertion.	`(?(A)A\d{2}\b|\b\d{3}\b)`	"A10 C103 910"	"A10", "910"
(?(*name* **)** *yes* **|** *no* **)**	Matches *yes* if name, a named or numbered capturing group, has a match; otherwise, matches the optional *no*.	`(?<quoted>")?(?(quoted).+?"|\S+\s)`	"Dogs.jpg "Yiska playing.jpg""	"Dogs.jpg", ""Yiska playing.jpg""

Substitutions

Substitutions are regular expression language elements that are supported in replacement patterns. The metacharacters listed in the following table are atomic zero-width assertions.

Character	Description	Pattern	Replacement pattern	Input string	Result string
$ *number*	Substitutes the substring matched by group *number*.	`\b(\w+)(\s)(\w+)\b`	`$3$2$1`	"one two"	"two one"
${ *name* **}**	Substitutes the substring matched by the named group *name*.	`\b(?<word1>\w+)(\s)(?<word2>\w+)\b`	`${word2} ${word1}`	"one two"	"two one"
$$	Substitutes a literal "$".	`\b(\d+)\s?USD`	`$$1`	"103 USD"	"$103"
$&	Substitutes a copy of the whole match.	`\$? \d*\.?\d+`	`**$&**`	"$1.30"	"**$1.30**"
$`	Substitutes all the text of the input string before the match.	`B+`	`$``	"AABBCC"	"AAAACC"

$'	Substitutes all the text of the input string after the match.	B+	$'	"AABBCC"	"AACCCC"
$+	Substitutes the last group that was captured.	B+(C+)	$+	"AABBCCDD"	AACCDD
$_	Substitutes the entire input string.	B+	$_	"AABBCC"	"AAAABBCCCC"

Regular Expression Options

You can specify options that control how the regular expression engine interprets a regular expression pattern.

You can specify an inline option in two ways:

- By using the miscellaneous construct **(?imnsx-imnsx)**, where a minus sign (-) before an option or set of options turns those options off. For example, **(?i-mn)** turns case-insensitive matching **(i)** on, turns multiline mode **(m)** off, and turns unnamed group captures **(n)** off. The option applies to the regular expression pattern from the point at which the option is defined, and is effective either to the end of the pattern or to the point where another construct reverses the option.
- By using the grouping construct **(?imnsx-imnsx:***subexpression***)**, which defines options for the specified group only.

The regular expression engine supports the following inline options.

Option	Description	Pattern	Input string	Matches
i	Use case-insensitive matching.	\b(?i)a(?-i)a\w+\b	"aardvark AAAuto aaaAuto Adam breakfast"	"aardvark", "aaaAuto"
m	Use multiline mode. **^** and **$** match the beginning and end of a line, instead of the beginning and end of a string.			
n	Do not capture unnamed groups.			
s	Use single-line mode.			
x	Ignore unescaped white space in the regular expression pattern.	\b(?x) \d+ \s \w+	"1 aardvark 2 cats IV centurions"	"1 aardvark", "2 cats"

Miscellaneous Constructs

Miscellaneous constructs either modify a regular expression pattern or provide information about it. The following table lists the supported miscellaneous constructs.

Construct	Beschreibung	Muster	Eingabezeichenfolge	Entsprechungen
(?imnsx-imnsx)	Sets or disables options such as case insensitivity in the middle of a pattern. For more information, see Regular Expression Options.	\bA(?i)b\w+\b	"ABA Able Act"	"ABA", "Able"
(?# *comment* **)**	Inline comment. The comment ends at the first closing parenthesis.	\bA(?#Matches words starting with A)\w+\b		
# [bis Zeilenende**]**	X-mode comment. The comment starts at an unescaped # and continues to the end of the line.	(?x)\bA\w+ \b#Matches words starting with A		

Reverse String

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns a string in which the character order of a specified string is reversed.

Syntax

```
$Reverse (text)
```

Parameter

text

String expression whose characters are to be reversed. If *text* is a zero-length string (""), a zero-length string is returned.

Return value

A string in which the character order of a specified string is reversed.

Examples

```
$Reverse ("abcDEF") -> "FEDcba"
$Reverse ("12345") -> "54321"
```


Get Rightmost Characters of String

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns a string containing a specified number of characters from the right side of a string.

Syntax

```
$Right (text, length)
```

Parameters

text
String expression from which the rightmost characters are returned.

length
Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *text*, the entire string is returned.

Return value

A string containing a specified number of characters from the right side of a string.

Examples

```
$Right ("abcDEF", 0) -> ""
$Right ("abcDEF", 2) -> "EF"
$Right ("abcDEF", 4) -> "cDEF"
$Right ("abcDEF", 10) -> "abcDEF"
$Right ("abcDEF", -2) -> Error
```

See also

- Get Leftmost Characters of String
- Get Middle Characters of String

Convert ASCII String to HEX String

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Converts an ASCII string to a HEX string.

Syntax

```
$StringToHex (text)
```

Parameter

text

The ASCII string to be converted.
Note: Each individual character is converted in a two-digit hexadecimal value.

Return value

The HEX string.

Examples

```
$StringToHex ("12345") -> "3132333435"
$StringToHex ("abcXYZ") -> "61626358595A"
```

See also

- Convert HEX String to ASCII String

Convert String to Lowercase

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns a copy of this string converted to lowercase.

Syntax

```
$ToLower (text)
```

Parameter

text
The string to be changed.

Return value

A string in lowercase.

Examples

```
$ToLower ("abcDEF") -> "abcdef"
```

See also

- Convert String to Uppercase

Convert String to Uppercase

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns a copy of this string converted to uppercase.

Syntax

```
$ToUpper (text)
```

Parameter

text
The string to be changed.

Return value

A string in uppercase.

Examples

```
$ToUpper ("abcDEF") -> "ABCDEF"
```

See also

- Convert String to Lowercase

Trim Leading And Trailing Characters

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Removes all leading and trailing occurrences of a set of characters specified in an array from the current string.

Syntax

```
$Trim (text, [TrimChars=trimChars])
```

Parameter

text
The string to be changed.

trimChars (optional)
An array of characters to remove.

Return value

The string that remains after all occurrences of the characters in the *trimChars* parameter are removed from the start and end of the current string. If *trimChars* is not defined, white-space characters are removed instead.

Examples

```
$Trim ("  abcDEF  ") -> "abcDEF"
$Trim ("abcDEF", TrimChars="aF") -> "bcDE"
```

See also

- Trim Leading Characters
- Trim Trailing Characters

Trim Leading Characters

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Removes all leading occurrences of a set of characters specified in an array from the current string.

Syntax

```
$TrimLeft (text, [TrimChars=trimChars])
```

Parameter

text
The string to be changed.

trimChars (optional)
An array of characters to remove.

Return value

The string that remains after all occurrences of characters in the *trimChars* parameter are removed from the start of the current string. If *trimChars* is not defined, white-space characters are removed instead.

Examples

```
$TrimLeft ("  abcDEF  ") -> "abcDEF "
$TrimLeft ("abcDEF", TrimChars="a") -> "bcDEF"
```

See also

- Trim Leading and Trailing Characters
- Trim Trailing Characters

Trim Trailing Characters

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Removes all trailing occurrences of a set of characters specified in an array from the current string.

Syntax

```
$TrimRight (text, [TrimChars=trimChars])
```

Parameter

text
The string to be changed.

trimChars (optional)
An array of characters to remove.

Return value

The string that remains after all occurrences of characters in the *trimChars* parameter are removed from the end of the current string. If *trimChars* is not defined, white-space characters are removed instead.

Examples

```
$TrimRight ("  abcDEF  ") -> " abcDEF"
$TrimRight ("abcDEF", TrimChars="F") -> "abcDE"
```

See also

- Trim Leading and Trailing Characters
- Trim Leading Characters

Truncate String

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Restricts the maximum length of a string and cuts the string if it is too long.

Syntax

```
$Truncate (text, maxLength)
```

Parameter

text
The string to be changed.

maxLength
Maximum number of characters (including the caret "..."). If the number of characters in *text* greater than *maxLength*, the string is cut and "..." is added.

Return value

The truncated string.

Examples

```
$Truncate ("Beispieltext", 8) -> "Beisp..."
$Truncate ($LabelPath, 20) -> "C:\...\Label1.lbex"
```


Counter

Required program variant LITE, BASIC, PROFESSIONAL

For more information, see Program Variants.

Inserts a system counter on the label.

Syntax

```
$Counter (value, [Prompt=prompt, UpdateInterval=updateInterval, Increment=increment,
MinValue=minValue, MaxValue=maxValue, TrimLeft=trimLeft, Mode=mode, Radix=radix])
```

Parameter

value

The current start value.
Note: The number of digits determines the output format (maximum "999999999").

prompt (optional, standard = empty)

If a prompt text is defined, the start value is queried at print start.

updateInterval (optional, standard = 1)

Indicates how often a variable is to be updated during a print order.
1: After each label
n: After n labels
-1: After each change of data record

increment (optional, standard = 1)

Increment.

minValue (optional, standard = empty)

Minimum value: If no *minValue* is defined, as default the number of start value digits is used to calculate a minimum value.

Start value	Radix	Calculated minimum value
0001	10	0000
001A	16	0000
ABC	1	AAA

maxValue (optional, standard = empty)

Maximum value: If no *maxValue* is defined, as default the number of start value digits is used to calculate a maximum value.

Start value	Radix	Calculated maximum value
0001	10	9999
001A	16	FFFF
ABC	1	ZZZ

trimLeft (optional, standard = false)

true|1: Remove leading zeros at output
false|0: Show leading zeros at the output

mode (optional, standard = 3)

Operating mode
0: Reset start value at print start
1: Reset start value at print start (automatic rollover)
2: Reset start value manually
3: Reset start value manually (automatic rollover)

radix (optional, standard = 10)
Radix, basis of counter (1-36)
1: Alphabetical (A-Z)
2: Binäry (0, 1)
8: Octal (0-7)
10: Decimal (0-9)
16: Hexadecimal (0-9, A-F)
36: Alphanumerical (0-9, A-Z)

Return value

The current counter value.

Examples

`$Counter ("0001", MinValue="0000", MaxValue="0009", Increment=1, Radix=10)` -> 0001, 0002, 0003, 0004, 0005, 0006, 0007, 0008, 0009, 0009, 0009, ...
`$Counter ("0001", Increment=1, TrimLeft=true, Radix=10)` -> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...
`$Counter ("0001", MinValue="0000", MaxValue="0009", Increment=-1, StartMode=0, Radix=10)` -> 0009, 0008, 0007, 0006, 0005, 0004, 0003, 0002, 0001, 0000, 0000, 0000, ...
`$Counter ("0001", MinValue="0000", MaxValue="0009", Increment=1, StartMode=1, Radix=10)` -> 0000, 0001, 0002, 0003, 0004, 0005, 0006, 0007, 0008, 0009, 0000, 0001, ...

Number of copies = "200"

`$Counter (`$Copies`, Increment=-1, Radix=10)` -> 200, 199, 198, 197, 196, 195, 194, 193, 192, 191, 190, ...

Hexadecimal counter

`$Counter ("0009", MinValue="0000", MaxValue="FFFF", Increment=1, Radix=16)` -> 0009, 000A, 000B, 000C, 000D, 000E, 000F, 0010, 0011, 0012, ...

See also

- Global Counter
- $PrnCounter - Counter (Printer)

Global Counter

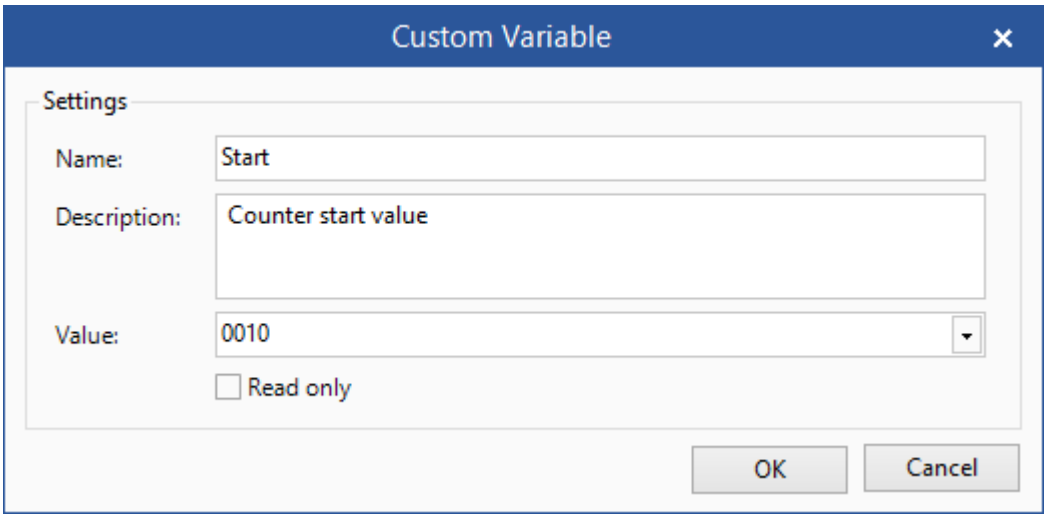
Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

The global counter is a special case of a Counter (System). The start value becomes global, i.e. defined and saved across labels.

To define a global counter, please proceed as follows:

1. Define a new user-defined variable.



2. Add a new text or bar code field on the label or select an existing field and open **Expression Builder**.
3. Define a Counter (System) and insert as start value the user-defined variable.
   `$Counter ($Start, UpdateInterval=1, Increment=1, Mode=3, Radix=10)` -> 0010, 0011, 0012, 0013, 0014, 0015, ...

User Input

Required program variant LITE, BASIC, PROFESSIONAL

For more information, see Program Variants.

Inserts a system user input on the label.

Syntax

```
$UserInput ([startText])
```

Parameters

startText (optional, standard = empty)
Start text that is displayed when editing the user input. If *startText* is empty the internal start text is used.

Internal Parameters

Prompt text
Prompt text that is displayed when editing the user input.

Update interval
Indicates how often a variable is to be updated during a print order.

Start text
Start text that is displayed when editing the user input. Click ... to define the type of the input field.

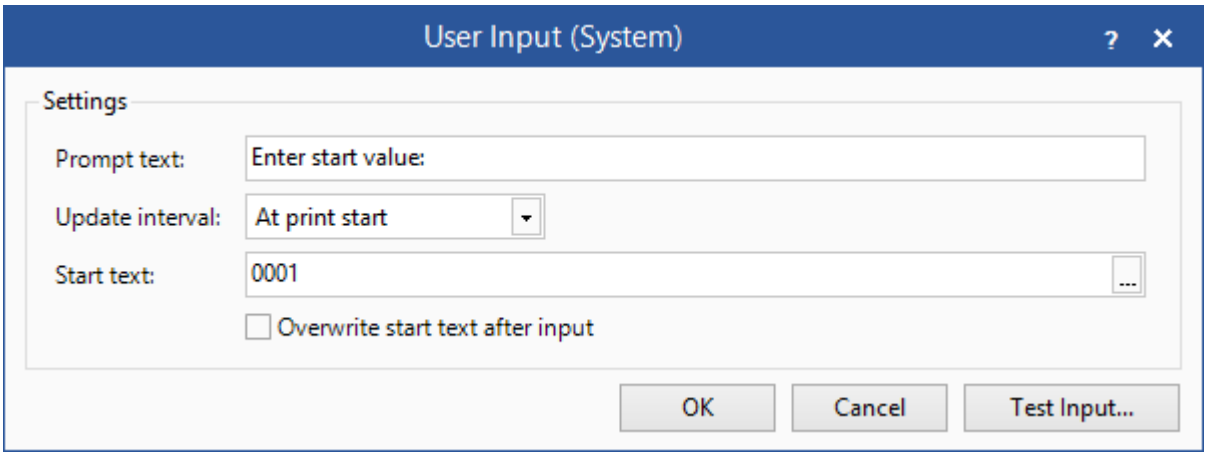
Field Type	Description
Text Box	Single-line text input
Drop-Down List	Displays a list of items from which a selection can be made.
Date/Time Picker	Date and/or time input
Slider	Can be used to select a value from a predefined range.
Check Box	Activates or deactivates a value.

Overwrite start text after input
If this option is enabled text entered by the user overwrites existing start text; otherwise, the current start text is maintained.

Return value

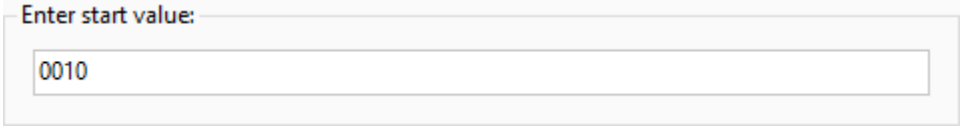
The entered text.

Example (Text Box)




```
$UserInput ->
```


```
$UserInput ("0010") ->
```



	ID	Name	Capital	Area	Population	NativeName	Flag
▶	12	United Kingdom	London	244820	60440000	United Kingdom	

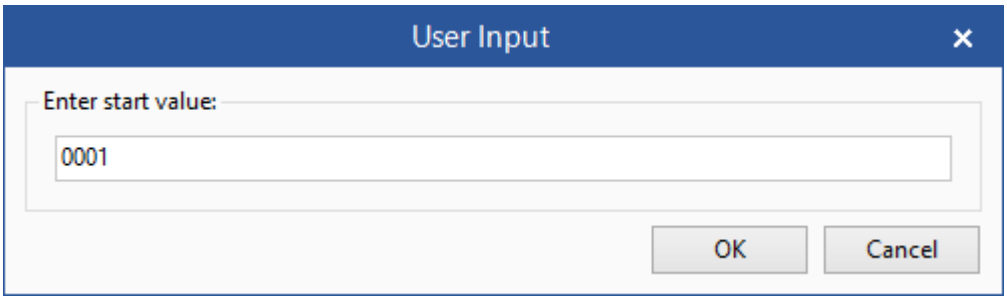
```
$UserInput ($DBField ("Europe", "Area")) ->
```

See also

- User Input (Printer)

Text Box

Single-line text input



Properties

	Name	Description
	AllowableCharacters	Determines which characters can be entered into the field. *All* - Letters, digits and special charcters *Numeric* - Digits *Alpha* - Letters *Alphanumeric* - Letters and digits *InputMask* - Input mask *Custom* - User-defined number of characters
	CustomCharacters	Defines a list of valid characters (e.g. "abcdef0123456789" for entering hexadecimal values). Will only be considered when the *AllowableCharacters* property is set to *Custom*.
	Font	Indicates the control font. If no font is defined, the default Windows font is used.
	InputMask	Indicates the input mask at runtime. Will only be considered when the *AllowableCharacters* property is set to *InputMask*. For more information, see Inputs Mask Format Strings.
	MaxLength	Indicates the maximum number of characters that can be entered in the text box.
	MinLength	Indicates the minimum number of characters that can be entered in the text box.
	OutputFormat	Indicates a value that determines whether literals and prompt characters are included in the output string. Will only be considered when the *AllowableCharacters* property is set to *InputMask*. *ExcludePromptAndLiterals* - Return only text input by the user. *IncludePrompt* - Return text input by the user as well as any literal characters defined in the mask. *IncludeLiterals* - Return text input by the user as well as any instances of the prompt character. *IncludePromptAndLiterals* - Return text input by the user as well as any literal characters defined in the mask and any instances of the prompt character.
	PaddedOn	Indicates that the text is to be filled to the maximum number of characters. *NotUsed* - Return only text input by the user. *Left* - The text input will be right aligned and filled to the maximum number of characters. *Right* - The text input will be left aligned and filled to the maximum number of characters.
	PaddingChar	Indicates character used to fill text when text contains less than maximum number of characters.
	PromptChar	Indicates character used to represent the absence of user input in the text box.

Input Mask Format Strings

The input mask must be a string composed of one or more of the masking elements, as shown in the following table.

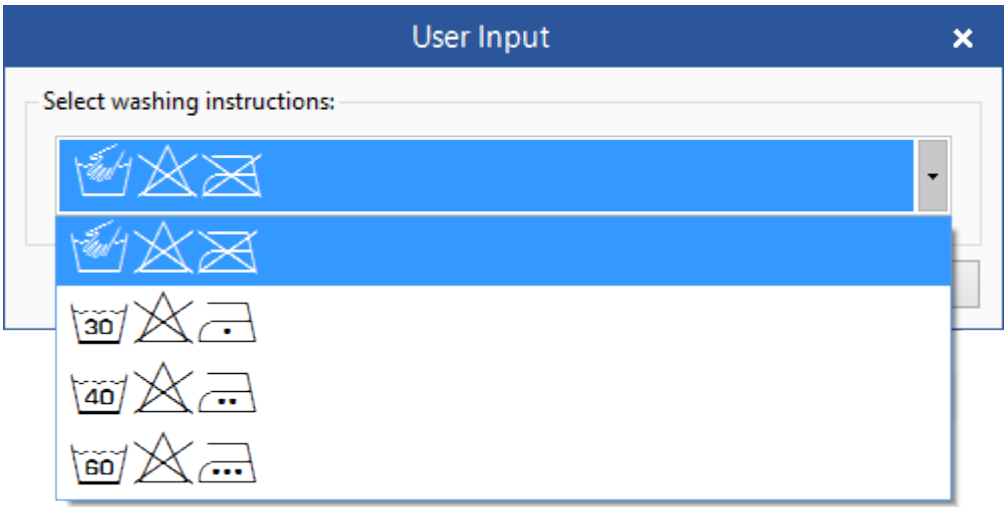
Masking Element	Description
0	Digit (entry required)
9	Digit or space (entry optional)
#	Digit/Space/+/- (entry optional)
L	Letter (entry required)
?	Letter (entry optional)
A	Letter or digit (entry required)
a	Letter or digit (entry optional)
&	Any character (entry required)
C	Any character (entry optional)
.	Decimal placeholder
,	Thousands placeholder
:	Time separator
/	Date separator
$	Currency symbol
<	Shift down. Converts all characters thats follows to lowercase.
>	Shift up. Converts all characters thats follows to uppercase.
|	Disable previous shift up or shift down.
\	Esacpe. Escapes a mask character, turning it into a literal.
All other characters	Literals. All non-mask elements will appear as themselves. Literals always occupy a static position in the input mask at run time, and cannot be moved or deleted by the user.

Examples

Mask	Behavior
00/00/0000	A date (day, numeric month, year) in international date format. The "/" character is a logical date separator, and will appear to the user as the date separator appropriate to the application's current culture.
00->L<LL-0000	A date (day, month abbreviation, and year) in United States format in which the three-letter month abbreviation is displayed with an initial uppercase letter followed by two lowercase letters.
(999)-000-0000	United States phone number, area code optional. If users do not want to enter the optional characters, they can either enter spaces or place the mouse pointer directly at the position in the mask represented by the first 0.
$999,999.00	A currency value in the range of 0 to 999999. The currency, thousandth, and decimal characters will be replaced at run time with their culture-specific equivalents.

Drop-Down List

Displays a list of items from which a selection can be made.

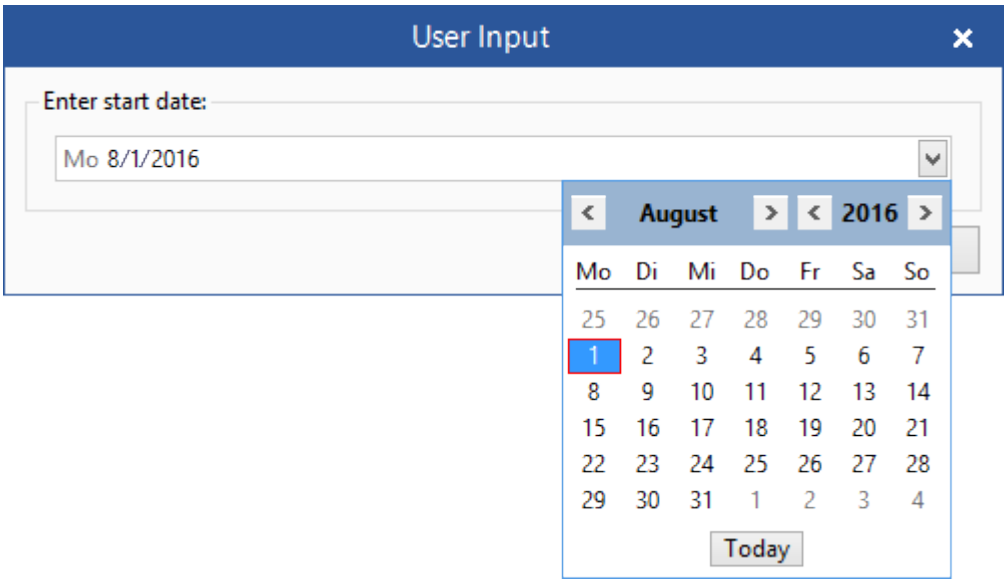


Properties

	Name	Description
	Font	Indicates the control font. If no font is defined, the default Windows font is used.
	Items	Represents the collection of the items contained in the combo box. A item has the following properties: *DisplayText* - Text to be displayed in the list. *ValueText* - Text to be returned. If no *ValueText* is defined, the *DisplayText* is returned.
	MaxDropDownItems	Indicates the maximum number of items to be shown in the drop-down portion of the combo box.
	Sorted	Indicates a value indicating whether the items in the combo box are sorted.
	WatermarkText	Indicates watermark text displayed when no item is selected.

Date/Time Picker

Date and/or time input

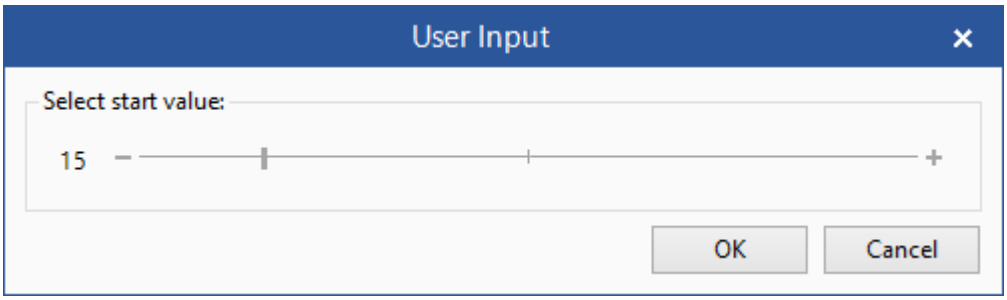


Properties

	Name	Description
	CustomFormat	Indicates the custom date/time format string. Will only be considered when the *Format* property is set to *Custom*. For more information, see Standard Date and Time Format Strings and Custom Date and Time Format Strings.
	Font	Indicates the control font. If no font is defined, the default Windows font is used.
	Format	Indicates the format date/time is displayed in. *Default* - 5/15/2015 12:00:00 *LongDate* - Date (long) - Friday, May 15, 2015 *ShortDate* - Date (short) - 5/15/2015 *LongTime* - Time - 12:00:00 *ShortTime* - Time (without seconds) - 12:00 *Custom* - User-defined format string
	Input	Defines what can be entered by the user. *Date* - Enter date *Time* - Enter time *DateTime* - Enter date and time
	MaxDate	Indicates maximum date that can be selected in the control.
	MinDate	Indicates minimum date that can be selected in the control.
🔒	Preview	Preview

Slider

Can be used to select a value from a predefined range.



Properties

	Name	Description
	DecreaseTooltip	Indicates tooltip for the Decrease (-) button of the slider.
	DisplayFormat	Indicates how the value is to be displayed. For more information, see Display Formats.
	Font	Indicates the control font. If no font is defined, the default Windows font is used.
	IncreaseTooltip	Indicates tooltip for the Increase (+) button of the slider.
	LabelVisible	Indicates whether the label text is displayed.
	LabelWidth	Indicates width of the label part of the item in pixels. If the value is 0, the size is calculated automatically.
	MaxValue	Indicates the maximum value of the range of the control.
	MinValue	Indicates the minimum value of the range of the control.
	Step	Indicates the amount by which a call to the PerformStep method increases the current position of the slider. Value must be greater than 0.

Display Formats

Numbers can be formatted in many ways. Following examples shows how to align numbers, how to format negative numbers or how to do custom formatting like phone numbers.

Add zeroes before number

Format	Value	Output
00000	15	00015
00000	-15	-00015

Different formatting for negative numbers and zero

You can have special format for negative numbers and zero. Use semicolon separator „;“ to separate formatting to two or three sections. The second section is format for negative numbers, the third section is for zero.

Format	Value	Output
#;minus #	15	15
#;minus #	-15	minus 15
#;minus #;zero	0	zero

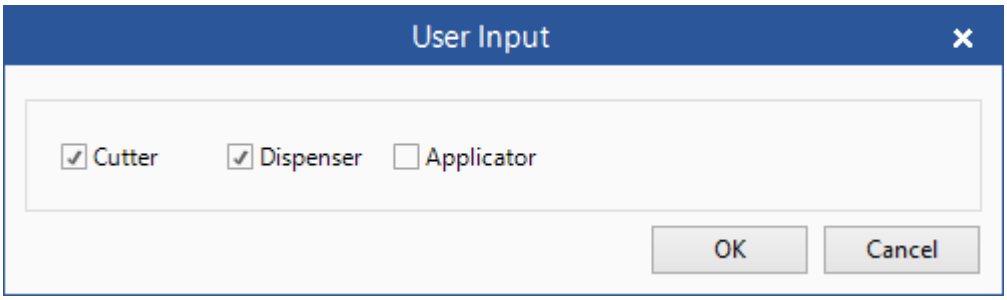
Custom number formatting

Numbers can be formatted also to any custom format, e.g. like phone numbers or serial numbers.

Format	Value	Output
+### ### ### ###	447900123456	+447 900 123 456
##-####-####	8958712551	89-5871-2551

Check Box

Activates or deactivates a value.



Eigenschaften

	Name	Beschreibung
	CheckedValue	Indicates what is to be printed when option is selected.
	Font	Indicates the control font. If no font is defined, the default Windows font is used.
	UncheckedValue	Indicates what is to be printed when option is not selected.

Mathematical Variables

With the help of these variables numbers can be processed and mathematic formulas can be calculated.

Supported Variables

Absolute Value	$Abs	Returns the absolute value of a number.
Format Number	$FormatNumber	Formats a number.
Calculate Formula	$MathField	Calculates a mathematical formula.
Maximum Value	$Max	Returns the larger of two numbers.
Minimum Value	$Min	Returns the smaller of two numbers.
Calculate Sum	$Sum	Returns the sum of two (or more) numbers.

Absolute Value

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the absolute value of a specified number.

Syntax

```
$Abs (value)
```

Parameters

value
A number.

Return value

Absolute value of the specified number.

Remarks

The absolute value of a number is its numeric value without its sign. For example, the absolute value of both 1.2 and -1.2 is 1.2.

Examples

```
$Abs (12.00) -> "12"
$Abs (-12.25) -> "12.25"
$Format ($Abs (-144), "00000") -> "00144"
```


Caluclate Formula

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Calculates a mathematical formula.

Syntax

```
$MathField (expression, [format])
```

Parameters

expression
Expression which is to be calculated. For more information, see Mathematical Operators.

format (optional, standard = empty)
Indicates how the result is to be formatted.

The *format* parameter should either contain an individual format identifier (see Standard Numeric Format Strings) or a customized format string (see Custom Numeric Format Strings), which defines the format of the returned value.

Return value

Calculated value.

Examples

ID01 = "-10.00"
ID02 = "12.00"

```
$MathField (12 * 12) -> "144"
$MathField (<<ID01>> + <<ID02>>) -> "2"
$MathField ($Abs (<<ID01>>) + <<ID02>>) -> "22"

$MathField ((12 * 12) / 10) -> "14,4"
$Format ($MathField ((12 * 12) / 10), "0.00") -> "14,40"
$Format ($MathField ((12 * 12) / 10), "0") -> "14"

$MathField (12.4 * 12.0) -> "148,8"
$MathField (12.4 * 12.0, "N2") -> "148,80"
$MathField (12.4 * 12.0, "0") -> "149"
```

	ID	Name	Capital	Area	Population	NativeName	Flag
▶	12	United Kingdom	London	244820	60440000	United Kingdom	

```
$MathField ($DbField ("Europe", "Population") * 2.00) -> "120880000"
```

Mathematical Operators

An operator is a term of a symbol to which one or several expressions and/or operands are handed over as input and which returns a value.

Unary Operators (operators with one operand)

Printout	Description
+x	Identity
-x	Negation
!x	Logical negation

Arithmetic Operators

Printout	Description
x + y	Addition, string concatenation
x - y	Subtraction
x * Y	Multiplication
x / y	Division
x % y	Modulus (calculates the remainder of the two operands)
x ^ y	Power (calculates the y-th power of x)

Compare Operators

Printout	Description
x = y or x == y	Equal to
x != y or x <> y	Not equal to
x < y	Less than
x <= y	Less than or equal to
x > y	Greater than
x >= y	Greater than or equal to

Logical Operators

Printout	Description
x && y	Conditioned And (y is evaluated only if x is true)
x || y	Conditioned Or (y is evaluated only if x is false)

Mathematical Constants

Symbol	Value	Description
E	2.718281828...	Euler's number e Base of natural logarithms.
Phi	1.618033988...	Golden Ratio Φ
Pi	3.141592653...	Archimedes' constant π Ratio of the circumference of a circle to its diameter.

Maximum Value

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the larger of two numbers.

Syntax

```
$Max (value1, value2)
```

Parameters

value1
The first of two numbers to compare.

value2
The second of two numbers to compare.

Return value

Parameter *value1* or *value2*, whichever is larger.

Examples

```
$Max (10, 20) -> "20"
$Max (10, 5) -> "10"
```

See also

- Minimum Value

Minimum Value

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the smaller of two numbers.

Syntax

```
$Max (value1, value2)
```

Parameters

value1
The first of two numbers to compare.

value2
The second of two numbers to compare.

Return value

Parameter *value1* or *value2*, whichever is smaller.

Examples

```
$Max (10, 20) -> "20"
$Max (10, 5) -> "10"
```

See also

- Maximum Value

Calculate Sum

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the sum of two (or more) numbers.

Syntax

```
$Sum (number1, number2, [number3, ...])
```

Parameters

number
A number.

Return value

Sum of the specified numbers.

Examples

ID01 = "20"

```
$Sum (10, 2) -> "12"
$Format ($Sum (<<ID01>>, 2), "00000") -> "00022"
```

Check Digits Calculations

With the help of these variables check digits can be calculated.

Supported Variables

Check Digit	$CheckDigit	Calculates a check digit.
Custom Check Digit	$CustomCheckDigit	Calculates a user-defined check digit.

Check Digit

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Calculates a check digit.

Syntax

```
$CheckDigit (data, checkDigitMethod, [AppendTo=appendTo])
```

Parameters

data
Data for which the check digit is to be calculated.

checkDigitMethod
Method after according to which the check digit is to be calculated.

Method	Description
MOD10	Modulo 10
MOD10_LUHN	Modulo 10 (Luhn-Algorithmus)
MOD11	Modulo 11
MOD43	Modulo 43
MOD47_15	Modulo 47 (weighting 15)
MOD47_20	Modulo 47 (weighting 20)
MOD103	Module 103

appendTo (optional, standard = empty)
Specifies where the calculated check digit is to be appended to the data.
Right: Indicates that the check digit is appended at the end of the data.
Left: Indicates that the check digit is inserted at the beginning of the data.

Return value

Calculated check digit.

Examples

NVE = "34012345123456789"

```
$CheckDigit ("12345", MOD10) -> "7"
$CheckDigit (<<NVE>>, MOD10) -> "5"
$CheckDigit (<<NVE>>, MOD10, AppendTo=Right) -> "340123451234567895"
```

See also

- Check Digit (Printer)
- Custom Check Digit (System)

- Custom Check Digit (Printer(

Custom Check Digit

Required program variant **PROFESSIONAL**

For more information, see Program Variants.

Calculates a user-defined check digit.

You can create completely custom check digit algorithms, if built-in are not sufficient. This function is very useful, if you want to add your own security to the bar codes. The algorithm can be defined based on the Modulo functionality and is a derivative of the Modulo 10 algorithm used in EAN-13. If you want to create different or more complex check digit algorithms, use the Visual Basic script variable to create it.

Syntax

```
$CustomCheckDigit (data, [AppendTo=appendTo])
```

Parameters

data
Data for which the check digit is to be calculated.

appendTo (optional, standard = empty)
Specifies where the calculated check digit is to be appended to the data.
Right: Indicates that the check digit is appended at the end of the data.
Left: Indicates that the check digit is inserted at the beginning of the data.

Return value

Calculated check digit.

Examples

NVE = "34012345123456789"

```
$CustomCheckDigit ("12345") -> "7"
$CustomCheckDigit (<<NVE>>) -> "5"
$CustomCheckDigit (<<NVE>>, AppendTo=Right) -> "340123451234567895"
```

See also

- Check Digit (System)
- Check Digit (Printer)
- Custom Check Digit (Printer)

Misc Variables

With the help of these variables different information can be defined on the label.

Supported Misc Variables

Number of Copies	$Copies	Returns the number of copies.
Format Value	$Format	Formats a value.
Format Date/Time	$FormatDateTime	Formats a date/time.
Format Number	$FormatNumber	Formats a number.
Format Text	$FormatText	Formats a text.
If..Then..Else Statement	$If	Inserts an If..Then..Else statement on the label.
Label Name	$LabelName	Returns the name of the current label file.
Label Number	$LabelNumber	Displays the current label number within the print order on the label.
Label Path	$LabelPath	Returns the complete path of the current label file.
Page Name	$PageName	Displays the current page name on the label.
Page Number	$PageNumber	Displays the current page number within a print order on the label.
Printer Name	$PrinterName	Displays the current printer name on the label.
Shift Description	$Shift	Displays the current shift description on the label.
User Domain Name	$UserDomianName	Returns the network domain name associated with the current user.
User Name	$UserName	Returns the current user name.
VBScript	$VBScript	Executes a VBScript.

Number of Copies

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Displays the current number of copies on the label.

Syntax

```
$Copies
```

Return value

Number of copies

Examples

Number of copies = 80
Label number = 5

```
$Format ($LabelNumber, "000") - $Format ($Copies, "000")
```
-> "005 - 080"

Format Value

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Formats a value.

Syntax

```
$Format (value, format, [Language=language, Type=type])
```

Parameters

value
Value which is to be formatted.

format
Indicates how the value is to be formatted. For more information, see Formatting Types.

language (optional, as default the language set under windows is used)
Language which is used to format the ouput. For more information, see Country Codes.

type (optional)
Indicates the type of *value* which is to be formatted. If no explicite type is indicated, it is first checked for number, then date/time and then for text.

Type	Description
number	*value* is interpreted as number.
datetime	*value* is interpreted as date/time.
text	*value* is interpreted as text.

Return value

Formatted value.

Examples

Show number with leading zeros

```
$Format (15, "00000") -> "00015"
$Format (-15, "00000") -> "-00015"
$Format (-15, "D5") -> "-00015"
```

Different formatting for negative numbers and zero

Note: You can define special formats for negative numbers and zero. Use a semicolon ";" as separator in order to separate the formatting in two or three sections. The second section is for negative numbers, the third section is for zero.

```
$Format (15, "#;minus #") -> "15"
$Format (-15, "#;minus #") -> "minus 15"
$Format (0, "#;minus #;zero") -> "zero"
```

Using different languages

Note: The default language is German (de-DE) if *language* is not specified.

```
$Format (1234.56, "N2") -> "1.234,56"
$Format (1234.56, "N2", Language="en-US") -> "1,234.56"
$Format (1234.56, "N2", Language="fr-FR") -> "1 234,56"

$Format ($CurrentDateTime, "yyyy MMMM dddd") -> "2013 September Dienstag"
$Format ($CurrentDateTime, "yyyy MMMM dddd", Language="fr-FR") -> "2013 septembre mardi"
$Format ($CurrentDateTime, "yyyy MMMM dddd", Language="zh-CN") -> "2013 九月星期二"
```

Using different ouput types

```
$Format ("123456", "00.00.00") -> "123456,0000"
$Format ("123456", "00.00.00", Type="number") -> "123456,0000"
$Format ("123456", "number:00.00.00") -> "123456,0000"
$Format ("123456", "00.00.00", Type="text") -> "12.34.56"
$Format ("123456", "text:00.00.00") -> "12.34.56"
```

See also

- Format Text
- Format Number
- Format Date/Time

Formatting Types

Formatting converts the value of a type into a string representation.

See also

- Standard Numeric Format Strings
- Custom Numeric Format Strings
- Standard Date and Time Format Strings
- Custom Date and Time Format Strings
- Text Format Strings

Standard Numeric Format Strings

Standard numeric format strings are used to format common numeric types. A standard numeric format string takes the form Axx, where A is an alphabetic character called the format specifier, and xx is an optional integer called the precision specifier. The precision specifier ranges from 0 to 99 and affects the number of digits in the result. Any numeric format string that contains more than one alphabetic character, including white space, is interpreted as a custom numeric format string. For more information, see Custom Numeric Format Strings.

The following table describes the standard numeric format specifiers.

Format specifier	Name	Description	Examples
C or c	Currency	Result: A currency value. Precision specifier: Number of decimal digits.	123.456 ("C", en-US) -> $123.46 123.456 ("C", fr-FR) -> 123,46 € 123.456 ("C", ja-JP) -> ¥123 -123.456 ("C3", en-US) -> ($123.456) -123.456 ("C3", fr-FR) -> -123,456 € -123.456 ("C3", ja-JP) -> -¥123.456
D or d	Decimal	Result: Integer digits with optional negative sign. Precision specifier: Minimum number of digits.	1234 ("D") -> 1234 -1234 ("D6") -> -001234
E or e	Exponential (scientific)	Result: Exponential notation. Precision specifier: Number of decimal digits.	1052.0329112756 ("E", en-US) -> 1.052033E+003 1052.0329112756 ("e", fr-FR) -> 1,052033e+003 -1052.0329112756 ("e2", en-US) -> -1.05e+003 -1052.0329112756 ("E2", fr_FR) -> -1,05E+003
F or f	Fixed-point	Result: Integral and decimal digits with optional negative sign. Precision specifier: Number of decimal digits.	1234.567 ("F", en-US) -> 1234.57 1234.567 ("F", de-DE) -> 1234,57 1234 ("F1", en-US) -> 1234.0 1234 ("F1", de-DE) -> 1234,0 -1234.56 ("F4", en-US) -> -1234.5600 -1234.56 ("F4", de-DE) -> -1234,5000
G or g	General	Result: The most compact of either fixed-point or scientific notation. Precision specifier: Number of significant digits.	-123.456 ("G", en-US) -> -123.456 123.456 ("G", sv-SE) -> -123,456 123.4546 ("G4", en-US) -> 123.5 123.4546 ("G4", sv-SE) -> 123,5 -1.234567890e-25 ("G", en-US) -> -1.23456789E-25 -1.234567890e-25 ("G", sv-SE) -> -1,23456789E-25
N or n	Number	Result: Integral and decimal digits, group separators, and a decimal separator with optional negative sign. Precision specifier: Desired number of decimal places.	1234.567 ("N", en-US) -> 1,234.57 1234.567 ("N", ru-RU) -> 1 234,57 1234 ("N1", en-US) -> 1,234.0 1234 ("N1", ru-RU) -> 1 234,0 -1234.56 ("N3", en-US) -> -1,234.560 -1234.56 ("N3", ru-RU) -> -1 234,560
P or p	Percent	Result: Number multiplied by 100 and displayed with a percent symbol. Precision specifier: Desired number of decimal places.	1 ("P", en-US) -> 100.00 % 1 ("P", fr-FR) -> 100,00 % -0.39678 ("P1", en-US) -> -39.7 % -0.39678 ("P1", fr-FR) -> -39,7 %
R or r	Round-trip	Result: A string that can round-trip to an identical number. Precision specifier: Ignored.	123456789.12345678 ("R") -> 123456789.12345678 -1234567890.12345678 ("R") -> -1234567890.1234567
X or x	Hexadecimal	Result: A hexadecimal string.	255 ("X") -> FF

		Precision specifier: Number of digits in the result string.	-1 ("x") -> ff 255 ("x4") -> 00ff -1 ("X4") -> 00FF
Any other single character	Unknown specifier		

Custom Numeric Format Strings

You can create a custom numeric format string, which consists of one or more custom numeric specifiers, to define how to format numeric data. A custom numeric format string is any format string that is not a Standard Numeric Format Strings.

The following table describes the custom numeric format specifiers.

Format specifier	Name	Description	Examples
0	Zero placeholder	Replaces the zero with the corresponding digit if one is present; otherwise, zero appears in the result string.	1234.5678 ("00000") -> 01235 0.45678 ("0.00", en-US) -> 0.46 0.45678 ("0.00", fr-FR) -> 0,46
#	Digit placeholder	Replaces the pound sign with the corresponding digit if one is present; otherwise, no digit appears in the result string.	1234.5678 ("#####") -> 1235 0.45678 ("#.##", en-US) -> .46 0.45678 ("#.##", fr-FR) -> ,46
.	Decimal point	Determines the location of the decimal separator in the result string.	0.45678 ("0.00", en-US) -> 0.46 0.45678 ("0.00", fr-FR) -> 0,46
,	Group separator and number scaling	Serves as both a group separator and a number scaling specifier. As a group separator, it inserts a localized group separator character between each group. As a number scaling specifier, it divides a number by 1000 for each comma specified.	Group separator specifier: 2147483647 ("##,#", en-US) -> 2,147,483,647 2147483647 ("##,#", es-ES) -> 2.147.483.647 Scaling specifier: 2147483647 ("#,#,,", en-US) -> 2,147 2147483647 ("#,#,,", es-ES) -> 2.147
%	Percentage placeholder	Multiplies a number by 100 and inserts a localized percentage symbol in the result string.	0.3697 ("%#0.00", en-US) -> %36.97 0.3697 ("%#0.00", el-GR) -> %36,97 0.3697 ("##.0 %", en-US) -> 37.0 % 0.3697 ("##.0 %", el-GR) -> 37,0 %
‰	Per mille placeholder	Multiplies a number by 1000 and inserts a localized per mille symbol in the result string.	0.03697 ("#0.00‰", en-US) -> 36.97‰ 0.03697 ("#0.00‰", ru-RU) -> 36,97‰
\	Escape character	Causes the next character to be interpreted as a literal rather than as a custom format specifier.	987654 ("\###00\#") -> #987654#
;	Section separator	Defines sections with separate format strings for positive, negative, and zero numbers.	12.345 ("plus #0.0#;minus #0.0#;null") -> plus 12.35 0 ("plus #0.0#;minus #0.0#;null") -> null -12.345 ("plus #0.0#;minus #0.0#;null") -> minus 12.35
All other characters	Literals	The character is copied to the result string unchanged.	68 ("# °") -> 68 °

Standard Date and Time Format Strings

A standard date and time format string uses a single format specifier to define the text representation of a date and time value. Any date and time format string that contains more than one character, including white space, is interpreted as a custom date and time format strin. For more information, see Custom Date and Time Format Strings.

The following table describes the standard date and time format specifiers.

Format specifier	Description	Examples
d	Short date pattern.	15.06.2009 13:45:30 -> 6/15/2009 (en-US) 15.06.2009 13:45:30 -> 15/06/2009 (fr-FR) 15.06.2009 13:45:30 -> 2009/06/15 (ja-JP)
D	Long date pattern.	15.06.2009 13:45:30 -> Monday, June 15, 2009 (en-US) 15.06.2009 13:45:30 -> 15 июня 2009 г.(ru-RU) 15.06.2009 13:45:30 -> Montag, 15.Juni 2009 (de-DE)
f	Full date/time pattern (short time).	15.06.2009 13:45:30 -> Monday, June 15, 2009 1:45 PM (en-US) 15.06.2009 13:45:30 -> Höhle 15 juni 2009 13:45 (sv-SE) 15.06.2009 13:45:30 -> Δευτέρα, 15 Ιουνίου 2009 1:45 μμ (el-GR)
F	Full date/time pattern (long time).	15.06.2009 13:45:30 -> Monday, June 15, 2009 1:45:30 PM (en-US) 15.06.2009 13:45:30 -> den 15 juni 2009 13:45:30 (sv-SE) 15.06.2009 13:45:30 -> Δευτέρα, 15 Ιουνίου 2009 1:45:30 μμ (el-GR)
g	General date/time pattern (short time).	15.06.2009 13:45:30 -> 6/15/2009 1:45 PM (en-US) 15.06.2009 13:45:30 -> 15/06/2009 13:45 (es-ES) 15.06.2009 13:45:30 -> 2009/6/15 13:45 (zh-CN)
G	General date/time pattern (long time).	15.06.2009 13:45:30 -> 6/15/2009 1:45:30 PM (en-US) 15.06.2009 13:45:30 -> 15/06/2009 13:45:30 (es-ES) 15.06.2009 13:45:30 -> 2009/6/15 13:45:30 (zh-CN)
M oder m	Month/day pattern.	15.06.2009 13:45:30 -> June 15 (en-US) 15.06.2009 13:45:30 -> 15juni (da-DK) 15.06.2009 13:45:30 -> 15 Juni (id-ID)
R oder r	RFC1123 pattern.	15.06.2009 13:45:30 - > Montag 15. Juni 2009 20:45:30 GMT
s	Sortable date/time pattern.	15.06.2009 13:45:30 -> 2009-06-15T13:45:30
t	Short time pattern.	15.06.2009 13:45:30 -> 1:45 PM (en-US) 15.06.2009 13:45:30 -> 13:45 (hr-HR) 15.06.2009 13:45:30 -> 01:45 م (ar-EG)
T	Long time pattern.	15.06.2009 13:45:30 -> 1:45:30 PM (en-US) 15.06.2009 13:45:30 -> 13:45:30 (hr-HR) 15.06.2009 13:45:30 -> 01:45:30 م (ar-EG)
u	Universal sortable date/time pattern.	15.06.2009 13:45:30 -> 2009-06-15 20:45:30Z
U	Universal full date/time pattern.	15.06.2009 13:45:30 -> Monday, June 15, 2009 8:45:30 PM (en-US) 15.06.2009 13:45:30 -> den 15 juni 2009 20:45:30 (sv-SE) 15.06.2009 13:45:30 -> Δευτέρα, 15 Ιουνίου 2009 8:45:30 μμ (el-GR)
Y oder y	Year/month pattern.	15.06.2009 13:45:30 -> June, 2009 (en-US) 15.06.2009 13:45:30 -> juni 2009 (da-DK) 15.06.2009 13:45:30 -> Juni 2009 (id-ID)

Custom Date and Time Format Strings

A custom format string consists of one or more custom date and time format specifiers. Any string that is not a standard date and time format string is interpreted as a custom date and time format string.

The following table describes the custom date and time format specifiers.

Format specifier	Description	Examples
d	The day of the month, from 1 through 31.	01.06.2009 13:45:30 -> 1 15.06.2009 13:45:30 -> 15
dd	The day of the month, from 01 through 31.	01.06.2009 13:45:30 -> 01 15.06.2009 13:45:30 -> 15
ddd	The abbreviated name of the day of the week.	15.06.2009 13:45:30 -> Mon (en-US) 15.06.2009 13:45:30 -> Пн (ru-RU) 15.06.2009 13:45:30 -> lun. (fr-FR)
dddd	The full name of the day of the week.	15.06.2009 13:45:30 -> Monday (en-US) 15.06.2009 13:45:30 -> понедельник (ru-RU) 15.06.2009 13:45:30 -> lundi (fr-FR)
h	The hour, using a 12-hour clock from 1 to 12.	15.06.2009 01:45:30 -> 1 15.06.2009 13:45:30 -> 1
hh	The hour, using a 12-hour clock from 01 to 12.	15.06.2009 01:45:30 -> 01 15.06.2009 13:45:30 -> 01
H	The hour, using a 24-hour clock from 0 to 23.	15.06.2009 01:45:30 -> 1 15.06.2009 13:45:30 -> 13
HH	The hour, using a 24-hour clock from 00 to 23.	15.06.2009 01:45:30 -> 01 15.06.2009 13:45:30 -> 13
m	The minute, from 0 through 59.	15.06.2009 01:09:30 -> 9 15.06.2009 13:09:30 -> 9
mm	The minute, from 00 through 59.	15.06.2009 01:09:30 -> 09 15.06.2009 13:09:30 -> 09
M	The month, from 1 through 12.	15.06.2009 13:45:30 -> 6
MM	The month, from 01 through 12.	15.06.2009 13:45:30 -> 06
MMM	The abbreviated name of the month.	15.06.2009 13:45:30 -> Jun (en-US) 15.06.2009 13:45:30 -> juin (fr-FR) 15.06.2009 13:45:30 -> Jun (zu-ZA)
MMMM	The full name of the month.	15.06.2009 13:45:30 -> June (en-US) 15.06.2009 13:45:30 -> juni (da-DK) 15.06.2009 13:45:30 -> uJuni (zu-ZA)
s	The second, from 0 through 59.	15.06.2009 13:45:09 -> 9
ss	The second, from 00 through 59.	15.06.2009 13:45:09 -> 09
y	The year, from 0 to 99.	01.01.0001 00:00:00 -> 1 01.01.0900 00:00:00 -> 0 01.01.1900 00:00:00 -> 0 15.06.2009 13:45:30 -> 9
yy	The year, from 00 to 99.	01.01.0001 00:00:00 -> 01 01.01.0900 00:00:00 -> 00 01.01.1900 00:00:00 -> 00 15.06.2009 13:45:30 -> 09
yyy	The year, with a minimum of three digits.	01.01.0001 00:00:00 -> 001 01.01.0900 00:00:00 -> 900 01.01.1900 00:00:00 -> 1900 15.06.2009 13:45:30 -> 2009
yyyy	The year as a four-digit number.	01.01.0001 00:00:00 -> 0001 01.01.0900 00:00:00 -> 0900

		01.01.1900 00:00:00 -> 1900 15.06.2009 13:45:30 -> 2009
yyyyy	The year as a five-digit number.	01.01.0001 00:00:00 -> 00001 15.06.2009 13:45:30 -> 02009
:	The time separator.	15.06.2009 13:45:30 -> : (en-US) 15.06.2009 13:45:30 -> .(it-IT) 15.06.2009 13:45:30 -> : (ja-JP)
/	The date separator.	15.06.2009 13:45:30 -> / (en-US) 15.06.2009 13:45:30 -> - (ar-DZ) 15.06.2009 13:45:30 -> . (tr-TR)
\	The escape character.	15.06.2009 13:45:30 (h \h) -> 1 h
All other characters	Literals. The character is copied to the result string unchanged.	15.06.2009 01:45:30 (arr hh:mm t) -> arr 01:45 A

Text Format Strings

Format string must be a string composed of one or more of the masking elements, as shown in the following table.

Masking Elements	Name	Description	Examples
0	Digit or 0	Replaces the placeholder by an appropriated existing number (0-9); otherwise 0 is indicated in the result string.	abc-123-DEF ("00000") -> 12300 abc-123-DEF ("!00000") -> 00123
9	Digit or space	Replaces the placeholder by an appropriate existing number (0-9); otherwise a space is indicated in the result string.	abc-123-DEF ("!99999") -> 123
#	Digit (optional)	Replaces the placeholder by an appropriate existing number (0-9) or by an appropriate existing plus sign or minus sign, otherwise no number is indicated in the result string.	abc-123-DEF (#####) -> -123
L	Letter	Replaces the placeholder by an appropriate existing letter (a-Z); otherwise a space is indicated in the result string.	abc-123-DEF (LLLLL) -> abc12
?	Letter (optional)	Replaces the placeholder by an appropriate existing letter (a-Z); otherwise no letter is indicated in the result string.	
A	Letter or digit	Replaces the placeholder by an appropriate existing alphanumeric character (0-9, a-Z); otherwise a space is indicated in the result string.	abc-123-DEF (AAAAA) -> abc12
a	Letter or digit (optional)	Replaces the placeholder by an appropriate existing alphanumeric character (0-9, a-Z); otherwise no character is indicated in the result string.	
&	Any character	Replaces the placeholder by an appropriate existing character; otherwise a space is indicated in the result string.	abc-123-DEF (&&&&&) -> abc-1
C	Any character (optional)	Replaces the placeholder by an appropriate existing character; otherwise no character is indicated in the result string.	
<	Convert into small letters	Converts all following characters (a-Z) in small letters.	abcDEF (<LLLLLL) -> abcdef
>	Convert into capital letters	Converts all following characters (a-Z) in capital letters.	abcDEF (>LLLLLL) -> ABCDEF
|	Deactivate conversion	Deactivates a preceding conversion in small letters or capital letters.	abcDEF (>LL<LL|LL) -> ABcdEF
!	Right-aligned output	Indicates the result string right-aligned.	123 (!00000) -> 00123
\	Escape character	The character which follows the escape character is interpreted as literal and not as customized format identifier.	abcDEF (LLL\LLLL) -> abcLDEF
All other characters	Literals		abc (L-L-L) -> a-b-c abc (>L:L:L) -> A:B:C

Country Codes

Country code	Language	
zh-CN	Chinese (People's Republic of China)	中文(中华人民共和国)
zh-Hans	Chinese (simplifiedvereinfacht)	中文(简体)
da	Danish	dansk
de	German	Deutsch
de-DE	German (Germany)	Deutsch (Deutschland)
de-LI	German (Liechtenstein)	Deutsch (Liechtenstein)
de-LU	German (Luxembourg)	Deutsch (Luxemburg)
de-CH	German (Switzerland)	Deutsch (Schweiz)
en	English	English
en-GB	English (Great Britain)	English (United Kingdom)
en-US	English (US)	English (United States)
fi	Finnish	suomi
fr	FrenchFrench	français
fr-BE	French (Belgium)	français (Belgique)
fr-FR	French (France)	français (France)
fr-LU	French (Luxembourg)	français (Luxembourg)
fr-CH	French (Switzerland)	français (Suisse)
el	Greek	ελληνικά
he	Hebrew	עברית
nl	Dutch	Nederlands
nl-BE	Dutch (Belgium)	Nederlands (België)
nl-NL	Dutch (Netherlands)	Nederlands (Nederland)
it	Italian	italiano
it-IT	Italian (Italy)	italiano (Italia)
it-CH	Italien (Switzerland)	italiano (Svizzera)
ja	Jaamese	日本語
pl	Polish	polski
pt-BR	Portuguese (Brazil)	Português (Brasil)
pt-PT	Portuguese (Portugal)	Português (Portugal)
ru	Russian	русский
es	Spanish	español
es-ES	Spanish (Spain)	español (España)
cs	Czech	čeština
tr	Turkish	Türkçe
hu	Hungarian	magyar

For a detailed list of all country codes, click here.

Format Number

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Formats a number.

Syntax

```
$FormatNumber (number, format, [Language=language])
```

Parameters

datetime
Number to be formatted.

format
Indicates how the number is to be formatted. For more information, see Standard Numeric Format String or Custom Numeric Format Strings.

language (optional, as default the language set under windows is used)
Language which is used to format the ouput. For more information, see Country Codes.

Return value

Formatted number

Examples

```
$FormatNumber ("1234.56", "N") -> "1.234,56"
$FormatNumber ("1234.56", "N", Language="en") -> "1,234.56"
$FormatNumber ("1234.56", "N", Language="fr") -> "1 234,56"
```

See also

- Format Value
- Format Text
- Format Date/Time

If..Then..Else Statement

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Insert an If..Then..Else Statement on the label. The If..Then..Else Statements serves for evaluating a condition and depending on the result to further proceed.

Syntax

```
$If (condition, thenValue, elseValue)
```

Parameters

condition
If the condition is **true** or **1**, then *value* is returned otherwise *elseValue*. For more information, see Mathematical Operators.

thenValue
Value which is returned if *condition* is **true** or **1**.

elseValue
Value which is returned if *condition* is **false** or **0**.

Return value

thenValue, if *condition* is **true** or **1**, otherwise *elseValue*.

Examples

	ID	Name	Capital	Area	Population	NativeName	Flag
▶	12	United Kingdom	London	244820	60440000	United Kingdom	

```
$If ($DbField ("Europe", "Area") <= 250000, "*", "**") -> "*"
```

Check whether a database field is empty

```
$If ($Length ($DBField (...)) == 0, "The database field is empty.", "The database field is not empty.")
```


Label Name

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the name of the current label file.

Syntax

```
$LabelName
```

Return value

Name of the current label file

Label Number

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Displays the current label number within the print order on the label.

Syntax

```
$LabelNumber
```

Return value

Label number

Examples

Number of copies = 80
Label number = 5

```
$Format ($LabelNumber, "000") - $Format ($Copies, "000")
```
-> "005 - 080"

Label Path

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Returns the complete path of the current label file.

Syntax

```
$LabelPath
```

Return value

Path name of the current label file

Page Name

Benötigte Programmvariante	BASIC, PROFESSIONAL
Weitere Informationen finden Sie unter Programmvarianten.	

Displays the current page name within a print order on the label.

Syntax

```
$PageName
```

Return value

Page name

Page Number

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Displays the current page number within a print order on the label.

Syntax

```
$PageNumber
```

Return value

Page number

Printer Name

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Displays the current printer name on the label.

Syntax

```
$PrinterName
```

Return value

Printer name

Shift Description

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Displays the current shift description on the label.

Syntax

```
$Shift
```

Return value

Shift definition of printer-specific variable definition.

Examples

Early shift -> 06:00 AM - 01:59 PM
Late shift -> 02:00 PM - 09:59 PM
Night shift -> 10:00 PM - 05:59 AM

System variable (TrueType font)

`$Shift` -> "Early shift" (08:20 AM)
`$Shift` -> "Late shift" (03:30 PM)

Printer variable (Printer font)

`$Shift` -> "=SH()"

See also

- Define Shift Times

Define Shift Times

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

To define the shift times, please proceed as follows:

1. Select the **Label** properties, and than click **Shift Definitions**.
   The **Shift Definitions** dialog box opens.
2. Click on:
   -  to define a new shift.
   - or Del to delete the selected shift.
   -  or double-click the selected shift to change the shift settings.
   -  or Ctrl + ↑ to move the selected shift one position upward.
   -  or Ctrl + ↓ to move the selected shift one position downward.
3. Click **OK** to save the modified settings.

Note

Please note that the individual shift times are not allowed to overlap.

Wrong

Early shift -> 06:00 AM - 02:00 PM
Late shift -> 02:00 PM - 10:00 PM
Night shift -> 10:00 PM - 06:00 AM

Right

Early shift -> 06:00 AM - 01:59 PM
Late shift -> 02:00 PM - 09:59 PM
Night shift -> 10:00 PM - 05:59 AM

User Domain Name

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns the network domain name associated with the current user.

Syntax

```
$UserDomainName
```

Return value

Network domain name

User Name

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Returns the current user name.

Syntax

```
$UserName
```

Return value

User name

Execute VBScript

Required program variant **PROFESSIONAL**

For more information, see Program Variants.

Executes a VBScript.

Syntax

```
$VBScript ([arg1, arg2, arg3, ...])
```

Parameters

arg (optional)
Argument

Return value

Printer Variables

With the help of these variables printer-internal variables can be defined on the label.
In contrast to zu System Variables, printer-internal variables are managed and calculated during the print order by the printer.

Note

- Printer variables can be used only in text boxes with printer fonts and barcodes which are not graphically transferred.
- For each text or barcode field only **one** printer variable can be defined.

Supported Printer Variables

Date/Time	$PrnDateTime	Defines a printer-internal date and time variable.
Field Link	$PrintFieldLink	Defines a printer-internal field link.
Database Field	$PrnDbField	Defines a printer-internal database field.
Counter	$PrnCounter	Defines a printer-internal alphanumeric counter.
Extended Counter	$PrnCounterExt	Defines a printer-internal numeric counter.
User Input	$PrnUserInput	Defines a printer-internal user input.
Check Digit	$PrnCheckDigit	Defines a printer-internal check digit.
Custom Check Digit	$PrnCustomCheckDigit	Defines a printer-internal user-defined check digit.
SAPscript Variable	$SAPField	Inserts a SAPscript variable.

Date/Time

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Defines a printer-internal date and time variable.

Syntax

```
$PrnDateTime (format, [UpdateInterval=updateInterval, MonthOffset=monthOffset, DayOffset=dayOffset,
MinOffset=minOffset, CorrectMonth=correctMonth])
```

Parameters

format
Indicates how the value is to be formatted. For more information, see Printer-specific Date and Time Format String.

updateInterval (optional, Standard = 0)
Indicates how often the variable is to be updated during a print order.
0: At print start
1: After each label

monthOffset (optional, Standard = 0)
Month offset (is added to the current date)

dayOffset (optional, Standard = 0)
Day offset (is added to the current date)

minOffset (optional, Standard = 0)
Minute offset (is added to the current time)

correctMonth (optional, Standard = false)
Month correction
true|1: Retain current month
false|0: Change into next month

Return value

Printer-specific variable definition.

Examples

Current printer settings: 2/25/2014 02:21:25 PM

```
$PrnDateTime ("DD.MO.YYYY") -> "=CL(0;0;0;0;0)<DD.MO.YYYY>" -> 25.02.2014
$PrnDateTime ("HH:MI:SS", UpdateInterval=1, MinOffset=-60) -> "=CL(0;0;1;-60;0)<HH:MI:SS>" -> 13:21:25
```

See also

- Date/Time (System)

Printer-specific Date and Time Format Strings

The format string is used to define the text presentation of a printer-internal date or time value.

The following table describes the date and time format identifiers.

Format specifier	Description	Examples
HH	Hour, from 00 to 23 (24-hours format)	6/15/2009 01:45:30 AM -> 01 6/15/2009 01:45:30 PM -> 13
HE	Hour, from 00 to 23 (12-hours format)	6/15/2009 01:45:30 AM -> 01 6/15/2009 01:45:30 PM -> 01
MI	Minute, from 00 to 59	6/15/2009 01:09:30 AM -> 09 6/15/2009 01:09:30 PM -> 09
SS	Second, from 00 to 59	6/15/2009 01:45:09 PM -> 09
AM, Am or am	AM/PM output	6/15/2009 13:45:09 -> PM (AM) 6/15/2009 13:45:09 -> p.m. (Am) 6/15/2009 13:45:09 -> pm (am)
DD	Day of month, from 01 to 31	6/1/2009 01:45:30 PM -> 01 6/15/2009 01:45:30 PM -> 15
MO	Month, from 01 to 12	6/15/2009 01:45:30 PM -> 06
YYYY	Year (four-digit)	6/15/2009 01:45:30 PM -> 2009
YY	Year, from 00 to 99	6/15/2009 01:45:30 PM -> 09
Y	Year, from 0 to 9	6/15/2009 01:45:30 PM -> 9
WW	Calendar week	6/15/2009 01:45:30 PM -> 25
DW	Day of week, from 0 (Sunday) to 6 (Saturday)	6/15/2009 01:45:30 PM -> 1
DW1	Day of week, from 1 (Sunday) to 7 (Saturday)	6/15/2009 01:45:30 PM -> 2
Dwx	Day of week For x any ASCII character can be used, from this is counted consecutively started with Sunday.	
DOWxxxxxxx	Day of week (variable) For x any ASCII character can be used. The first 'x' stands for Sunday, the next for Monday etc. until Saturday. **Note:** For each weekday a sign must be indicated.	
DOY	Day of year, from 001 to 365	6/15/2009 01:45:30 PM -> 166
DY	Day of year, from 000 to 364	6/15/2009 01:45:30 PM -> 165
\	Escape character	
Any other character	The character is copied to the result string unchanged.	

The following table describes the country-specific date and time formats.

Identifier	Description	Examples
xMO	Shortened name of month	6/15/2009 01:45:30 PM -> JN (CMO) 6/15/2009 01:45:30 PM -> JUN (DMO) 6/15/2009 01:45:30 PM -> GUI (IMO)
xSO	Full name of month	6/15/2009 01:45:30 PM -> June (ESO) 6/15/2009 01:45:30 PM -> Juin (FSO) 6/15/2009 01:45:30 PM -> Junio (SSO)
xSD	Shortened name of weekday	6/15/2009 01:45:30 PM -> MO (GSD) 6/15/2009 01:45:30 PM -> MA (NSD) 6/15/2009 01:45:30 PM -> LUN (SSD)
xLD	Full name of weekday	6/15/2009 01:45:30 PM -> Monday (ELD) 6/15/2009 01:45:30 PM -> Montag (GLD)

		6/15/2009 01:45:30 PM -> Mandag (OLD)

For x the country abbreviation of the desired language can be used.

C = Canadian
D = Danish
E = English
F = French
G = German
I = Italian
N = Dutch
O = Norwegian
S = Spanish
U = Finnish
W = Swedish

Field Link

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Defines a printer-internal field link.

Syntax

```
$PrnFieldLink (value, [value, ...])
```

Parameters

value

The designation of linking elements (field name or text constant). A text constant must be placed into quotation marks ("). The quotation marks are not printed.
Note: For the linking only printer-internal fields can be used.

Return parameter

Printer-specific variable definition.

Examples

ID01 = "12345"
ID02 = "67890"

`$PrnFieldLink (ID01, " - ", ID02)` -> "SC=(0;" - ";1)" -> "12345 - 67890"

See also

- Field Link (System)

Database Field

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Defines a printer-internal database field.

Syntax

```
$PrnDbField (fileName, fieldName, indexColumn, outputColumn, [FieldDelimiter=fieldDelimiter,
HasColumnNames=hasColumnNames, DummyText=dummyText])
```

Parameters

fileName
The file name of the table on the Memory Card which contains the CSV data. More information can be found under How to Create a CSV File and Save CSV File on Memory Card.

fieldName
The name of the field on the label, the content of which should be used for the data set search in *indexColumn*.
Note: Only printer-internal fields should be used for linking.

indexColumn
Name or number of the column which should be used as the index.
Note: Please take care to ensure that capitalisation is taken into account in the column names and the column number has a base of 1.

outputColumn
Name or number of the column which should be output.
Note: Please take care to ensure that capitalisation is taken into account in the column names and the column number has a base of 1.

fieldDelimiter (optional, standard = ";")
Field separator

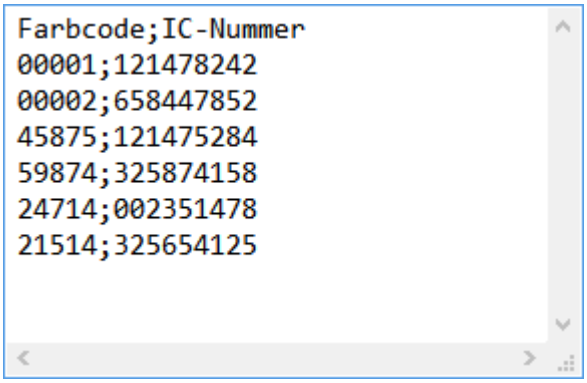
hasColumnNames (optional, standard = true)
true|1: Column names in the first row
false|0: No column names in the first row

dummyText (optional, standard = empty)
Placeholder for the display. By default, outputColumn is shown.

Return value

Printer-specific variable definition.

Examples



```
Farbcode;IC-Nummer
00001;121478242
00002;658447852
45875;121475284
59874;325874158
24714;002351478
21514;325654125
```

`$PrnDbField ("A:\Mappe1.csv", "Text1", "Farbcode", "IC-Nummer")` -> "=MD(FN="A:\Mappe1.csv";SE=';';CH=1;SC="Farbcode",SF="Text1";RC="IC-Nummer")

Printout

Text1 = "00001" -> "121478242"
Text1 = "00002" -> "658447852"
Text1 = "45875" -> "121475284"

See also

- Database Field (System)

How to Create a CSV File

CSV is a simple file format used to store tabular data. CSV stands for "comma-separated values". Its data fields are most often separated, or delimited, by a comma or a semicolon. For example, let's say you had a spreadsheet containing the following data.

Farbcode	IC-Nummer
00001	121478242
00002	658447852
45875	121475284
59874	325874158
24714	002351478
21514	325654125

This data could be represented in a CSV-formatted file as follows:

```
Farbcode;IC-Nummer
00001;121478242
00002;658447852
45875;121475284
59874;325874158
24714;002351478
21514;325654125
```

Here, the fields of data in each row are delimited with a semicolon and individual rows are separated by a newline.

A CSV is a text file, so it can be created and edited using any text editor. More frequently, however, is created by exporting (File Menu -> Export) a spreadsheet or database in the program that created it. Click on a link below for the steps to create a CSV file in Notepad and Microsoft Excel.

Notepad (or any text editor)
Microsoft Excel

See also

- CVS File Format

Notepad (or any text editor)

To create a CSV file with a text editor, first choose your favorite text editor, such as Notepad, and open a new file. Then enter the text data you want the file to contain, separating each field with a semicolon and each row with a new line.

```
Column1;Column2;Column3
one;two;three
example1;example2;example3
```

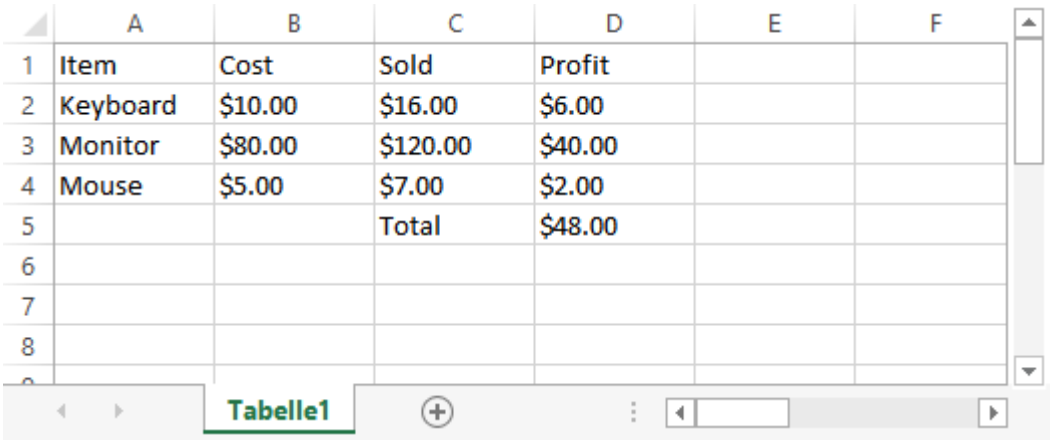
Save this file with the extension **.csv**.

See also

- How to Create a CSV File

Microsoft Excel

To create a CSV file using Microsoft Excel, launch Excel and then open the file you want to save in CSV format. For example, below is the data contained in our example Excel worksheet:



	A	B	C	D	E	F
1	Item	Cost	Sold	Profit		
2	Keyboard	$10.00	$16.00	$6.00		
3	Monitor	$80.00	$120.00	$40.00		
4	Mouse	$5.00	$7.00	$2.00		
5			Total	$48.00		
6						
7						
8						

Tabelle1

Once open, click **File**, choose the **Save As** option, and for the **Save as type** option, select **CSV (Comma delimited)**.

After you save the file, you are free to open it up in a text editor to view it or to edit it manually. Its contents will resemble the following:

```
Item;Cost;Sold;Profit
Keyboard;$10.00;$16.00;$6.00
Monitor;$80.00;$120.00;$40.00
Mouse;$5.00;$7.00;$2.00
;;Total;$48.00
```

Note: You'll notice that the last row begins with two semicolons. This is because the first two fields of that row were empty in our spreadsheet. Don't delete them... the two semicolons are required so that the fields correspond from row to row. They cannot be omitted.

See also

- How to Create a CSV File

CVS File Format

Here are the rules of how data should be formatted in a CSV file, from the IETF's document, RFC 4180. In these examples, "CRLF" is used to represent a carriage return and a linefeed (which together constitute a newline).

- Each record (row of data) is to be located on a separate line, delimited by a line break. For example:

  ```
  aaa;bbb;ccc CRLF
  ```

- There may be an optional header line appearing as the first line of the file with the same format as normal record lines. This header will contain names corresponding to the fields in the file and should contain the same number of fields as the records in the rest of the file. For example:

  ```
  column1;column2;column3 CRLF
  aaa;bbb;ccc CRLF
  zzz;yyy;xxx CRLF
  ```

- Within the header and each record, there may be one or more fields, separated by semicolons. Each line should contain the same number of fields throughout the file. Spaces are considered part of a field and should not be ignored. The last field in the record must not be followed by a semicolon. For example:

  ```
  aaa;bbb;ccc
  ```

See also

- How to Create a CSV File

Save CSV File on Memory Card

In order to save a CSV file on the memory card, proceed as follows:

1. Click on Save File on the Tools tab in the Memory Card group.
2. Select the printer and the source file (e.g. on the hard disk).
3. Enter the pathnames for the target file on the memory card.
4. Click **OK** in order to save the file on the memory card.

Sample

This example shows how you can define a database label using the $PrnDbField variable.

The example can be found in the following directory: *%InstallDir%\Samples\Memory Card*.

Data.csv This file contains a list of the data which should be shown.
Label.lbex Label definition

```
Farbcode;IC-Nummer
00001;121478242
00002;658447852
45875;121475284
59874;325874158
24714;002351478
21514;325654125
```

In order to execute the example, proceed as follows:

1. Save CSV File on Memory Card.
   In the example, the file is saved under *U:\Standard\Data.csv*.
2. Open the label **Label.lbex**.

3. If you have saved the CSV file under a different file name on the memory card, double click on "xxxxx" and change the path name.
   `$PrnDbField ("U:\Standard\Data.csv", "FCODE", "Farbcode", "IC-Nummer", DummyText="xxxxx")`
4. Select a label printer an click **Print**.
5. Enter the color code at the printer.

Counter

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Defines a printer-internal alphanumeric counter.

Syntax

```
$PrnCounter (value, [Prompt=prompt, UpdateInterval=updateInterval, Increment=increment, Pos=pos,
Radix=radix, Mode=mode, ResetTime=resetTime, ResetValue=resetValue])
```

Parameters

value
Current start value.
Note: The number of characters specifies the output format (max. 9 characters).

prompt (optional, standard = empty)
If a prompt text is defined, the start value is queried at print start.

updateInterval (optional, standard = 1)
Indicates how often the variable is to be updated during a print order.
1: After each label
n: After n labels

increment (optional, standard = 1)
Increment.

pos (optional, standard = 0)
Defines the position at which the counter starts to count. If the position is equal 0, the number of character in *value* is used as start position.

radix (optional, standard = 10)
Radix, basis of the counter (1-36)
1: Alphabetical (A-Z)
2: Binary (0, 1)
8: Octal (0-7)
10: Decimal (0-9)
16: Hexadecimal (0-9, A-F)
36: Alphanumeric (0-9, A-Z)

mode (optional, standard = 1)
Operating mode
0: Reset start value manually
1: Reset start value manually (automatic rollover)
2: Enter start value at printer
3: Enter start value (= last end value) at printer
4: Reset start value at cycle end
5: Reset start value by I/O signal
6: Reset start value time-controlled
7: Reset start value time-controlled (enter start value at printer)

resetTime (optional, for operating mode 6 and 7 only, standard = empty)
Time to which the start value is to be reset.
Format: "HH:MM"

resetValue (optional, for operating mode 6 and 7 only, standard = empty)
Value to which the start value is to be reset. If no value is indicated, the counter is reset to its original start value.

Return value

Printer specific variable definition.

Examples

```
$PrnCounter ("0001", Mode=1) -> "=CN(10;1;4;+1;1)0001"
$PrnCounter ("1234", Mode=7, ResetTime="06:00", ResetValue="0001") -> "=CN(10;7;4;+1;1;06:00;0001)1234"
```

See also

- Extended Counter (Printer)
- Counter (System)

Extended Counter

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Defines a printer-internal numeric counter.

Syntax

```
$PrnCounterExt (value, [Prompt=prompt, UpdateInterval=updateInterval, Increment=increment,
MinValue=minValue, MaxValue=maxValue, TrimLeft=trimLeft, Mode=mode])
```

Parameters

value
Current start value.
Note: The number of digits specifies the output format (maximum "999999999").

prompt (optional, Standard = empty)
If a prompt text is defind, the start value is queried at print start.

updateInterval (optional, Standard = 1)
Indicates how often the variable is to be updated during a print order.
1: After each label
n: After n labels

increment (optional, Standard = 1)
Increment.

minValue (optional, Standard = "0")
Minimum value.

maxValue (optional, Standard = empty)
Maximum value. If no *maxValue* is indicated, as default the number of start value digits is used to calculate the maximum value.

Start value	Calculated maximum value
0001	9999
01	99

trimLeft (optional, Standard = false)
true|1: Enable leading zeros at output
false|0: Show leading zeros at output

mode (optional, Standard = 5)
Operating mode
0: Reset start value manually
1: Reset start value manually (automatic rollover)
2: Enter start value at printer
3: Enter start value (= last end value) at printer
4: Reset start value at cycle end
5: Reset start value manually (to min/max)
6: Reset start value manually (to start value)
7: Reset start value manually (stop printing)

Return value

Printer-specific variable definition.

Examples

```
$PrnCounterExt ("0050", Mode=5, Increment=1, UpdateInterval=1, MinValue="1", MaxValue="999",
TrimLeft=true) -> "=CC(+1,1,5,0,1,999)0050" -> 50, 51, ... 999, 1, 2, ...
```

See also

- Counter (Printer)
- Counter (System)

User Input

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Defines a printer-internal user input.

Syntax

```
$PrnUserInput (prompt, startText, [StartPos=startPos, AllowableChars=allowableChars,
SkipSpecialChars=skipSpecialChars, PrintAlignment=printAlignment, InputAlignment=inputAlignment,
InputMask=inputMask])
```

Parameters

prompt
The prompt text shown in the first line of the printer display.

startText
The input text shown in the second line of printer display.

startPos (optional, Standard = 0)
Start position for the input. If start position is 0 then the *startText* length is used as start position.

allowableChars (optional, Standard = Alphanumeric)
Indicates which characters are allowed for the input.
Numeric: Numeric
Alphanumeric: Alphanumeric
InputMask: Input mask

skipSpecialChars (optional, Standard = false)
Indicates whether special characters are to be retained at the input or not.
true|1: Skip special characters
false|0: Do not skip special characters

printAlignment (optional, Standard = Right)
Print alignment
Right: Right-aligned
Left: Left-aligned

inputAlignment (optional, Standard = Left)
Input alignment
Right: Right-aligned
Left: Left-aligned

inputMask (optional, Standard = empty)
Input mask. The input mask must be a string composed of one or more of the masking elements, as shown in the following table.

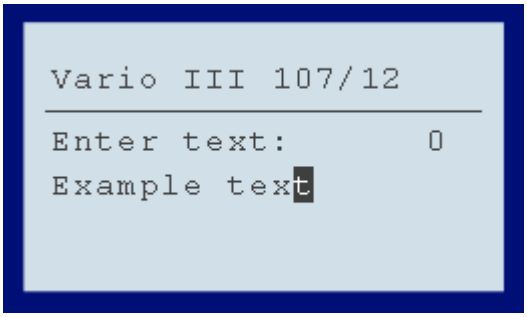
Masking Element	Description
9	Digit
#	Digit/+/-
?	Letter
a	Letter or digit
C	Any character

Return value

Printer-specific variable definition.

Examples

`$PrnUserInput ("Enter text:", "Example text", StartPos=0, AllowableChars=1)` -> "=UG(12;1;0;0;0;"Enter text:")<Example text>"



See also

- User Input (System)

Check Digit

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

Defines a printer-internal check digit.

Syntax

```
$PrnCheckDigit (data, checkDigitMethod)
```

Parameters

data

Data (field name or text constant) for which the check digit is to be calculated. A text constant must be enclosed in quotes ("). The quotation marks will not be printed.
Note: For the linking only printer-internal fields may be used.

checkDigitMethod

Method after according to which the check digit is to be calculated.

Method	Description
MOD10	Modulo 10
MOD11	Modulo 11
MOD43	Modulo 43
MOD47_15	Modulo 47 (weighting 15)
MOD47_20	Modulo 47 (weighting 20)
MOD103	Module 103

Return value

Printer-specific variable definition.

Examples

```
$PrnCheckDigit (ID01, MOD10) -> "CD=(0;0;0)"
$PrnCheckDigit ("123456789012", MOD10) -> "=CD("123456789012";0;0)"
```

See also

- Check Digit (System)

Custom Check Digit

Required program variant **PROFESSIONAL**

For more information, see Program Variants.

Defines a printer-internal user-defined check digit.

Syntax

```
$PrnCustomCheckDigit (data)
```

Parameters

data

Data (field name or text constant) for which the check digit is to be calculated. A text constant must be enclosed in quotes ("). The quotation marks will not be printed.
Note: For the linking only printer-internal fields may be used.

Return value

Printer-specific variable definition.

Examples

ID01 = "12345"

```
$PrnCustomCheckDigit (ID01, MOD10) -> "=CD(0;0;0)"
$PrnCustomCheckDigit ("123456789012", MOD10) -> "=CD("123456789012";0;0)"
```

See also

- Check Digit (System)
- Custom Check Digit (System)
- Check Digit (Printer)

SAPscript Variable

Required program variant **PROFESSIONAL**

For more information, see Program Variants.

Inserts a SAPscript variable.

Syntax

```
$SAPField (fieldName, [dummyText])
```

Parameters

fieldName

SAPscript field name such as "VBAK-KUNNR".
Note: Do not insert '&' delimiter. **Labelstar Office** will automatically add the '&' delimiter to both ends of the field name when the variable is exported.

dummyText (optional, Standard = empty)

Indicates whether another value for the screen display is to be used.

Return value

Printer-specific variable definition.

Examples

```
$SAPField ("VBAK-KUNNR") -> "&VBAK-NUNNR&"
$SAPField ("VBAK-KUNNR", "12345") -> "12345"
```

See also

- Printing in an SAP Environment

Custom Variables

Required program variant **BASIC, PROFESSIONAL**

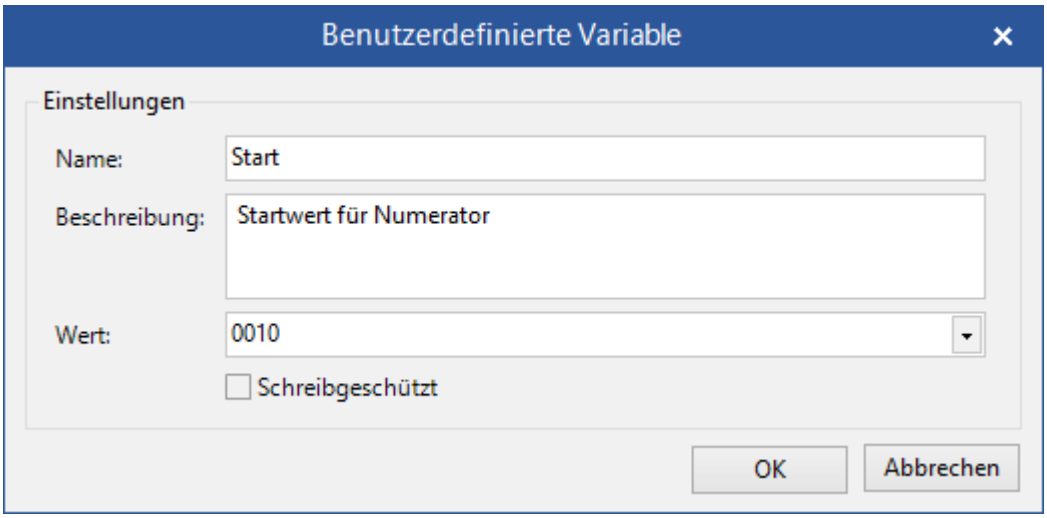
For more information, see Program Variants.

Labelstar Office supports user-defined variables.

Custom variables can be used in many ways in **Labelstar Office**. For example to define the file path of a data connection or to define a global counter.

In order to manage user-defined variables, proceed as follows:

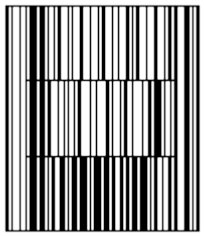
1. Activate the **Variables** view and open **Custom Variables**.
2. **Create a new variable**
   - Click on **New Variable**.
     The **Custom Variable** dialog box opens.



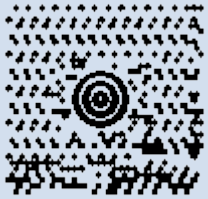
3. **Edit variable**
   In order to edit a variable, perform one of the following actions:
   - Right-click on the variable and select the **Edit** context menu option.
   - Double-click on the variable which you want to edit.
4. **Delete variable**
   In order to delete a variable, perform one of the following actions:
   - Right-click on the variable and select the **Delete** context menu option.
   - Select the variable which you want to delete and press Del.

Bar Codes

Following is a list of supported bar code types:

Code	Beispiel	Beschreibung
Aztec Code		2D bar code, developed by Welch Allyn.
Aztec Runes		2D bar code based on the Aztec Code.
Codabar		Numeric bar code, encoded digits and special characters.
Codablock F		2D bar code based on the Code 128.
Code 128		Alphanumeric bar code, encoded ASCII character set.
Code 128 (Subset A)		Alphanumeric bar code, encoded digits, capital letters and special characters.
Code 128 (Subset B)		Alphanumeric bar code, encoded digits, letters and special characters.
Code 2 of 5 Industrial		Numeric bar code.
Code 2 of 5 Interleaved		Numeric bar code, with an even number of digits.

Code 39		Alphanumeric bar code, encoded digits, capital letters, special characters and spaces.
Code 39 Extended		Alphanumeric bar code based on Code 39, encoded ASCII character set.
Code 93		Alphanumeric bar code, encoded digits, capital letters, special characters and spaces.
Code 93 Extended		Alphanumeric bar code based on Code 93, encoded ASCII character set. **Note:** This barcode is transmitted graphically.
DataMatrix		2D bar code, developed by Acuity Corp.
Deutsche Post Identcode		Numeric bar code based on Code 2/5 Interleaved with different check digit calculation.
Deutsche Post Leitcode		Numeric bar code based on Code 2/5 Interleaved with different check digit calculation.
EAN-13, GTIN-13		Numeric bar code.
EAN-13 + 2 Digits		EAN-13 with 2-digit add on.
EAN-13 + 5 Digits		EAN-13 with 5-digit add on.

EAN-8, GTIN-8		Numeric bar code.
GS1-128		Alphanumeric bar code.
GS1 DataBar		Alphanumeric bar code.
GS1 DataMatrix		2D bar code.
HIBC-128		Alphamumeric bar code based on Code 128.
HIBC-39		Alphamumeric bar code based on Code 39.
HIBC DataMatrix		2D bar code.
HIBC QR Code		2D bar code.
ITF-14, SCC-14		Numeric bar code based on Code 2/5 Interleaved.
MaxiCode		2D bar code.
PDF417		2D bar code.

Pharmacode		Numeric bar code.
PZN		Numeric bar code based on Code 39.
QR Code		2D bar code.
UPC-A, GTIN-12		Numeric bar code.
UPC-E		Numeric bar code.

1D Bar Codes

Linear (one dimensional) bar codes consist of a single row of parallel bars and spaces of varying widths that represent data. The bars and spaces are arranged in a predetermined pattern defined by the symbology.

Supported Bar Codes

- Codabar
- Code 128
  - Code 128 (Subset A)
  - Code 128 (Subset B)
- Code 2 of 5 Industrial
- Code 2 of 5 Interleaved
- Code 39
- Code 39 (Full ASCII)
- Code 93
- Code 93 (Full ASCII)
- Deutsche Post Identcode
- Deutsche Post Leitcode
- EAN-13
  - EAN-13 + 2 Digits
  - EAN-13 + 5 Digits
- EAN-8
- GTIN-8
- GTIN-12
- GTIN-13
- ITF-14
- Pharmacode
- PZN
- SCC-14
- UPC-A
- UPC-E

Codabar

The **Codabar** is mainly used in libraries, in photo sector and in medical ranges (blood banks). The **Codabar** is a universal, numeric bar code that contains 6 special characters additionally to the numbers 0 to 9. The number of representable signs is not given of the code.

Additionally four different start/stop signs (A-D) are defined, i.e. each code must begin and end with A, B, C or D. However, the start/stop signs cannot be used in the bar code itself.

Each sign of the code consists of elf units, four bars and three spaces. A fourth gap is always narrow.

Length	Variable
Zeichensatz	Digits 0-9 Special characters - $: / . +
Prüfziffer	Optional Modulo 16

Code 128

Code 128 is a universal, alphanumeric bar code mainly used in shipping/ transport, on documents of identification and in warehousing/distribution.

Code 128 can encode the complete ASCII character set. This code uses an internal check digit that won't be displayed in the text line under the code. By the use of four different widths for bars and gaps. the information density is very high.

The structure of a **Code 128** consists of a start sign, data area, check digit and a stop sign. Before the start sign and behind the stop sign a white zone (quiet zone) with a width of at least 10 modules must be defined.

Length	Variable
Zeichensatz	ASCII character set including control characters
Prüfziffer	Modulo 103

See also

- Code 128 (Subset A)
- Code 128 (Subset B)
- GS1-128

Code 128 (Subset A)

Special type of Code 128.

Length	Variable
Zeichensatz	Digits 0-9 Upper case letters A-Z Control characters
Prüfziffer	Modulo 103

See also

- Code 128
- Code 128 (Subset B)

Code 128 (Subset B)

Sepcial type of Code 128.

Length	Variable
Zeichensatz	Digits 0-9 Upper and lower case letters A-z
Prüfziffer	Modulo 103

See also

- Code 128
- Code 128 (Subset A)

Code 2 of 5 Industrial

The **Code 2 of 5 Industrial** is a very simple numeric code which is able to display digits from 0 to 9. The code is mainly used in industrial sector and particularly in transport and warehousing. **Code 2 of 5** has no built in check digit.

As the information density of the bar code is low and its space consumption very high, it is barely used nowadays.

The bar code has its name because each number is coded in 5 bars, two broad bars and three narrow bars. The spaces between the bars not contain any information.

Length	Variable
Zeichensatz	Digits 0-9
Prüfziffer	Optional Modulo 10 Modulo 10 (Luhn Algorithm)

Siehe auch

- Code 2 of 5 Interleaved

Code 2 of 5 Interleaved

Code 2 of 5 Interleaved is a special type of Code 2 of 5 Industrial that is also a numeric code able to display digits from 0 to 9. The advantage of **Code 2 of 5 Interleaved** is that the code uses self-checking and it is very compact so it does not need much space like the simple Code 2 of 5 Industrial.

Code 2 of 5 Interleaved is only valid if there is a even number of digits. To display an odd number of digits you have to add a zero to the beginning (123 becomes 0123) or you may use your own check digit.

Length	Variable (even number of digits)
Zeichensatz	Digits 0-9
Prüfziffer	Optional Modulo 10 Modulo 10 (Luhn Algorithm)

See also

- Code 2 of 5 Industrial

Code 39

Code 39 is an alphanumeric bar code mainly used in shipping/ transport, electronics and chemical industries, the health sector and and in warehousing/distribution.

Each character is composed of nine elements: five bars and four spaces. Three of the nine elements in each character are wide (binary value 1), and six elements are narrow (binary value 0). The width ratio between narrow and wide is not critical, and may be chosen between 1:2 and 1:3. The bar code itself does not contain a check digit, but it can be considered self-checking on the grounds that a single erroneously interpreted bar cannot generate another valid character.

Possibly the most serious drawback of **Code 39** is its low data density: It requires more space to encode data in **Code 39** than, for example, in Code 128.

Length	Variable
Zeichensatz	Digits 0-9 Upper case letters A-Z Special characters - . $ / + % Space
Prüfziffer	Optional Modulo 43 Modulo 11 (weighting 7) Modulo 10 (Luhn Algorithm)

See also

- Code 39 Extended

Code 39 Extended

Code 39 Extended is an extended version of Code 39 that can encode the complete ASCII character set.

The additional characters (e.g. lower case letters) are created using the existing characters of Code 39 by combining two characters each.

Length	Variable
Zeichensatz	ASCII character set
Prüfziffer	Optional Modulo 43 Modulo 11 (weighting 7) Modulo 10 (Luhn Algorithm)

See also

- Code 39

Code 93

Code 93 is an alphanumeric code similar to Code 39 and can encode 48 different characters.
By the use of various bar widths and gap widths it has a higher information density.
Each sign of the code consists of nine units, three bars and three spaces.

Length	Variable
Zeichensatz	Digits 0-9 Upper case letters A-Z Special characters - . $ / + % Space
Prüfziffer	Modulo 47

See also

- Code 93 Extended

Code 93 Extended

Note

This barcode is transmitted graphically.

Code 93 Extended is an extended version of Code 93 that can encode the complete ASCII character set.

Length	Variable
Zeichensatz	ASCII character set
Prüfziffer	Modulo 47

See also

- Code 93

Deutsche Post Identcode

The **Identcode** is a variant of Code 2 of 5 Interleaved, but with a different check digit. This code is used by the Deutsche Post AG (DHL) and serves the automatic distribution of freight parcels in the post-office centres.

Structure of the **Identcode**:

- **1..2:** Mail center (outgoing)
- **3..5:** Customer code
- **6..11:** Delivery number
- **12:** Check digit

Length	12
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

See also

- Deutsche Post Leitcode

Deutsche Post Leitcode

The **Leitcode** is a variant of Code 2 of 5 Interleaved, but with a different check digit. This code is used by the Deutsche Post AG (DHL) and serves the automatic distribution of freight parcels in the post-office centres.

Structure of the **Leitcode**:

- **1..5:** ZIP code
- **6..8:** Street's code number
- **9..11:** House number
- **12..13:** Product code
- **14:** Check digit

Length	14
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

See also

- Deutsche Post Identcode

EAN-13, GTIN-13

EAN-13 is used world-wide for marking retail goods. Each packaging is uniquely identified by the GTIN - Global Trade Item Number (formerly European Article Number - EAN).

The symbol encodes 13 characters: the first two or three are a country code which identify the country in which the manufacturer is registered (not necessarily where the product is actually made). The country code is followed by 9 or 10 data digits (depending on the length of the country code) and a single check digit.

2-digit and 5-digit supplemental barcodes may be added for a total of 14 or 17 data digits.

Length	13
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

See also

- EAN-13 + 2 Digits
- EAN-13 + 5 Digits
- EAN-8, GTIN-8

EAN-13 + 2 Digits

EAN 13 with two additional characters.

Length	15
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

See also

- EAN 13, GTIN-13
- EAN 13 + 5 Digits

EAN-13 + 5 Digits

EAN 13 with five additional characters.

Length	18
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

See also

- EAN 13, GTIN-13
- EAN 13 + 2 Digits

EAN-8, GTIN-8

EAN-8 is a shortened version of the EAN-13 code.

It includes a 2 or 3 digit country code, 4 or 5 data digits (depending on the length of the country code), and a check digit.

A GTIN short number will only be output upon request since these numbers have limited availability. The **EAN-8 bar codes** with a 2 as the starting digit can be used freely within the company, but they are not unique in the world.

Length	8
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

See also

- EAN-13, GTIN-13

ITF-14, SCC-14

The **ITF-14**, which is based on Code 2 of 5 Interleaved, is used to create the Shipping Container Code (SSC). This code is used to mark cartons and palettes that are including goods with an EAN-13 code.

A **SCC-14** number contains the following information:

- **1:** Package indicator
- **2..3:** UPC numbering system/GS1 country prefix
- **4..8:** GS1 company prefix
- **9..13:** Item identification number
- **14:** Check digit

Length	14
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

Pharmacode

The **Pharmacode** is a simple, numeric bar code placed on the marked from company Laetus. It is used in pharmaceutical industry for the control of packaging means and/or for the control of packaging machines.

The **Pharmacode** applied on the packaging and on the package insert provides that the correct package insert is sorted into the appropriate packaging. With the **Pharmacode** only integers can be coded from 3 to 131070.

Length	Variable
Zeichensatz	Digits 0-9
Prüfziffer	None

PZN

The **PZN** (Pharmazentralnummer) serves for marking of drugs and other pharmacy products according to trademarks, dosage form, intensity and package size.

The **PZN** is assigned by the Informationsstelle für Arzneispezialitäten (IFA).

The **PZN-8** replaces the old **PZN-7** from the 01.01.2013. You will be able to convert old **PZN-7** code to **PZN-8** by just adding a leading zero.

Length	7-8
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 11

UPC-A, GTIN-12

The Universal Product Code (UPC) is a bar code symbology that is widely used in the United States, Canada, the United Kingdom, Australia, New Zealand and in other countries for tracking trade items in stores.

Its most common form, the **UPC-A**, consists of 12 numerical digits, which are uniquely assigned to each trade item. Along with the related EAN-13 bar code, the **UPC-A** is the bar code mainly used for scanning of trade items at the point of sale.

The symbol encodes 12 characters:

- **1:** System identification
- **2..6:** UPC ID number (manufacturer)
- **7..11:** Individual article number (issued by the manufacturer)
- **12:** Check digit

Length	12
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

Siehe auch

- UPC-E

UPC-E

The **UPC-E** is intended to be used on packaging which would be otherwise too small to use one of the other versions. The code is smaller because it drops out zeros which would otherwise occur in the symbol. For example, the code 59300-00066 would be encoded as 593663. The last digit (3 in the example) indicates the type of compression.

Length	8
Zeichensatz	Digits 0-9
Prüfziffer	Modulo 10

See also

- UPC-A

2D Bar Codes

Most 2D bar codes consist of small black and white squares and encode information in the area. A distinction is made between stacked bar codes, matrix codes, item codes and other special shapes.

Supported Bar Codes

- Aztec Code
- Aztec Runes
- Codablock F
- DataMatrix
- MaxiCode
- PDF417
- QR Code

Aztec Code

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Aztec Code is a 2D matrix bar code which is built on a square grid with a bulls-eye pattern at its centre for locating the code. In the concentric square rings around the bulls-eye pattern data is encoded. **Aztec Code** is used mainly in transportation e.g. for online tickets of Deutsche Bahn.

Very small (starting from 12 characters) and large data volumes (up to 3067 alphanumeric characters) can be coded.

Aztec Code consists of three fix and two variable components. The fix components are: the central Finder Pattern, Orientation Pattern and Reference Grid. Mode Message and Data Layers are the variable components of the code.

The **Aztec Code** is one of the few bar codes which do not need a quiet zone. Thanks to the Reed-Solomon error correction the reconstruction of data contents is still possible even if the code (25% with large codes and 40% with small codes) was destroyed. The so-called Core Symbol of the **Aztec Code** contains the central Finder Pattern, Orientation Pattern and Mode Message.

Length	3067 alphanumeric characters 3832 numeric characters
Zeichensatz	ASCII character set
Prüfziffer	Internal

See also

- Aztec Runes

Aztec Runes

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Aztec Runes are a set of small barcode symbols that are used for special applications.

Length	3
Zeichensatz	An integer between 0 and 255 (including the boundaries).
Prüfziffer	Internal

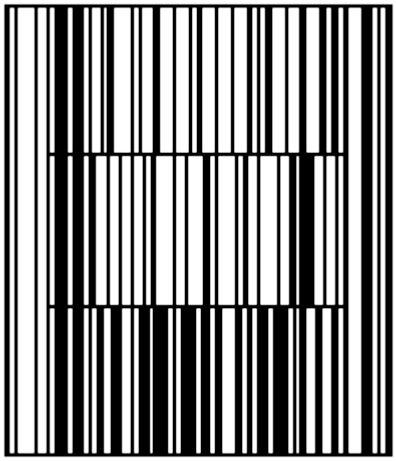
See also

- Aztec Code

Codablock F

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.



Codablock F is a 2D bar code with several stacked Code 128 one above the other. The code is mainly used in health care.

The lines of **Codablock F** are marked by line numbers exactly as the total character number of the code. With **Codablock F** 2 to 44 lines can be displayed. Each individual line can have up to 62 characters according to Code 128.

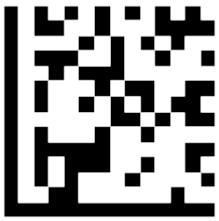
The principle of Codablock bar code is the same like the line break of a text editor - if the line is full it is wrapped to the next line, i.e. to each line the line number is added and to the finished block the number of lines. For the orientation of the reading device, each line contains a line indicator and additionally two check digits to secure the contents of the total message.

Length	In 2 to 44 lines each, 4 to 62 characters (max 2725 characters) are encoded.
Zeichensatz	ASCII character set
Prüfziffer	Internal

DataMatrix

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.



DataMatrix code is one of the most popular 2D bar codes. Many data can be coded onto a small surface. For this reason it is often used for permanent marking with lasers in production (e.g. circuit boards). Additionally the code is used in the automotive sector, with analyzers and instruments (chemistry, medicine) and also increasingly as printed code image in documentation (e.g. tickets, digital postmarks).

DataMatrix symbols consist of square modules arranged within black (active ->binary 1) and white (inactive -> binary 0) cells.

Finder pattern as area of **DataMatrix** code has the width of a module and consists at the left and upper sides of two lines, which contain only dark modules. The lines at the right and upper side of the symbol are represented alternating in dark and bright modules. A quiet zone with the width of one module surrounds the barcode.

The uniform symbol size and the firm symbol distance make reading and decoding of the code very safe. A **DataMatrix** barcode symbol consists of the following four components:

- **Data area:** This area contains redundant data in codified form for data protection.
- **Closed limitation line (finder pattern):** This is the corner that is represented in normal alignment to the left and below data area with an uninterrupted line. This boundary is used for erection and rectification of the code in order to permit each reading angle.
- **Open borderline (alternating pattern):** This represents the opposite corner of the 'closed limitation line'. These lines are on top and right sides and consist of white and black dots (open lines). These are used to the determination of lines and columns while scanning.
- **Quiet zone:** Zone around the code containing no information or pattern. This area must be at least so wide as one column/line res. one dot of the code.

For the creation of **DataMatrix** code the Reed-Solomon error correction code ECC 200 is used. With this error correction code a **DataMatrix** barcode is still readable even if up to 25% of the code is covered or destroyed.

Length	2335 alphanumeric characters 3116 numeric characters
Zeichensatz	ASCII character set
Prüfziffer	Internal

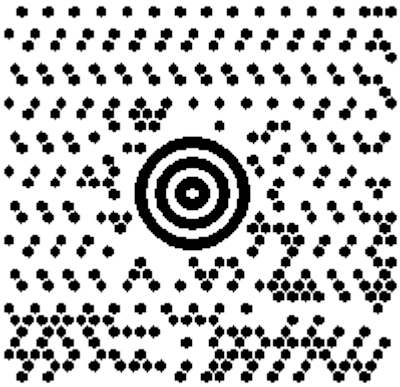
See also

- GS1 DataMatrix

MaxiCode

Required program variant	BASIC, PROFESSIONAL

For more information, see Program Variants.



MaxiCode is a 2D bar code with a fixed size of 1 in. x 1 in. (approx. 25,4 mm x 25,4 mm). In this area of 1 square in. (approx. 645 mm^2) data can be coded. The code is build of 884 hexagonal modules that show finder pattern.

MaxiCode is a machine-readable symbol system created and used by United Parcel Service. The code is suitable for fast identification, tracking and managing the shipment of packages and contains the UPS control number, weight, kind of dispatch and address.

The code is easily identifiable at the bull's-eye pattern in the middle of the symbol. By the Reed-Solomon error correction a reconstruction of the 2D bar code is still possible even if up to 25% of the code were destroyed.

MaxiCode defines 6 modes that determines that how data should be interpreted. The mode 0 and 1 are no longer used. Mode 4 and 5 are used to encode "raw data" with mode 5 offers a slight higher data error correction. Mode 2 and 3 are used to encode “structure message” which comprises two parts: Primary Message and Secondary Message. The Primary Message encodes a postal code, 3-digit country code and 3-digit class of service code. The Second Message encodes other data.

Labelstar Office supports the following modes:

- **Mode 2:** Indicates the symbol contains a Structured Carrier Message with a numeric postal code (up to 9 digits).
- **Mode 3:** Indicates the symbol contains a Structured Carrier Message with an alphanumeric postal code (up to 6 characters).
- **Mode 4:** Indicates the symbols contains general-purpose data, protected by the standard error correction.

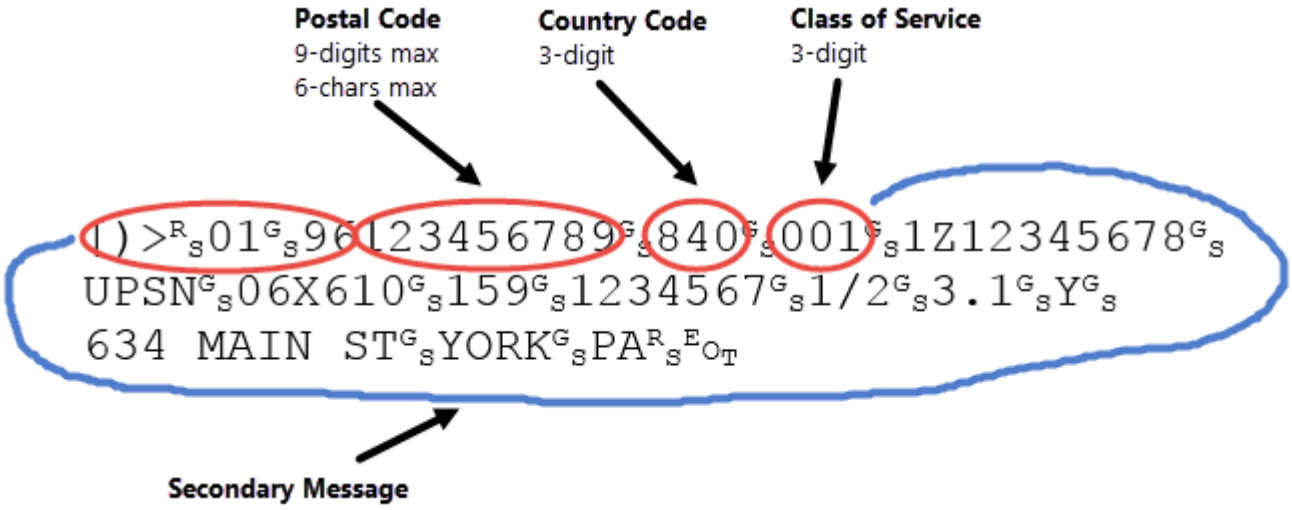
Length	93 alphanumeric characters 138 numeric characters
Zeichensatz	ASCII character set
Prüfziffer	Internal

Structured Carrier Message

UPS has specific requirements of their customers for the content of **MaxiCode** symbols.

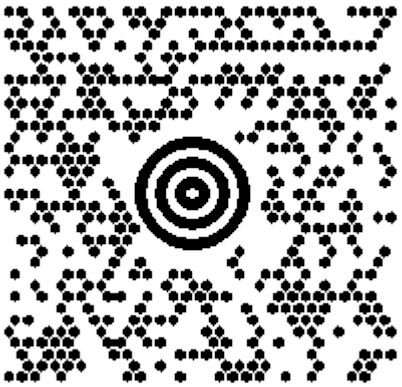
The message format that UPS uses in **MaxiCode** symbols conform to the ANSI MH10.8M-1993 standard. ANSI MH10.8M-1993 is an American National Standard for barcodes on unit loads and transport packages. This standard sets down guidelines on how package information can be coded so that it can be consistently and reliably exchanged between organizations. This record format is commonly referred to as a Structured Carrier Message.

Message Structure



Field	Description	Size and Type	Required	Example
Primary Message				
Postal Code		5 or 9 digits in the USA (Mode 2), up to 6 alphanumeric charcters in other countries (Mode 3).	Yes	123456789
Country Code	Country code encoded per ISO 3166. Mode 2 supports the US Country Code (840). For other country codes please use Mode 3 instead.	3 numeric digits	Yes	840
Class of Service	Assigned by the carrier	3 numeric digits	Yes	001
Secundary Message				
Tracking Number		10 or 11 alphanumeric characters	Yes	1Z12345678
Standard Carrier Number		"UPSN"	Yes	UPSN
Shipper Number		6 alphanumeric characters	-	06x610
Julian Day of Pickup		3 numeric digits	-	156
Shipment ID Number		0-30 alphanumeric characters	-	1234567
Package n/x	Package n of x total packages.	1-3 numeric digits"/"1-3 numeric digits	-	1/2
Package Weight (lb.)	Round up to the next pound.	1-3 numeric digits	-	3
Address Validation		"Y" or "N"	-	Y
Ship-To Address		0-35 alphanumeric characters	-	634 MAIN ST
Ship-To City		0-20 alphanumeric characters	-	YORK
Ship-To State		2 alpha characters	-	PA

Example code



PDF417

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

The **PDF417** is a stacked linear bar code based on a rectangular field. PDF stands for Portable Data File. It is used in a variety of applications, primarily identification cards, transport, automobile industry, inventory management and in administrative authority, e.g. Agentur für Arbeit to prevent manipulation at questionnaires.

The bar code symbol consists of 3 to 90 lines and 1 to 30 columns. Each line has a left and a right Quiet Zone, a Start/ Stop Patterns, a left and a right indicator and 1 to 30 Symbol Characters. A **PDF417** symbol is formed of bar code data, check digit and correction sign. The used characters are coded in code words. A code word consists of 17 modules that are formed of 4 bars and spaces.

The error correction is determined with the Reed Solomon algorithm in 9 selectable Error Correction Levels. With selected error correction level 0 an error can be recognized but not corrected. With the error correction levels 1 to 8 errors can also be corrected.

Use of the error correction:

- **ECL 2:** less than 41 code words
- **ECL 3:** 41 to 160 code words
- **ECL 4:** 161 to 320 code words
- **ECL 5:** more than 320 code words

Length	1850 alphanumeric characters 2725 numeric characters
Zeichensatz	ASCII character set
Prüfziffer	Internal

QR Code

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

A **QR Code** (quick response code) is a type of 2D bar code that is used to provide easy access to information through a smartphone.

In this process, known as mobile tagging, the smartphone's owner points the phone at a **QR Code** and opens a barcode reader app which works in conjunction with the phone's camera. The reader interprets the code, which typically contains a call to action such as an invitation to download a mobile application, a link to view a video or an SMS message inviting the viewer to respond to a poll. The phone's owner can choose to act upon the call to action or click cancel and ignore the invitation.

Length	4296 alphanumeric characters 7089 numeric characters
Zeichensatz	ASCII character set
Prüfziffer	Internal

What are the different types of QR Codes?

QR Codes can trigger various actions on the smartphone where they are read. Directing a user to a website isn't the only possible action and some of them are worth knowing (such as saving a business card or connecting to wireless networks).

With **Labelstar Office** you can create the following types of QR Codes:

- **Plain text:** This is the simplest QR Code type. A raw text is encoded and will be displayed on the screen after scanning. You can write anything you like.
- **Business card:** With these business card QR Codes, a contact card with the details you entered will be automatically stored into the contact list of the smartphone. You can enter your names, address, phone number, email and so on.
- **Add an event to a calendar:** After scanning these QR Codes, you will be asked if you want to save the event in your smartphone's calendar. By adding the event to your calendar, you will be reminded of the correct date.
- **Website:** By scanning this type of QR Codes, users will be directed to a webpage and will discover the content available. This is the most common QR Code type.
- **Call a phone number:** Type in a phone number when you create the QR Code. When scanning, users will be proposed to call the phone number.
- **Send an SMS:** Save the content and the recipient's phone number of an SMS. After scanning, you will only have to confirm before sending it.
- **Send an email:** This works exactly like the SMS QR Code type. Only this time, you enter the email content, the subject and the recipients to enable sending after scanning.
- **Geo location:** Geographic co-ordinates are stored and when scanned will redirect to a static mobile google map of your location.
- **Wifi access information:** Whoever scans the code will be able to access your Wi-fi.

GS1 Bar Codes

GS1 (Global Standard One) is a worldwide system, which facilitates unmistakable identification. Bar coded GS1 identifiers for automated processing clearly label products/items, logistics units, reusable packaging/containers etc., as they are unique. Scanners then read the GS1 symbols error-free and process them. The GS1 keys form the basis for efficient and cost-effective goods flow management from the manufacturer to the end user.

Supported Bar Codes

- GS1-128
- GS1 DataBar
- GS1 DataMatrix

GS1-128

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

The **GS1-128** is a special form of Code 128. This barcode is used for goods and palettes mainly in commerce and industry. The name **GS1-128** replaces the old name EAN/UCC-128.

The length of **GS1-128** is variable, however should not exceed the maximum length of 165 mm. Altogether a maximum of 48 rated character including the Application Identifier and FNC1 signs can be coded.

Length	Variable
Valid characters	ASCII character set
Check digit	Modulo 103

See also

- Code 128

GS1 DataBar

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

GS1 DataBar is a family of symbols most commonly seen in the GS1 DataBar Coupon. Formerly known as Reduced Space Symbology (RSS-14), this family of bar codes include:

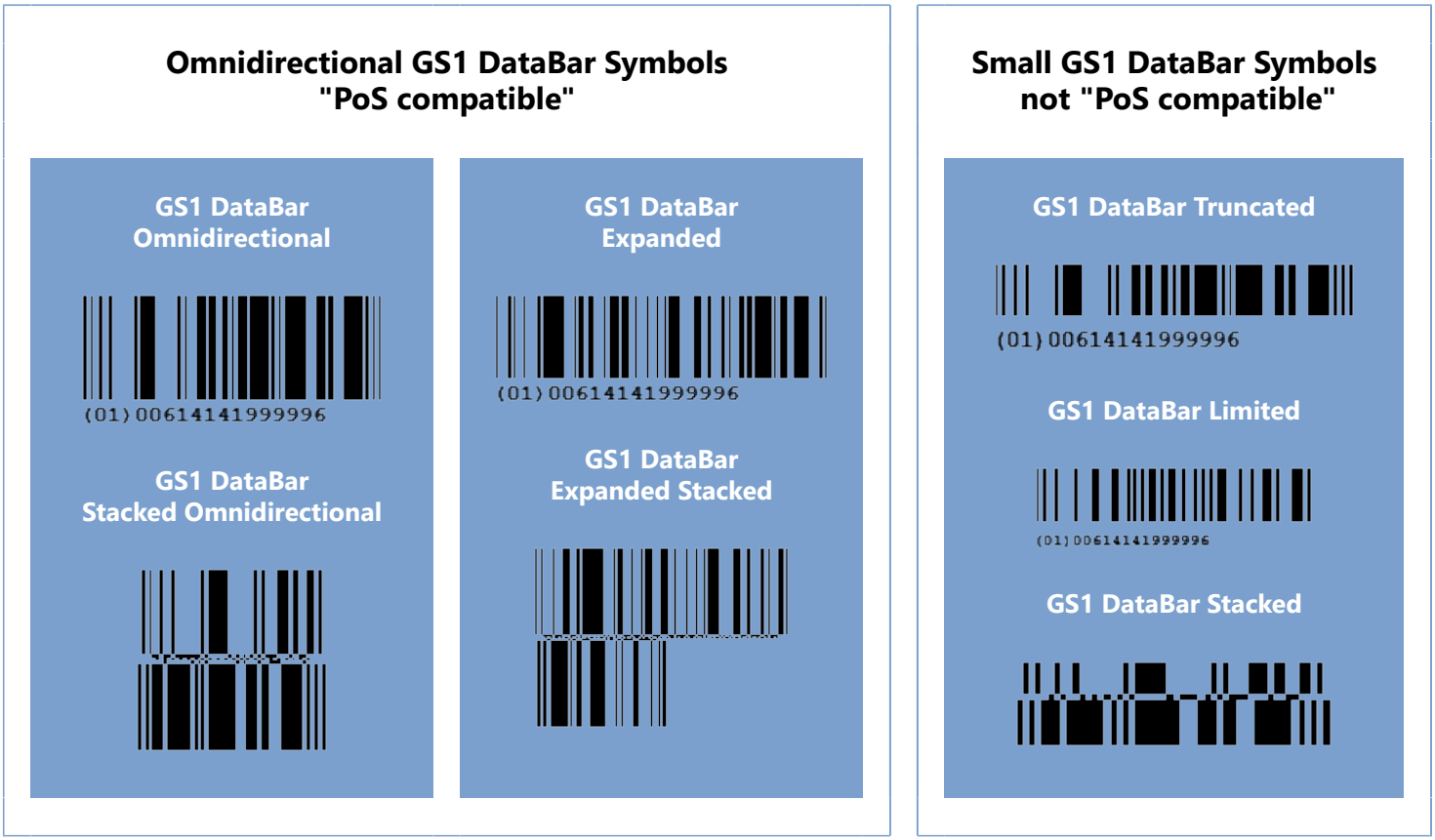
All **GS1 DataBar** bar codes encode a GTIN-12 or GTIN-13 in a 14-digit data structure. In order to make the GTIN-12 or GTIN-13 a 14-digit data structure, a leading zero or zeros is filled to the left of the GTIN. **GS1 DataBar Omnidirectional**, **GS1 DataBar Stacked Omnidirectional**, **GS1 DataBar Expanded**, and **GS1 DataBar Expanded Stacked** have omnidirectional scanning capability. **GS1 DataBar Truncated**, **GS1 DataBar Stacked** and **GS1 DataBar Limited** can only be scanned by a linear hand held or imaging scanning device: they cannot be scanned by omnidirectional scanners and are intended to be read by handheld scanners.

GS1 DataBar Stacked Omnidirectional is designed to condense the GTIN information into a more compact and square bar code suitable for use on smaller packages (such as the label stickers on fresh produce).

GS1 DataBar Limited, **GS1 DataBar Stacked** and **GS1 DataBar Truncated** are designed for very small item identification and are mainly used in the healthcare industry. Each encodes a GTIN-12 or GTIN-13 in 14-digit data structure. Only **GS1 DataBar Limited** uses an indicator digit 1.

In addition to encoding Application Identifier (01) GTIN, **GS1 DataBar Expanded** and **GS1 DataBar Expanded Stacked** can encode additional GS1 Application Identifiers such as sell-by date, weight, and lot number. Each symbol has a capacity of up to 74 characters. These attributes can help in controlling shrinkage, optimizing product replenishment, and improving the traceability of a product at the point of sale. They are seeing increased use in manufacturers' coupons.

This family of bar codes include:



GS1 DataMatrix

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

The **GS1 DataMatrix** is a 2D bar code with a high information density on relatively small space. A GTIN can be represented e.g. already on a space of 5 x 5 mm.

In the **GS1 DataMatrix** it is possible to code several data at the same time. It is mainly used in trade and industry particularly for labelling goods and pallets. It is usual to code additionally to the product code e.g. the weight and minimum durability date.

The **GS1 DataMatrix** is compatible to the existing GS1 standard and is protected for all GS1 applications.

Length	Variable
Valid characters	ASCII character set
Check digit	None

See also

- DataMatrix

GS1 Application Identifiers

Note

This list should only give an overview and has no claims to correctness and completeness.

AI	Dateninhalt	Format (*)	FNC1 required (****)
00	SSCC - Serial Shipping Container Code	n2 + n18	
01	GTIN - Global Trade Item Number	n2 + n14	
02	GTIN of contained trade items	n2 + n14	
10	Batch or lot number	n2 + an..20	(FNC1)
11 (**)	Production date (YYMMDD)	n2 + n6	
12 (**)	Due date (YYMMDD)	n2 + n6	
13 (**)	Packaging date (YYMMDD)	n2 + n6	
15 (**)	Best before date (YYMMDD)	n2 + n6	
16 (**)	Sell by date (YYMMDD)	n2 + n6	
17 (**)	Expiration date (YYMMDD)	n2 + n6	
20	Variant number	n2 + n2	
21	Serial number	n2 + an..20	(FNC1)
240	Additional item identification	n3 + an..30	(FNC1)
241	Customer part number	n3 + an..30	(FNC1)
242	Made-to-Order variantion number	n3 + n..6	(FNC1)
243	Packaging component number	n3 + an..20	(FNC1)
250	Secondary serial number	n3 + an..30	(FNC1)
251	Reference to source entity	n3 + an..30	(FNC1)
253	GDTI - Global Document Type Identifier	n3 + n13 + an..17	(FNC1)
254	GLN extension component	n3 + an..20	(FNC1)
255	GCN - Global Coupon Number	n3 + n13 + n..12	(FNC1)
30	Count of items	n2 + n..8	(FNC1)
310 (***)	Net weight, kilograms (variable measure trade item)	n4 + n6	
311 (***)	Length or first dimension, metres (variable measure trade item)	n4 + n6	
312 (***)	Width, diameter, or second dimension, metres (variable measure trade item)	n4 + n6	
313 (***)	Depth, thickness, height, or third dimension, metres (variable measure trade item)	n4 + n6	
314 (***)	Area, square metres (variable measure trade item)	n4 + n6	
315 (***)	Net volume, litres (variable measure trade item)	n4 + n6	
316 (***)	Net volume, cubic metres (variable measure trade item)	n4 + n6	
320 (***)	Net weight, pounds (variable measure trade item)	n4 + n6	
321 (***)	Length or first dimension, inches (variable measure trade item)	n4 + n6	
322 (***)	Length or first dimension, feet (variable measure trade item)	n4 + n6	
323 (***)	Length or first dimension, yards (variable measure trade item)	n4 + n6	
324 (***)	Width, diameter, or second dimension, inches (variable measure trade item)	n4 + n6	
325 (***)	Width, diameter, or second dimension, feet (variable measure trade item)	n4 + n6	

326 (***)	Width, diameter, or second dimension, yards (variable measure trade item)	n4 + n6	
327 (***)	Depth, thickness, height, or third dimension, inches (variable measure trade item)	n4 + n6	
328 (***)	Depth, thickness, height, or third dimension, feet (variable measure trade item)	n4 + n6	
329 (***)	Depth, thickness, height, or third dimension, yards (variable measure trade item)	n4 + n6	
330 (***)	Logistic weight, kilograms	n4 + n6	
331 (***)	Length or first dimension, metres	n4 + n6	
332 (***)	Width, diameter, or second dimension, metres	n4 + n6	
333 (***)	Depth, thickness, height, or third dimension, metres	n4 + n6	
334 (***)	Area, square metres	n4 + n6	
335 (***)	Logistic volume, litres	n4 + n6	
336 (***)	Logistic volume, cubic metres	n4 + n6	
337 (***)	Kilograms per square metre	n4 + n6	
340 (***)	Logistic weight, pounds	n4 + n6	
341 (***)	Length or first dimension, inches	n4 + n6	
342 (***)	Length or first dimension, feet	n4 + n6	
343 (***)	Length or first dimension, yards	n4 + n6	
344 (***)	Width, diameter, or second dimension, inches	n4 + n6	
345 (***)	Width, diameter, or second dimension, feet	n4 + n6	
346 (***)	Width, diameter, or second dimension, yards	n4 + n6	
347 (***)	Depth, thickness, height, or third dimension, inches	n4 + n6	
348 (***)	Depth, thickness, height, or third dimension, feet	n4 + n6	
349 (***)	Depth, thickness, height, or third dimension, yards	n4 + n6	
350 (***)	Area, square inches (variable measure trade item)	n4 + n6	
351 (***)	Area, square feet (variable measure trade item)	n4 + n6	
352 (***)	Area, square yards (variable measure trade item)	n4 + n6	
353 (***)	Area, square inches	n4 + n6	
354 (***)	Area, square feet	n4 + n6	
355 (***)	Area, square yards	n4 + n6	
356 (***)	Net weight, troy ounces (variable measure trade item)	n4 + n6	
357 (***)	Net weight (or volume), ounces (variable measure trade item)	n4 + n6	
360 (***)	Net weight, quarts (variable measure trade item)	n4 + n6	
361 (***)	Net weight, gallons U.S. (variable measure trade item)	n4 + n6	
362 (***)	Net weight, quarts	n4 + n6	
363 (***)	Net weight, gallons U.S.	n4 + n6	
364 (***)	Net volumne, cubic inches (variable measure trade item)	n4 + n6	
365 (***)	Net volumne, cubic feet (variable measure trade item)	n4 + n6	
366 (***)	Net volumne, cubic yards (variable measure trade item)	n4 + n6	
367 (***)	Logistic volumne, cubic inches	n4 + n6	
368 (***)	Logistic volumne, cubic feet	n4 + n6	
369 (***)	Logistic volumne, cubic yards	n4 + n6	
37	Count of trade items	n2 + n..8	(FNC1)
390 (***)	Applicable amount payable or Coupon value, local currency	n4 + n..15	(FNC1)
391 (***)	Applicable amount payable with ISO currency code	n4 + n3 + n..15	(FNC1)
392 (***)	Applicable amount payable, single monetary area (variable measure trade item)	n4 + n..15	(FNC1)

393 (***)	Applicable amount payable with ISO currency code (variable measure trade item)	n4 + n3 + n..15	(FNC1)
394 (***)	Percentage discount of a coupon	n4 + n4	(FNC1)
400	Customer's pruchase order number	n3 + an..30	(FNC1)
401	GINC - Global Identification Number for Consignment	n3 + an..30	(FNC1)
402	GSIN - Global Shipment Identification Number	n3 + n17	(FNC1)
403	Routing code	n3 + an..30	(FNC1)
410	Ship to - Deliver to Global Location Number	n3 + n13	
411	Bill to - Invoice to Global Location Number	n3 + n13	
412	Purchased from Global Location Number	n3 + n13	
413	Ship for - Deliver for - Forward to Global Location Number	n3 + n13	
414	Identification of physical location - Global Location Number	n3 + n13	
415	Global Location Number of the invoicing party	n3 + n13	
420	Ship to - Deliver to postal code within a single postal authority	n3 + an..20	(FNC1)
421	Ship to - Deliver to postal code with ISO country code	n3 + n3 + an..20	(FNC1)
422	Country of origin of trade item	n3 + n3	(FNC1)
423	Country of initial processing	n3 + n3 + n..12	(FNC1)
424	Country of processing	n3 + n3	(FNC1)
425	Country of disassembly	n3 + n3	(FNC1)
426	Country covering full process chain	n3 + n3	(FNC1)
427	Country subdivision of origin	n3 + an..3	(FNC1)
7001	NSN - NATO Stock Number	n4 + n13	(FNC1)
7002	UN/ECE meat carcasses and cuts classification	n4 + an..30	(FNC1)
7003	Expiration date and time	n4 + n10	(FNC1)
7004	Active potency	n4 + n..4	(FNC1)
7005	Catch area	n4 + an..12	(FNC1)
7006	First freeze date	n4 + n6	(FNC1)
7007	Harvest date	n4 + n6	(FNC1)
7008	Species for fishery pruposes	n4 + an..3	(FNC1)
7009	Fishing gear type	n4 + an..10	(FNC1)
7010	Prduction method	n4 + an..2	(FNC1)
703s	Number of processor with ISO country code	n4 + n3 + an..27	(FNC1)
710	National Healthcare Reimbursement Number (NHRN) - German PZN	n3 + an..20	(FNC1)
711	National Healthcare Reimbursement Number (NHRN) - France CIP	n3 + an..20	(FNC1)
712	National Healthcare Reimbursement Number (NHRN) - Spain CN	n3 + an..20	(FNC1)
713	National Healthcare Reimbursement Number (NHRN) - Brasil DRN	n3 + an..20	(FNC1)
8001	Roll products (width, length, core diameter, direction, splices)	n4 + n14	(FNC1)
8002	Cellular mobile telephone identifier	n4 + an..20	(FNC1)
8003	GRAI - Global Returnable Asset Identifier	n4 + n14 + an..16	(FNC1)
8004	GIAI - Global Individual Asset Identifier	n4 + an..30	(FNC1)
8005	Price per unit of measure	n4 + n6	(FNC1)
8006	Identification of components of a trade item	n4 + n14 + n2 + n2	(FNC1)
8007	International Bank Account Number (IBAN)	n4 + an..34	(FNC1)
8008	Date and time of production	n4 + n8 + n..4	(FNC1)
8010	Component/Part Identifier (CPID)	n4 + an..30	(FNC1)

8011	Component/Part Identifier serial number (CPID SERIAL)	n4 + n..12	(FNC1)
8012	Software version	n4 + an..20	(FNC1)
8017	Global Service Relation Number to identify the relationship between an organisation offering services and the provider of services	n4 + n18	(FNC1)
8018	Global Service Relation Number to identify the relationship between an organisation offering services and the receipient of services	n4 + n18	(FNC1)
8019	Service Relation Instance Number (SRIN)	n4 + n..10	(FNC1)
8020	Payment slip reference number	n4 + an..25	(FNC1)
8110	Coupon code identification for use in North America	n4 + an..30	(FNC1)
8111	Loyalty points of a coupon	n4 + n4	(FNC1)
8200	Extended Packaging URL	n4 + an..70	(FNC1)
90	Information mutually agreed between trading partners	n2 + an..30	(FNC1)
91 - 99	Company internal information	n2 + an..30	(FNC1)

(*) he first position indicates the length (number of digits) of the GS1 Application Identifier. The following value refers to the format of the data content. The following convention is applied:
n - numeric digit
an - any character
n3 - 3 numeric digits, fixed length
n..3 - up to 3 numeric digits
an..3 - up to 3 characters

(**) If only year and month are available, DD must be filled with two zeroes.

(***) The fourth digit of this GS1 Application Identifier indicates the implied decimal point position.
Example:
3100 - Net weight in kg without a decimal point
3102 - Net weight in kg with two decimal points

(****) All GS1 Application Identifiers indicated with (FNC1) are defined as of variable length and shall be delimited unless this element string is the last one to be encoded in the symbol.

HIBC Bar Codes

About the HIBC standards

In 1983 HIBCC was provided its initial mandate for the health care industry: develop a uniform bar code labeling standard for products shipped to hospitals. This relatively straight-forward concept represented a dramatic potential for the entire industry. By agreeing to place a consistent pattern of computer-readable bar codes on their products, manufacturers would provide a control mechanism that would yield enormous benefits to both their hospital customers and distributors.

The mandate to develop the standard grew out of a task force hosted by the American Hospital Association and composed of numerous other health care trade organizations, including those which ultimately founded HIBCC. Bar code technology had already proven a valuable tool for reducing labor costs and human error in other industries, such as retailing. In health care, the potential was even greater because of the impact errors can have on the quality of patient care.

The task force ultimately created the **Health Industry Bar Code (HIBC)** Standard, composed of two parts: Part One, the HIBC Supplier Labeling Standard, is the basis for the Universal Product Number, and covers the formats used by suppliers of healthcare products. Part Two, the HIBC Provider Applications Standard covers the formats used for internal labeling by health care providers themselves.

The following year HIBCC was formed to administer the standard and issue the Labeler Identification Codes (LIC's) which identify individual manufacturers and are included within each bar code. The LIC database now additionally provides the identifiers which are key elements in EDI transaction message formats and the HIBCC Universal Product Number Repository (UPN®).

Bar code technology changes rapidly. Advances in hardware have made it possible for "scanners" to understand multiple symbologies including those which are used for small-package and unit-of-use labeling. These developments have in turn made bar code use far more useful and valuable to the industry.

HBIC data structure

- Unique data structure for short to long alphanumeric product codes
- Symbologies for marking large to smallest products
- ISO conforming specification
- Interoperability with any other coding system
- Suitable for any ISO Bar Code, 2D-Code and RFID

The *HIBC.lbex* sample label can be found in the following directory: *%InstallDir%\Samples*.

To combine the Primary and Secondary Data use the HIBC LIC Concatenation Format. In this case, a slash (/) is used as a delimiter between the Primary and Secondary Data. One check character (Module 43) at the end of the symbol is used which will verify the entire symbol.

	Field Length	(F)ixed Length (V)ariable Length	Description	Example
Primary Data (UDI DI)				
+	1	F	HIBC Supplier Labeling Flag Character	+
	4	F	LIC - Labeler Identification Code Alphanumeric number, with the first character always being alphabetic.	E999
	1-18	V	Labeler Product or Calatog Number Alphanumeric data.	FF6098765
	1	F	Unit of Measure ID Numeric value only, 0 through 9, where 0 is for unit-of use items.	1
Secondary Data (UDI PI)				
$	1-18	V	Lot/Batch Number, alphanumeric	
$$	4 1-18	F V	Date Field (MMYY) Lot/Batch Number, alphanumeric	0716 811302X811106
$$2	6 1-18	F V	Date Field (MMDDYY) Lot/Batch Number, alphanumeric	
$$3	6 1-18	F V	Date Field (YYMMDD) Lot/Batch Number, alphanumeric	
$$4	8 1-18	F V	Date Field (YYMMDDHH) Lot/Batch Number, alphanumeric	
$$5	5 1-18	F V	Date Field (YYJJJ) - julian date format Lot/Batch Number, alphanumeric	
$$6	7 1-18	F V	Date Field (YYJJJHH) - julian date format Lot/Batch Number, alphanumeric	
$+	1-18	V	Serial Number, alphanumeric	
$$+	4 1-18	F V	Date Field (MMYY) Serial Number, alphanumeric	
$$+2	6 1-18	F V	Date Field (MMYYDD) Serial Number, alphanumeric	
$$+3	6 1-18	F V	Date Field (YYMMDD) Serial Number, alphanumeric	
$$+4	8 1-18	F V	Date Field (YYMMDDHH) Serial Number, alphanumeric	
$$+5	5 1-18	F V	Date Field (YYJJJ) - julian date format Serial Number, alphanumeric	
$$+6	7 1-18	F V	Date Field (YYJJJHH) - julian date format Serial Number, alphanumeric	
/S	1-18	V	Additional Serial Number, alphanumeric	
/14D	8	F	Expiry Date (YYYYMMDD)	
/16D	8	F	Date of Manufacture (YYYYMMDD)	

Check Digit Calculations

A check digit is a form of redundancy check used for error detection. It consists of a single digit (sometimes more than one) computed by an algorithm from the other digits (or letters) in the sequence input. With a check digit, one can detect simple errors in the input of a series of characters (usually digits) such as a single mistyped digit or some permutations of two successive digits.

See also

- Modulo 10 (EAN)
- Modulo 10 (Code 2 of 5)
- Modulo 10 (Identcode/Leitcode)
- Modulo 10 (Luhn Algorithm)
- Modulo 11
- Modulo 43

Modulo 10 (EAN)

Modulo 10 is used by many bar code symbologies, for example, EAN-13, GTIN-13.

The check digit is calculated according to Modulo 10 with a weigthing of 3 from the right.

Example Code

Digits	4 0 1 2 3 4 5 9 8 7 6 5
Weights	1 3 1 3 1 3 1 3 1 3 1 3
Multiply digits by weight	4 0 1 6 3 12 5 27 8 21 6 15
Add results	4 + 0 + 1 + 6 + 3 + 12 + 5 + 27 + 8 + 21 + 6 + 15 = 108
Find the remainder mod 10	108 Mod. 10 = 8 (108/10 = 10 Rest 8)
Subtract from 10	10 - 8 = 2
Check digit	2

See also

- Modulo 10 (Code 2 of 5)
- Modulo 10 (Identcode/Leitcode)
- Modulo 10 (Luhn Algorithm)

Modulo 10 (Code 2 of 5)

For **Code 2 of 5 bar codes** (e.g. Code 2 of 5 Interleaved), calculation of the check digit is done in accordance with Modulo 10 with a weighting of 3. For the calculation, it begins with the first digit from the left with the weight factor of 3. The individual products are added to get a sum. The difference between the product and the next full "tens" (rounding) gives the check digit.

Example Code

Digits	1 2 3 4 5
Weights	3 1 3 1 3
Multiply digits by weight	3 2 9 4 15
Add results	3 + 2 + 9 + 4 + 15 = 33
Find the remainder mod 10	33 Mod 10 = 3 (33/10 = 3 Rest 3)
Subtract from 10	10 - 3 = 7
Check digit	7

See also

- Modulo 10 (EAN)
- Modulo 10 (Identcode/Leitcode)
- Modulo 10 (Luhn Algorithm)

Modulo 10 (Identcode/Leitcode)

For the Deutsche Post Identcode and the Deutsche Post Leitcode, calculation of the check digit is done in accordance with Modulo 10 with a weighting of 4 for odd positions and with a weighting of 9 for even positions.

Example Code

Digits	0 1 2 3 4 5 6 7 8 9 0 1 2
Weights	4 9 4 9 4 9 4 9 4 9 4 9 4
Multiply digits by weight	0 9 8 27 16 45 63 32 81 0 9 8
Add results	0 + 9 + 8 + 27 + 16 + 45 + 63 + 32 + 81 + 0 + 9 + 8 = 298
Find the remainder mod 10	298 Mod. 10 = 8 (298/10 = 29 Rest 8)
Check digit	8

See also

- Modulo 10 (EAN)
- Modulo 10 (Code 2 of 5)
- Modulo 10 (Luhn Algorithm)

Modulo 10 (Luhn Algorithm)

The **Luhn Algorithm** or **Luhn Formula**, also known as the "modulus 10" or "mod 10" algorithm, is a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers, National Provider Identifier numbers in US and Canadian Social Insurance Numbers. It was created in 1960 by IBM scientist Hans Peter Luhn.

The algorithm is in the public domain and is in wide use today. It is not intended to be a cryptographically secure hash function; it was designed to protect against accidental errors, not malicious attacks. Most credit cards and many government identification numbers use the algorithm as a simple method of distinguishing valid numbers from mistyped or otherwise incorrect numbers.

Example Code

Digits	4 5 5 6 7 3 7 5 8 6 8 9 9 8 5
Weights	2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Multiply digits by weight	8 5 10 6 14 3 14 5 16 6 16 9 18 8 10
Checksum of weighted digits	8 5 1 6 5 3 5 5 7 6 7 9 9 8 1
Add checksums	8 + 5 + 1 + 6 + 5 + 3 + 5 + 5 + 7 + 6 + 7 + 9 + 9 + 8 + 1 = 85
Find the remainder mod 10	85 Mod. 10 = 8 (85/10 = 8 Rest 5)
Check digit	5

See also

- Modulo 10 (EAN)
- Modulo 10 (Code 2 of 5)
- Modulo 10 (Identcode/Leitcode)

Modulo 11

Modulo 11 is e.g. used by PZN.

Calculating the the check digit of PZN-8: Multiply the first digit by 1, the second digit by 2, ... the seventh digit by 7. Add the results and divide it by 11 and the remainder (Modulo 11) is the check digit. If the check digit is a "10", the PZN is not released as it is not considered valid.

Example Code PZN-8

Digits	3 6 3 1 9 4 2
Weights	1 2 3 4 5 6 7
Multiply digits by weight	3 12 9 4 45 24 14
Add results	3 + 12 + 9 + 4 + 45 + 24 + 14 = 111
Find the remainder mod 11	111 Mod 11 = 1 (111/11 = 10 Rest 1)
Check digit	1

Modulo 43

A check digit in accordance with Modulo 43 is used by Code 39 and the HIBC Bar Codes, for example.

First, all characters in the code are allocated reference numbers (see reference table). These reference numbers are added to get a sum. This sum is divided by 43. The remainder from this division (Modulo 43) equates to the check sum which is then represented consistent with the reference number by means of the reference table.

Example Code

Data	1 5 9 A Z
Reference numbers	1 5 9 19 35
Add reference numbers	1 + 5 + 9 + 19 + 35 = 60
Find the remainder mod 43	60 Mod 43 = 17 (60/43 = 1 Rest 17)
Check digit	H (reference number 17)

Reference Table

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
W	X	Y	Z	-	.	Space	$	/	+	%					
32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**					

GTIN - Global Trade Item Number

Global Trade Item Number (GTIN) is an identifier for trade items. Such identifiers are used to look up product information in a database (often by inputting the number through a bar code scanner pointed at an actual product) which may belong to a retailer, manufacturer, collector, researcher, or other entity. The uniqueness and universality of the identifier is useful in establishing which product in one database corresponds to which product in another database, especially across organizational boundaries.

GTINs may be 8, 12, 13 or 14 digits long, and each of these 4 numbering structures are constructed in a similar fashion, combining Company Prefix, Item Reference and a calculated Check Digit (GTIN-14 adds another component - the Indicator Digit, which can be 1-8).

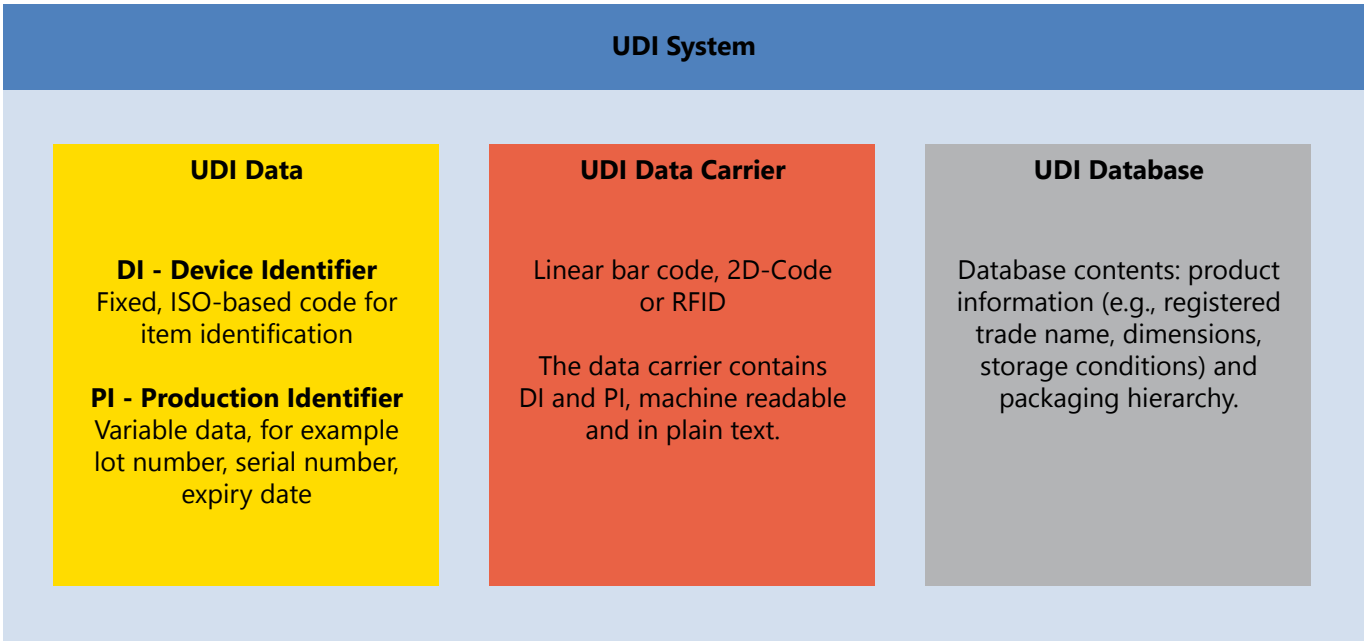
Name	Former Name
GTIN-8	EAN-8
GTIN-12	UPC-A
GTIN-13	EAN-13
GTIN 14	-

The following table demonstrates the structure of **GTINs** in a GTIN-compliant database:

Type of GTIN	GTIN Digit													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GTIN-8	0	0	0	0	0	0	N	N	N	N	N	N	N	C
GTIN-12	0	0	N	N	N	N	N	N	N	N	N	N	N	C
GTIN-13	0	N	N	N	N	N	N	N	N	N	N	N	N	C
GTIN 14	N	N	N	N	N	N	N	N	N	N	N	N	N	C

UDI - Unique Device Identification

Unique Device Identification (UDI) is a global normalized system for the identification of medical devices. The **UDI** should be visible, machine-readable and in plain text on the product itself. The **UDI** system also includes the development and operation of a **UDI** database (UDID) that contains a variety of information about the products.



A directive from the United States Food and Drug Administration (FDA) and the European Commission has introduced legislation regarding the **Unique Device Identification (UDI)** of medical devices and instruments. From September 2014, all medical class III devices and instruments that require direct part marking (DPM) must also carry a UDI. Class II and Class I devices will require UDI from September 2016 and September 2018 respectively, with the last deadline in September 2020.

See also

- HIBC Bar Codes

Databases

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

A wealth of data which are outside the label can be used within the label. But how can I find and import these data in **Labelstar Office**? The answer is very simple: You have to create and use a data connection.

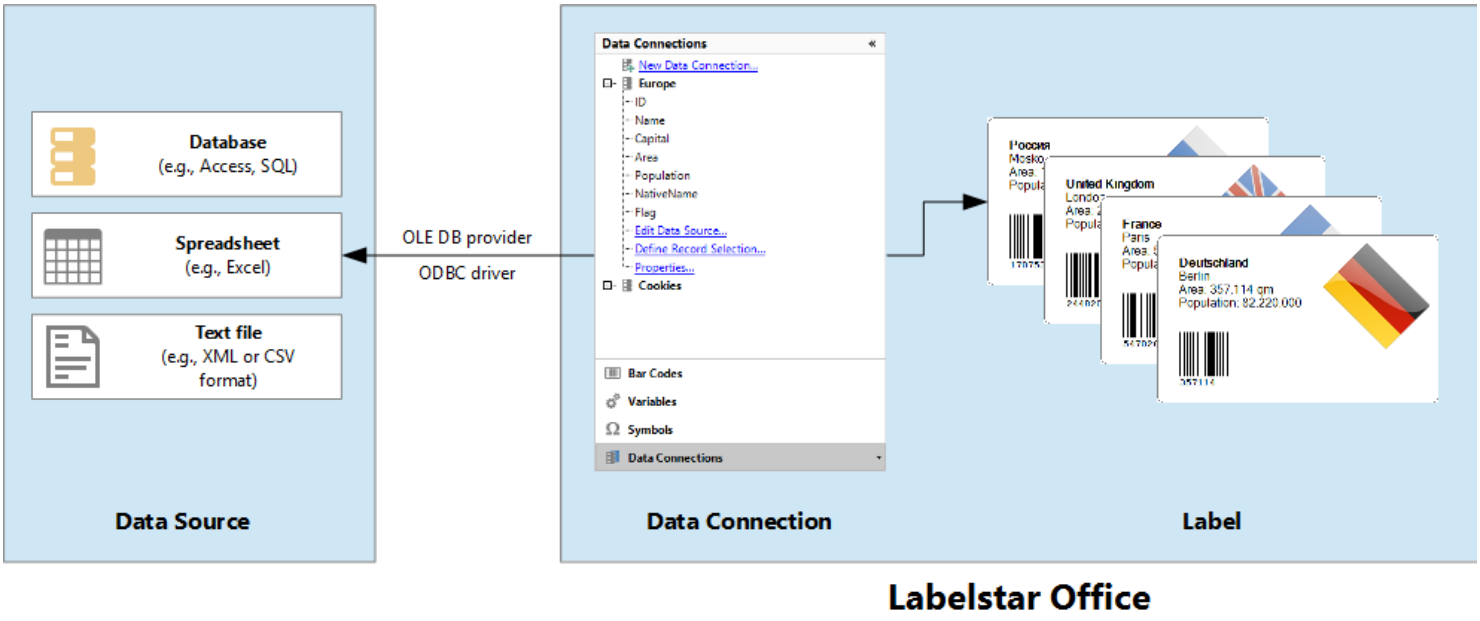
Data on a label can come from two different locations. The data may be stored directly within the label, or it may be stored in an external data source, such as a text file or a database. This external data source is connected to the label through a data connection, which is a set of information that describes how to locate, log in to, and access the external data source.

The main benefit of connecting to external data is that you can periodically analyze this data without repeatedly copying the data to your label, which is an operation that can be time consuming and prone to error.

To bring external data into **Labelstar Office**, you need access to the data. If the external data source that you want to access is not on your local computer, you may need to contact the administrator of the database for a password, user permissions, or other connection information. If the data source is a database, make sure that the database is not opened in exclusive mode. If the data source is a text file or a spreadsheet, make sure that another user does not have it open for exclusive access.

Many data sources also require an OLE DB provider or ODBC driver to coordinate the flow of data between **Labelstar Office**, the connection file, and the data source.

The following diagram summarizes the key points about data connections.



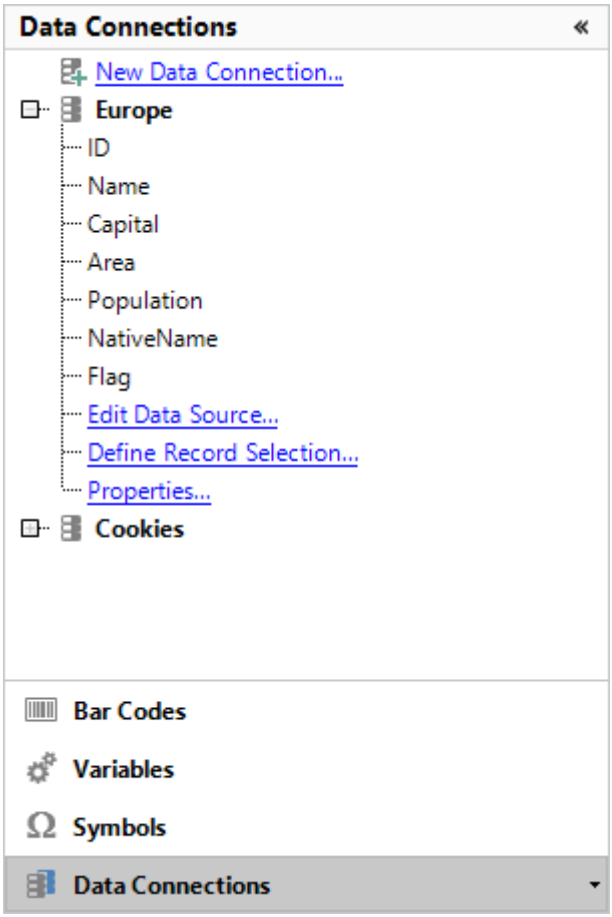
See also

- Create Data Connection
- Define Logical Connection Path
- Create Database Label

Create Data Connection

To specify a new data connection, proceed as follows:

1. Activate **Data Connections** view.
2. Click on **New Data Connection**.
   The **Data Connection Wizard** opens.
3. Select the data source you want to use.
4. Follow the instructions in the wizard.
5. After the successful definition, the new data connection is shown in the list and the associated database fields can be used on a label.



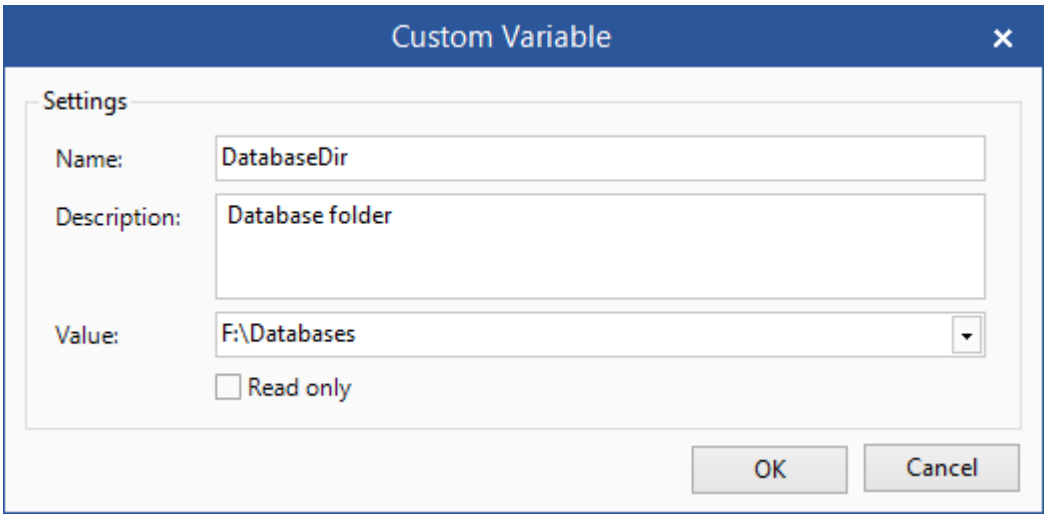
Define Logical Connection Path

You can use variables to define the file path of the data connection (for example, "`$InstallDir\Samples\Allergens\Cookies.accdb`").

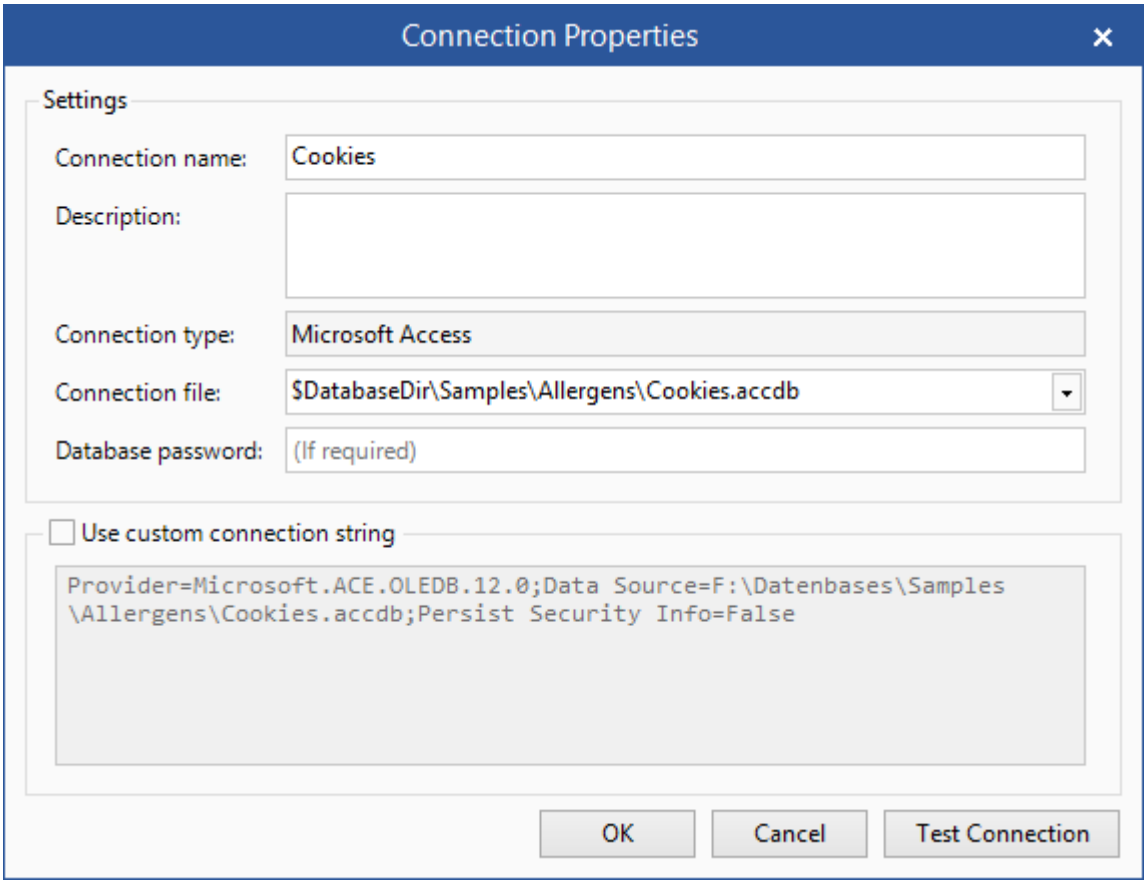
Labelstar Office supports a larger number of predefined path variables, but you can also create your own user-defined variables.

To define the file path of a connection file using a user-defined variable, proceed as follows:

1. Define a new user-defined variable.



2. Activate **Data Connections** view and select a data connection.
3. Expand data connection and then click on **Properties**.
4. Enter the *$DatabaseDir* variable in the text field in the connection file at the desired point as a placeholder.


Connection Properties
Settings
Connection name:
Cookies
Description:
Connection type:
Microsoft Access
Connection file:
$DatabaseDir\Samples\Allergens\Cookies.accdb
Database password:
(If required)
Use custom connection string
Provider=Microsoft.ACE.OLEDB.12.0;Data Source=F:\Datenbases\Samples
\Allergens\Cookies.accdb;Persist Security Info=False
OK
Cancel
Test Connection

Create Database Label

To learn how you can create a database label and to see just how easy it is, click on this link to see our video tutorials.

The sample data used in the video can be found in the directory: *%InstallDir%\Samples\Database*.

Europe.accdb Database (Microsoft Access format)
Europe.lbex Label definition
Europe.txt Database (Text format)
Europe.xml Database (XML format)
Import Europe_accdv data connection.lbdx Import file for data connection *Europe* (Microsoft Access format)
Import Europe_txt data connection.lbdx Import file for data connection *Europe* (Text format)
Import Europe_xml data connection.lbdx Import file for data connection *Europe* (XML format)

OLE DB Provider and ODBC Driver

Labelstar Office uses the OLE DB interface in order to generate connections to external databases. Using OLE DB provider and ODBC driver, you can access a large number of common database systems. Alternatively, you can export the data from the database and save it in a file format (e.g. XML or CSV) which **Labelstar Office** is able to read directly.

Is the right OLE DB Provider or OLE Driver installed?

Normally, the operating system provides a range of OLE DB providers and ODBC drivers. In addition, you can buy OLE DB providers and ODBC drivers on the software market, download them online or get them directly from the producer of the database system. More information about installing these OLE DB providers or ODBC drivers can be found in the database documentation, or contact the database producer.

Note

Labelstar Office is a 32 bit application and can therefore only integrate 32 bit OLE DB providers and ODBC drivers.

Logging

Required program variant **PROFESSIONAL**

For more information, see Program Variants.

With the logging you can retrace which data when, by whom and on which printer was printed.

Which information is saved at logging?

The logging option contained in **Labelstar Office** logs the following features:

- Date/Time of printing
- Number of copies
- Page name
- Label name
- Printer name
- User name
- Field contents

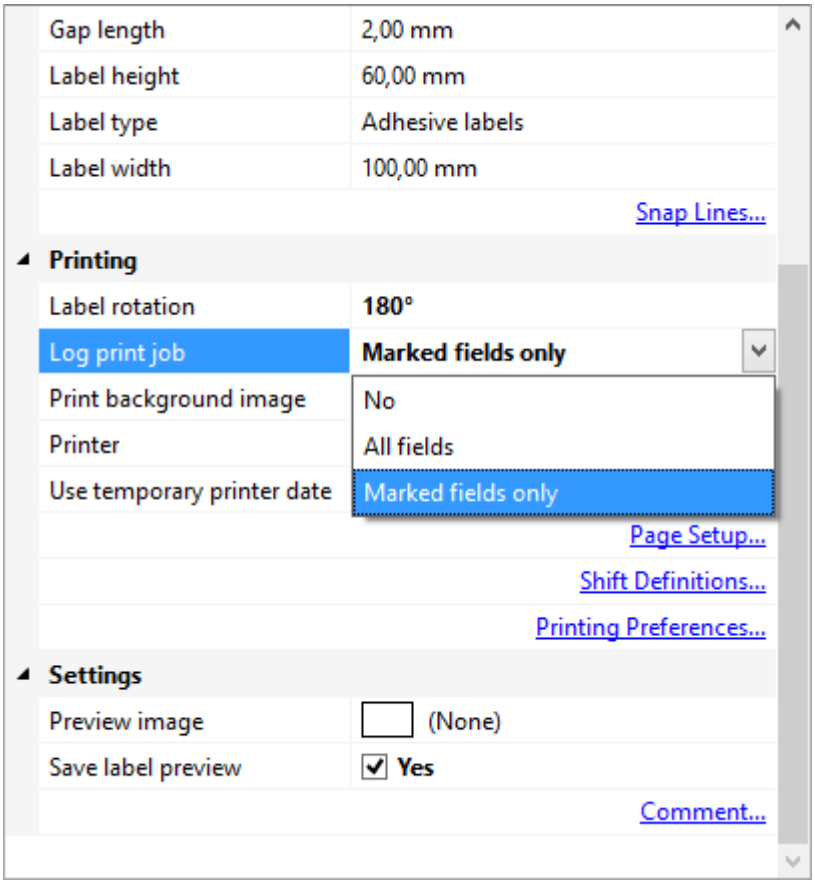
See also

- Activate and Deactivate Logging
- Log File Location
- «Logging» Tab

Activate and Deactivate Logging

So aktivieren und deaktivieren Sie die Protokollierung

1. Select **Label Properties** and click **Log print job**.



2. To activate logging select one of the following options:
   - **All fields** All field contents are logged.
   - **Marked fields only** Only the contents of the fields are logged, in which the **Log** option is enabled.
3. To deactivate logging select **No**.

See also

- «Logging» Tab

Log File Location

In the «Logging» tab, you can specify the file path of the log files. Select only one path on which all users have access.

Note

Select never only *C:* or *C:\Windows*. If at all, please create a new folder where the program can save its log files (e.g. *C:\Log*).

Program Options

In this dialog box, you can change several basic settings and customize the program to suit your personal preferences.

To change the program options, proceed as follows:

1. Select the **File** tab, and then click **Options**.
   The **Options** dialog box opens.
2. Change the desired settings.
3. Click **OK** to save your changes.

See also

- «General» Tab
- «Printing» Tab
- «Advanced» Tab
- «Label Preview» Tab
- «Memory Card» Tab
- «Logging» Tab
- «Standard Labels» Tab
- «File Locations» Tab

«General» Tab

Settings

Save label without prompt Activate this check box if the label should be saved without confirmation prompt.

Single document interface Activate this check box if only one label should ever be opened. If this option is not selected then multiple labels can be opened in parallel.

Retain initial settings for label directory Activate this check box if the currently set label directory should be shown by default when a label is opened or saved. If this option is not selected then the last directory used is always kept.

Retain initial settings for image directory Activate this check box if the currently set image directory should be shown by default when an image is opened or saved. If this option is not selected then the last directory used is always kept.

Check for updates automatically Here, you specify how often **Labelstar Office** should search for program updates. In order to search for updates, your computer must have an internet connection and your firewall must not block access.

Personalize your copy

Language Select an entry from the list in order to change the program language. The system language which is currently set will be used by default (provided that the language is available, if not then English will be used).

At program startup

Select how **Labelstar Office** should behave at program startup:

- **Empty label** Opens a blank label.
- **Open recent label** Shows the recently opened label.
- **Open label** Opens a particular label. Click  to choose a file.
- **Show 'Open File' dialog box** Displays the 'Open File' dialog box to choose a label.

«Printing» Tab

Settings

Default printer You can select the default printer for **Labelstar Office** here. **Labelstar Office** initially uses the Windows default printer, but you can select a different default printer for the print output. The Windows default printer and the **Labelstar Office** default printer are independent of one another. If you change one of the defaults, this does not affect the other one.

The default printer which you select for **Labelstar Office** is a program setting, i.e. all labels which you print with **Labelstar Office** will be sent to this printer unless you select another printer for a specific label.

Quick Print

Number of copies Here, you specify the number of copies to be printed by entering a number or changing the number with the help of the arrow control elements.

«Advanced» Tab

"User Inputs" Dialog Box

Use Enter key as Tab key Activate this check box if the input focus should move to the next item in the activation sequence when the enter key  is pressed. The dialog box is closed automatically (equivalent to clicking on **OK**) when the input focus has reached the last item in the tab order.

Align check boxes Activate this option if the check boxes in the dialog box should be arranged in columns.

Show error message dialog Activate this check box if errors should be show with a notification in addition to a red border.

Calendar

By selecting the **First day of week** and the **First week of year**, you can define how **Labelstar Office** works out the calendar weeks.

In Europe, calendar weeks are counted in accordance with ISO 8601. Calendar weeks therefore have 7 days, begin on a Monday and are numbered consecutively throughout the year. Calendar week 1 in a year is the week which contains the first Thursday (first 4-day week). As a result of this, it may occur that 01/01 does not fall in CW 1, but rather in CW 52 or 53 of the previous year (as was the case for 01/01/2012 or 01/01/2016).

In the USA and other countries, calendar weeks begin on a Sunday and calendar week 1 in a year is the week which contains 1 January. This can (occasionally) result in a year having 54 calendar weeks.

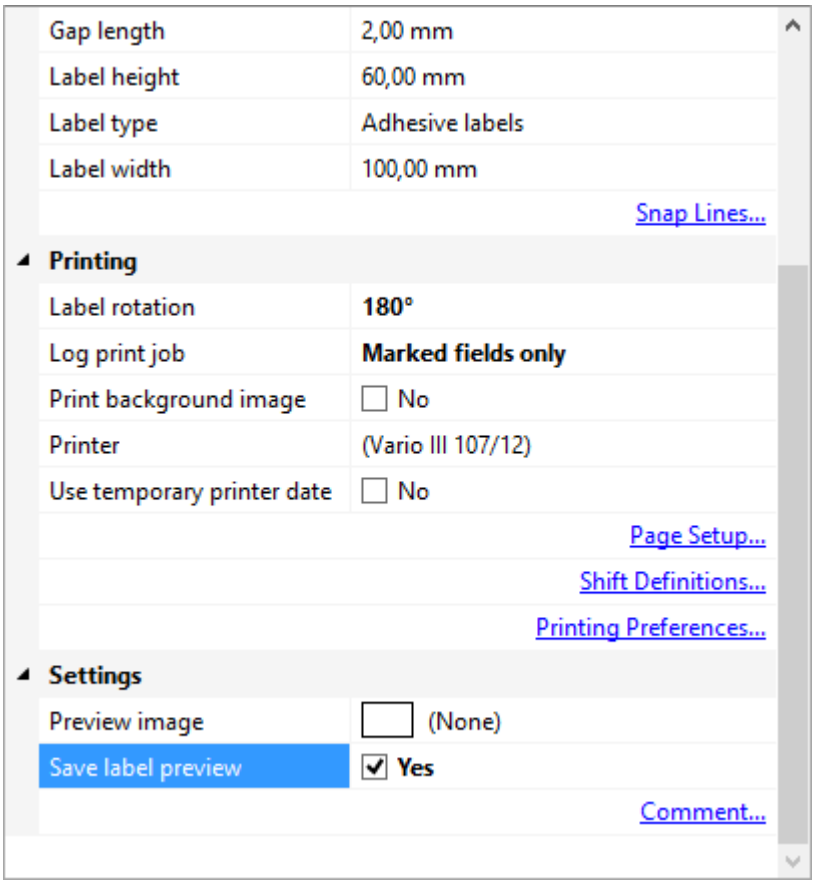
«Label Preview» Tab

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

In this tab you can specify different settings (size, output format and colour depth) of the preview image which is saved paralled to the label.

Activate the option **Save label preview for all labels** if you want to save a preview to each label. Do you want to activate the label preview only for particular labels then activate the option **Save label preview** in the label settings.



«Memory Card» Tab

Required program variant **BASIC, PROFESSIONAL**

For more information, see Program Variants.

Settings

Overwrite file on Memory Card without prompt Activate this check box if the label should be saved on the memory card without a confirmation prompt.

Save preview image as PNG file Activate this check box if a preview image in PNG format should be saved on the memory card in addition to the label.

Save preview image as JPEG file Activate this check box if a preview image in JPEG format should be saved on the memory card in addition to the label.

Append file extension (.prn) Activate this check box if the .prn file name extension should be appended to the label names by default. If this option is not selected then the file path is created without a file name extension.

Save/Read File

Default file location (printer-internal drive) Here, you can enter the default directory which should be used for creating the file path.

Default file location (system drive) Here, you can enter the default directory which should be used for creating the file path.

«Logging» Tab

Required program variant PROFESSIONAL

For more information, see Program Variants.

In this tab, you can change the log settings.

Log File

You can specify where the log file is to be saved.

Folder Enter the name of the folder in which the log file is to be created or click on ... to browse the folder.

File Name Enter a fixed file name or use placeholder *%date%*, *%time%*, *%labelname%*, *%printername%*, which are replaced by current values to define a variable file name (e.g. *%labelname%%date%.log*).

Note

Select only one path on which all users have access. Select never only *C:* or *C:\Windows*. If at all, please create a new folder where the program can save its log files (for example, *C:\Log*).

Overwrite existing log file Activate this check box if the log file should overwrite and replace the existing log file.

Use local time for file naming Activate this check box if the local time is to be used for creating the log file name. If this option is not selected, the coordinated world time (Coordinated Universal Time/UTC) is used.

Log File Format

In this section you can define the file format. The log file is saved in CSV format. For more information about data to be saved, see Logging.

Log File Rollover

In this section you specify if and when a new log file is to be commenced.

Maximum file size Select this option to permit the creation of aditional log files if the maximum file size is reached. Each new file name consists of the original name of the log file with an ascending number.

Do not create new log file All information is logged in an only one ever growing file.

«Standard Labels» Tab

All defined default labels are shown in this tab. You can define your own labels beyond the labels which are available by default with the help of the **Create a new standard label** option. These labels can then be used for creating a new label. This results in the option of determining not only the label size but also the label type (adhesive or continuous labels).

«File Locations» Tab

In this tab, you can change the location of the directories and files used in the program.

To change the location, proceed as follows:

1. Select an entry.
2. Click  .
   The dialog box **Select File** is opened.
3. Choose a new location.

Markup Tags

Required program variant BASIC, PROFESSIONAL

For more information, see Program Variants.

With the help of markup tags formatting instructions can be inserted into the text.

Note

Markup tags must be well formatted. That means that all tags must be properly closed and all attributes must have values enclosed in single quotes.

Supported Markup Tags

Markup Tag	Description	Examples
b	Defines bold text.	bold text -> **bold text**
br	Defines a single line break.	line 1 line 2 -> line 1 line 2
em	Defines emphasized text (same as italic text).	emphasized text -> *emphasized text*
font	Defines font, color, and size for text. *Arributes:* • ***size**: Font size* • ***name**: Font name* • ***color**: Font color*	Example text -> Example text Example text -> Example **text**
i	Defines italic text.	italic text -> *italic text*
rtl	Right-to-left text.	<rtl>Example text</rtl> -> txet elpmaxE
shadow	Defines text with a shadow. *Attributes:* • ***color**: Shadow color (Default: Black)* • ***offset**: Shadow offset (Default: 1,1)* • ***strength**: Shadow size (Default: 1,1)* • ***style**: Shadow style (Default: Solid)* Solid Blurred	<shadow style='Blurred'>Example text</shadow> -> Example text <shadow color='Black'>Example text</shadow> -> Example text
strike	Defines strikethrough text.	<strike>strikethrough text</strike> -> ~~strikethrough text~~
stroke	Defines stroked text. *Attributes:* • ***width**: Border width (Default: 1)* • ***color**: Border color (Default: Black)*	<stroke color='#FF0000'>Example text</stroke> -> Example text
strong	Defines important text.	important text -> **important text**
sub	Defines subscripted text.	H₂O -> H_2O
sup	Defines superscripted text.	footer¹ -> footer[1]
u	Defines underlined text.	<u>underlined text</u> -> underlined text

Supported entity characters:

Character	Description	Code
"	quotation mark	"
'	single quote, apostrophe	'
&	ampersand sign	&
<	less than sign	<
>	greater than sign	>
	non-breaking space	
©	copyright sign	©
®	registered trade mark sign	®
™	Registered Trademark sign	™

Supported Graphic and Vector Formats

Note

Which formats are supported depends on the program variant you use. For more information, see Program Variants.

- **ANIMATED GIF** - Graphics Interchange Format
- **BMP** - Standard Windows Bitmap
- **CUT** - Dr. Halo/Dr. Genius Clipboard Format
- **DDS** - Microsoft DirectDraw Surface Format
- **DIB** - Standard Windows Bitmap Format
- **DICOM** - Digital Imaging and Communications in Medicine
- **EMF** - Enhanced Windows Metaformat
- **EXIF** - Exchangable Image Format
- **EXR** - OpenEXR Format
- **FAX, G3** - Group 3 Raw Fax Format
- **GIF, Interlaced GIF** - Graphics Interchange Format
- **HDR** - High Dynamic Range Format
- **IFF** - Interchange Format
- **ICO (single and multi page)** - Icone Format
- **J2K, J2C** - JPEG-2000 Codestream
- **JB2, JBIG2** - Joint Bi-level Image Experts Group
- **JIF, JFIF** - JPEG File Interchange Format
- **JNG** - JPEG Network Graphics
- **JP2** - JPEG-2000 Format
- **JPEG, JPG, JPE** - Joint Pointgraphic Expert Group
- **JPEG progressive**
- **KOA** - KOALA Format
- **LBM** - Interchange File Format-Interleaved Bitmap
- **MNG** - Multiple-image Network Graphics
- **PBM** - Portable Bitmap File
- **PBM Raw** - Portable Bitmap BINARY
- **PCD** - Kodak Photo-CD file
- **PCT, PICT, PIC** - Macintosh PICT Format
- **PCX** - PC Paintbrush Format
- **PDF Multi-page** - Portable Document Format
- **PDF/A** - Document Format for long term preservation
- **PFM** - Portable Float Map
- **PGM** - Portable Graymap BINARY
- **PGM RAW** - Protable Graymap File
- **PSD** - Photoshop File
- **PNG** - Portable Network Graphics Format
- **PNM** - Portable Any Map
- **PPM** - Portable Pixmap File
- **PPM RAW** - Portable Pixmap BINARY
- **RAS** - Sun Raster Format
- **RAW camera image**

- **RAW memory bits** - RAW bitmap
- **RLE** - Standard Windows Bitmap format
- **SGI** - Silicon Graphics Image Format
- **TGA, TARGA** - TARGA Image Format
- **TIFF, TIF** - Tagged Image Format
- **TIFF Multi-page** - Multi-page Tagged Image Format
- **WBMP, WAP, WBM** - Wireless Bitmap
- **WEBP** - WebP Image Format
- **WMF** - Standard Windows Metaformat
- **XBM** - X Bitmap Format
- **XPM** - X Pixmap Format

Food Allergen Labelling

The new EU Regulation 1169/2011 on the provision of food information to consumers changes existing legislation on food labelling including:

- Mandatory nutrition information on processed foods.
- Mandatory origin labelling of unprocessed meat from pigs, sheep, goats and poultry.
- Highlighting allergens e.g. peanuts or milk in the list of ingredients.
- Better legibility i.e. minimum size of text.
- Requirements on information on allergens also cover non pre-packed foods including those sold in restaurants and cafés.

The new rules will apply from 13 December 2014. The obligation to provide nutrition information will apply from 13 December 2016.

Foods that need to be labelled on pre-packed foods when used as ingredients are:

- **Cereals containing gluten** such as wheat, rye, barley, oats, spelt or khorasan
- **Crustaceans** for example prawns, crabs, lobster, crayfish
- **Eggs**
- **Fish**
- **Peanuts**
- **Soybeans**
- **Milk**
- **Nuts** such as almonds, hazelnuts, walnuts, cashews, pecan nuts, Brazil nuts, pistachio nuts, macadamia (or Queensland) nuts
- **Celery** (including celeriac)
- **Mustard**
- **Sesame seeds**
- **Sulphur dioxide and sulphites** (>10mg/kg or 10mg/l)
- **Lupin**
- **Mollusc** for example clams, mussels, whelks, oysters, snails and squid

Sample

This sample shows how you can create a database label, using the variable $ReplacePattern, to automatically highlight certain words.

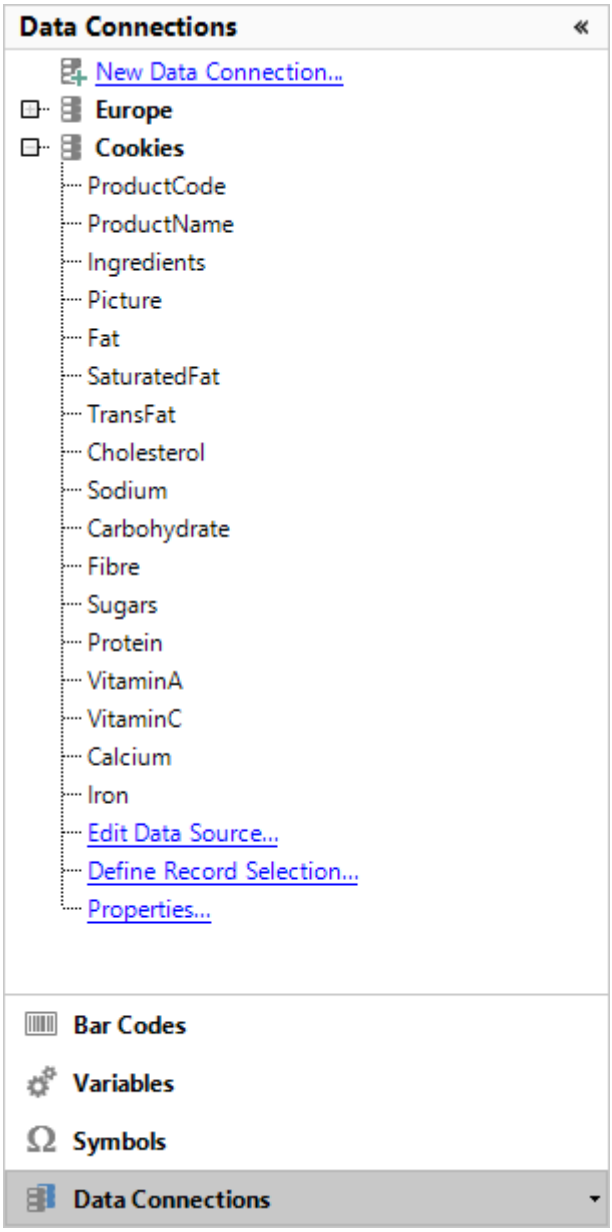
The example can be found in the samples folder: *%InstallDir%\Samples\Allergens*.

%InstallDir% refers to the installation directory that was specified during the installation process, what is typicall *C:\Programme (x86)\Carl Valentin GmbH\Labelstar Office*.

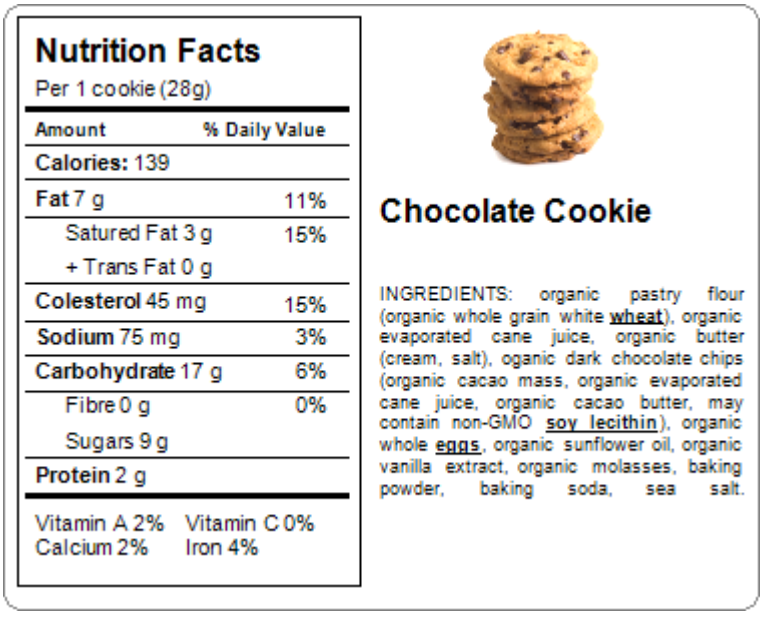
Allergens.txt This file contains the list of allergens that are being emphasized.
Cookies.accdb Microsoft Access database with the ingredients lists.
Cookies.lbex Label definition
Import Cookies data connection.lbdx Import file for data connection *Cookies*.

To open the sample label, proceed as follows:

1. Open **Labelstar Office**.
2. Select **Data Connections** view.
3. Click on ▼ on the right to expand and then **Import**.
4. Browse for the file **Import Cookies data connection.lbdx**.



4. Open the label **Cookies.lbex**.



Printing in an SAP Environment

Required program variant **PROFESSIONAL**

For more information, see Program Variants.

Prerequisites

Label Printing Software: **Labelstar Office** Version 4.30 Build 1015 or later
Windows Printer Drivers: Carl Valentin Printer Drivers Version 2.3.1 or later
Start/Stop Characters: 0x5E/0x5F

Designing the Label on the PC

Open **Labelstar Office** Version 4.30 Build 1015 or later to design the label layout. To define SAP form fields use $SAPField variables to define named data fields where SAPscript ITF will insert variable data.

For text fields, you must use printer-internal fonts, not TrueType fonts from Windows. This is because TrueType fonts are created in binary form, which SAPscript does not support.

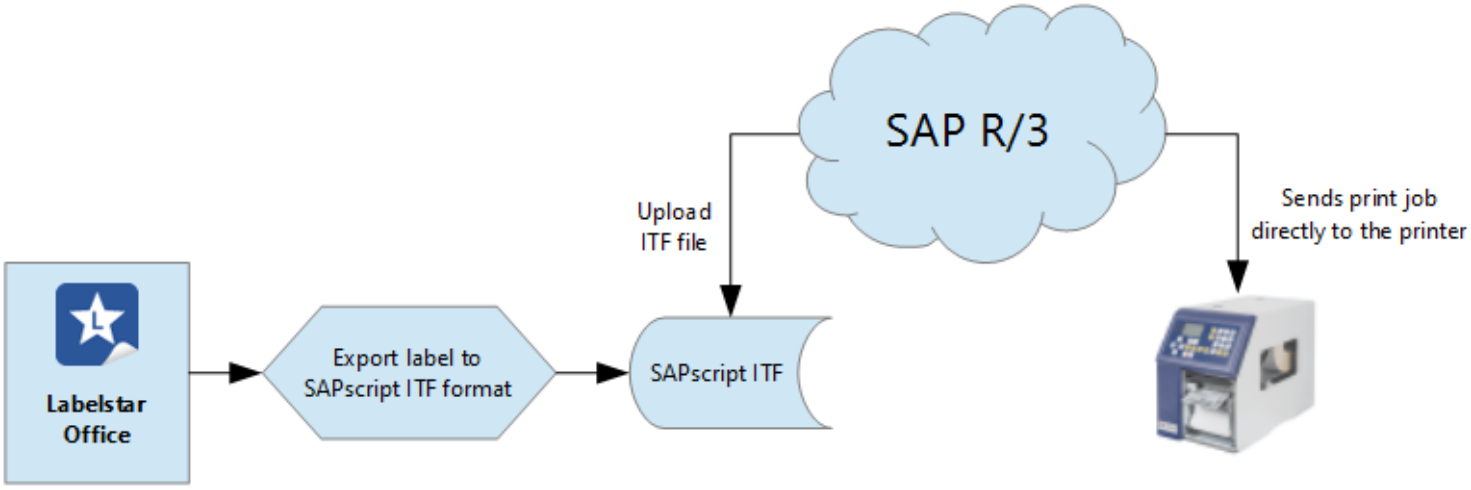
If there are images or text fields which use TrueType fonts on the label, two separate printer code template files are exported. One for the variable data and the other containing formatting information and binary data. Only the file for variable data will be uploaded in SAPscript ITF. The other file is stored on the memory card.

Exporting the Label to SAPscript ITF Printer Code Template File

1. Click the **File** tab, and then click **Export**.
2. Click **Create SAPScipt ITF File**, and the on the right, click **Create ITF**.
3. In the **Browse** dialog box, select a name and location for the file.

Upload the ITF File to SAPscript

Upload the ITF file, generated with **Labelstar Office**, using the SAP R/3 Windows Client. For this, the upload function available for the standard text editor (Transaktion SO10) is used. In this case, however, the editor is only used as a "clipboard" for the file until you insert the file in a Form Window.



Print Only

Labelstar Office provides a wide variety of additional programs with the help of which the labels created in **Labelstar Office** can "only" be printed or with which label printing can be automated.

- **Quick Print**
  Quick print is a practical little program with which you can print labels quickly and simply.

- **Print Manager**
  With the help of the Print Manager you can create print lists.

- **Print Form**
  With this program, the lists created in the Print Manager can be printed.

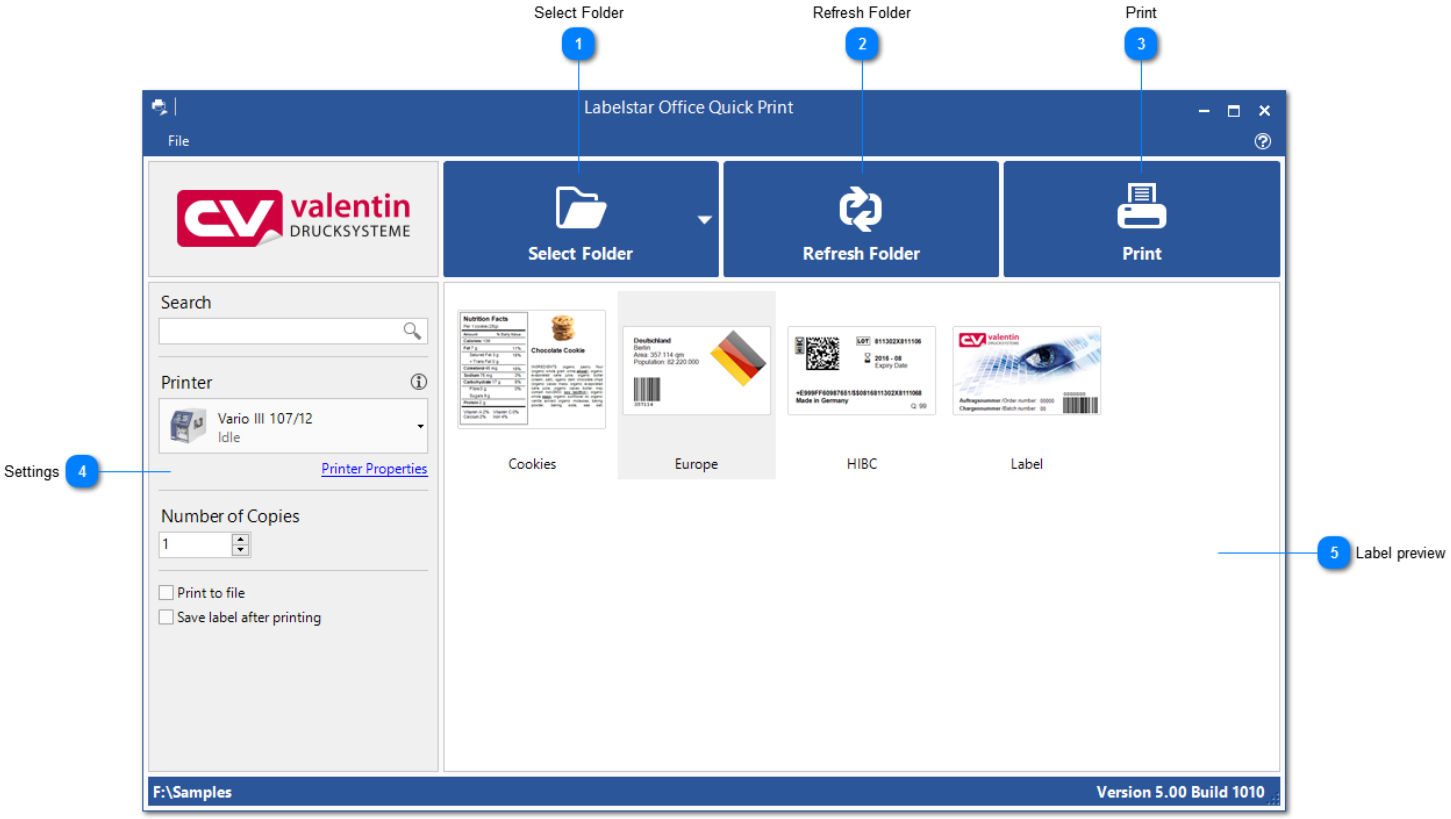
- **Folder Monitoring**
  Label printing can be automated and integrated into its print environment (e.g. LVS system) with the help of Folder Monitoring. The application triggers the printing of one or more labels when a change is made in the monitored directories.

Quick Print

Required program variant **BASIC, PROFESSIONAL (Version 5.00 Build 1010)**

For more information, see Program Variants.

Labelstar Office Quick Print allows you to view all the label (.lbex) files in a specified folder and even print them.



1 Select Folder

Select the directory in which the label files are saved or click ▼ in order to be shown a list of recently used directories.

2 Refresh Folder

Refreshing previews (if a file was added to the folder for example).

3 Print

Prints the selected label on the selected printer.

4 Settings

Search If you want to narrow the number of label previews shown, you can enter a search/filter term in the **Search** box and press ↵. Only label files that contain the entered term will be displayed.

Printer Select the printer on which the selected label should be printed. If a printer is saved for a label then the printer selection will be adjusted when the label is selected. In order to change various print settings, click on **Printer Properties**.

Number of Copies Enter the number of copies which should be printed. If a number of prints is saved for a label then the display will be adjusted and deactivated when the label is selected.

Print to file If this option is activated, the print output is redirected into a file.

Save label after printing If this option is activated, the label will be saved after printing. This is important, for example, if the end value of a counter should be retained.

5 Label preview

Print Manager

Required program variant **PROFESSIONAL (Version 5.00 Build 1010)**

For more information, see Program Variants.

Using the **Print Manager** you can create print lists.

Print Form

Required program variant **PROFESSIONAL (Version 4.40 Build 1010)**

For more information, see Program Variants.

Use this program to open and print labels and print lists.

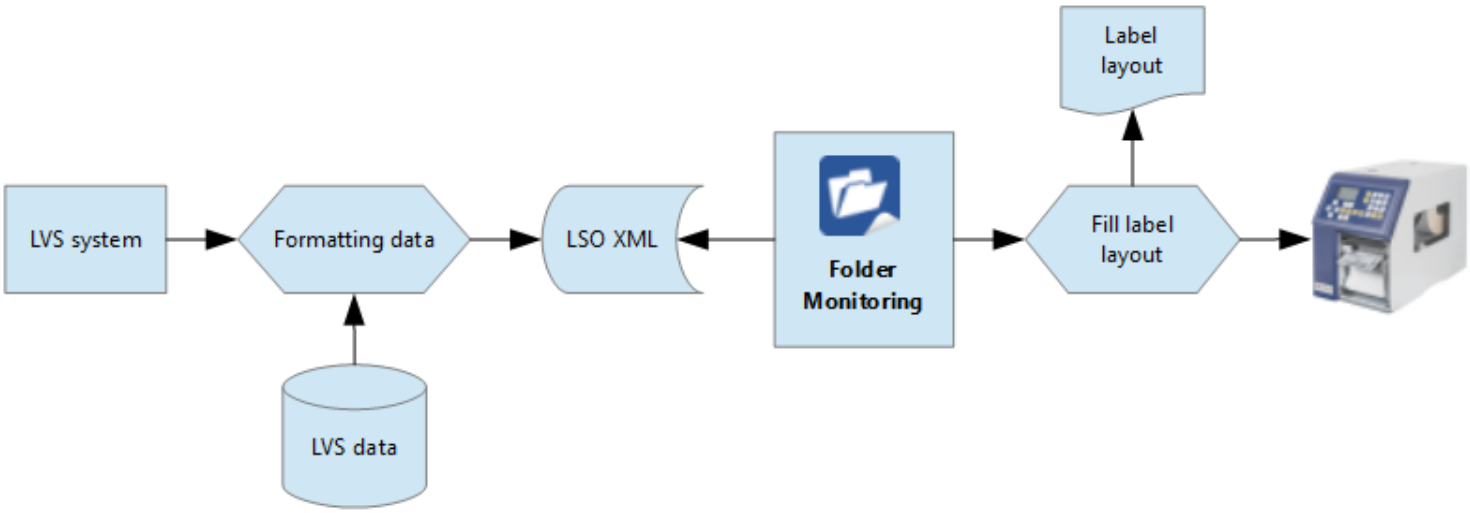
Folder Monitoring

Required program variant **PROFESSIONAL**

For more information, see Program Variants.

With this application you can monitor one or more folders.

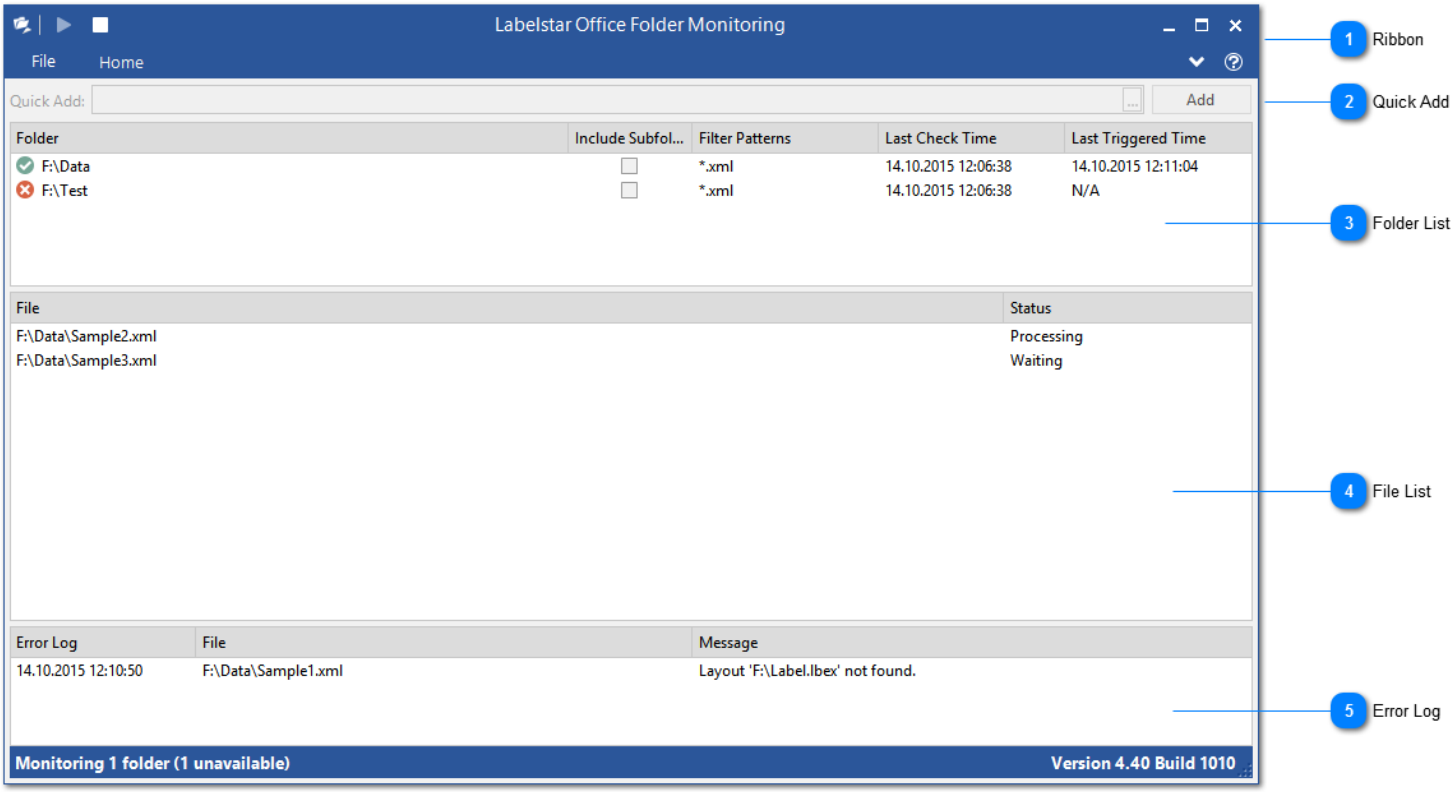
Example: A third-party system (e.g. LVS system) creates a file in LSO XML format with the required data and copies the file to one of the monitored folders. If a change is detected in any of the monitored folders, the application evaluates the XML file, fills the associated label layout and transmits the data to the printer.



Activate Monitoring

1. Click **Start** > **Programs** > **Labelstar Office** > **Print Only** > **Folder Monitoring**
   The program **Labelstar Office Folder Monitoring** opens.
2. To select a folder, use one of the following procedures:
   - **Quick Add:** Click ..., browse for the directory you want to monitor and then click **Add**.
   - On the **Home** tab, in the **Folder** group, click **Add**.
3. On the Quick Access Toolbar, click ▶ to start monitoring.

Main Form



1 Ribbon

Start Monitoring Start monitoring of the selected folders.

Stop Monitoring Stop monitoring of the selected folders.

2 Quick Add

Using the **Quick Add** feature, you can quickly and easily define which folders are to be monitored. Click , browse for the directory you want to monitor and then click **Add**.

3 Folder List

List of all monitored folders. Non-existent or invalid directories are marked with .

4 File List

List of the files which are currently being processed.

5 Error Log

List of the error messages that occur during the processing of a file.

LSO XML File Format

To control printing using **Folder Monitoring**, a file in LSO XML format must be created and copied to one of the monitored folders. In the XML file you can define which labels are to be printed, the number of copies and the printer name. You can also customize the label content.

Sample LSO XML File

```
<?xml version="1.0" encoding="utf-8"?>
<LSO version="1.0">
   <layout>F:\Label.lbex</layout>
   <printer>Vario III 107/12</printer>
   <copies>2</copies>
   <field name="Text1">Example</field>
   <field name="Text2" />
   <field name="EAN">1234567890123</field>
</LSO>
```

More examples are found here.

Supported XML Tags

Tag	Description	Attributes	Available sub-tags
<LSO>	XML top level element.	*version* - 1.0	<layout> <printer> <copies> <field> <label>
<layout>	This tag contains the file name of the label used to create the print job. If no layout is defined, the layout defined in the **Monitored Folder** settings will be used.	None	None
<printer>	This tag contains the printer name. If no printer name is defined, the printer defined in the **Monitored Folder** settings or the printer stored with the label will be used.	None	None
<copies>	This tag contains the number of copies to print. If no number of copies is defined, the number of copies defined in the **Monitored Folder** settings or the number of copies stored with label will be used.	None	None
<field>	This tag contains the field content of a specific field.	*name* - Field name	None
<label>	This tag defines a label. Using this tag multiple labels can be printed consecutively (see also *Sample2.xml*).	None	<layout> <printer> <copies> <field>

Sample

This sample shows how labels can be printed automatically using **Folder Monitoring**.

The example can be found in the samples folder: *%InstallDir%\Samples\Folder Monitoring*.

%InstallDir% refers to the installation directory that was specified during the installation process, what is typicall *C:\Programme (x86)\Carl Valentin GmbH\Labelstar Office*.

Images Image folder
Label.lbex Label layout
Sample1.xml This file prints the label *Label.lbex* twice with the same content.
Sample2.xml This file prints the label *Label.lbex* three times with different content.

To run the sample, proceed as follows:

1. Create a new folder, for example, *C:\Data*
2. Click **Start** > **Programs** > **Labelstar Office** > **Print Only** > **Folder Monitoring**.
   The program **Labelstar Office Folder Monitoring** opens.
3. **Quick Add:** Click , browse for the directory you just created and click **Add**.
4. On the Quick Access Toolbar, click to start monitoring.
5. Copy *Sample1.xml* and *Sample2.xml* to the monitored folder.
6. The labels are printed on the current default printer.

Sample1.xml

This file prints the label *Label.lbex* twice with the same content.

File Content

```
<?xml version="1.0" encoding="utf-8"?>
<LSO version="1.0">
  <layout>%InstallDir%Samples\Folder Monitoring\Label.lbex</layout>
  <copies>2</copies>
  <field name="OrderNumber">P459730</field>
  <field name="BatchNumber">003</field>
  <field name="Deutsch">Zange</field>
  <field name="English">Pliers</field>
  <field name="Francais">Pince</field>
  <field name="Image">%InstallDir%Samples\Folder Monitoring\Images\Pliers.png</field>
  <field name="Barcode">P459730003</field>
</LSO>
```

Printout

Sample2.xml

This file prints the label *Label.lbex* three times with different content.

File Content

```
<?xml version="1.0" encoding="utf-8"?>
<LSO version="1.0">
   <label>
       <layout>%InstallDir%Samples\Folder Monitoring\Label.lbex</layout>
       <copies>1</copies>
       <field name="OrderNumber">H876324</field>
       <field name="BatchNumber">002</field>
       <field name="Deutsch">Hammer</field>
       <field name="English">Hammer</field>
       <field name="Francais">Marteau</field>
       <field name="Image">%InstallDir%Samples\Folder Monitoring\Images\Hammer.png</field>
       <field name="Barcode">H876324002</field>
   </label>
   <label>
       <layout>%InstallDir%Samples\Folder Monitoring\Label.lbex</layout>
       <copies>1</copies>
       <field name="OrderNumber">C128703</field>
       <field name="BatchNumber"/>
       <field name="Deutsch">Messschieber</field>
       <field name="English">Caliper</field>
       <field name="Francais">Pied à coulisse</field>
       <field name="Image">%InstallDir%Samples\Folder Monitoring\Images\Caliper.png</field>
       <field name="Barcode">C128703</field>
   </label>
   <label>
       <layout>%InstallDir%Samples\Folder Monitoring\Label.lbex</layout>
       <copies>1</copies>
       <field name="OrderNumber">B080213</field>
       <field name="BatchNumber">001</field>
       <field name="Deutsch">Schraube</field>
       <field name="English">Bolt</field>
       <field name="Francais">Vies</field>
       <field name="Image">%InstallDir%Samples\Folder Monitoring\Images\Bolt.png</field>
       <field name="Barcode">B080213001</field>
   </label>
</LSO>
```

Printout

B080213 - 001

Schraube
Bolt
Vies

B080213001

C128703

Messschieber
Caliper
Pied à coulisse

C128703

H876324 - 002

Hammer
Hammer
Marteau

H876324002

Command Line Parameters

When starting **Folder Monitoring** using the command line, you can affect the program execution by entering a variety of parameters.

Note

The use of command line parameters is only recommended for experienced users. Command line parameters are not required for normal use.

Syntax

```
[path]FolderMonitor.exe [switch]
```

Parameters

path

The path of the file, either absolute, or relative to the current directory.

switch

switch is used to mark different options:

Notation	Beschreibung
/help	Opens the help file.
/minimized	This parameter prevents the program from being visible on the screen. It will run in minimized form.

Tools

The **Labelstar Office** provides various tools to manage and change program data.

- **License Wizard**
  With this application you can license **Labelstar Office**. For more information, see Licensing.

- **Program Settings**
  With this application, the internal program settings can be changed.

- **Language Settings**
  With this application you can select the language which is used for the user interface (e.g. in the menus and dialog boxes).

Program Settings

- With this application, the internal program settings can be changed.
- By default, the program settings are stored in the directory *C:\ProgramData\Labelstar Office*
- To open the program, click **Start** > **Programs** > **Labelstar Office** > **Tools** > **Program Settings**.

Note

You should not change any setting without having a specific reason to do so. Incorrect settings can make the program unusable.

Language Settings

- With this application you can change the language of the **Labelstar Office** user interface.
- To open the program, click **Start** > **Programs** > **Labelstar Office** > **Tools** > **Language Settings**.

Note

In order to transfer the new settings you have to close all opened **Labelstar Office** applications, close the program and restart it.

Automation Interface

Required program variant **PROFESSIONAL**

For more information, see Program Variants.

Labelstar Office is equipped with an LSOffice automation interface (previously also referred to as OLE Automation). Using this interface, **Labelstar Office** labels can be controlled from external programs.

A complete documentation of the automation interface can be found under Programming Interface.

Note

The automation interface is a core element of **Labelstar Office Professional**. No additional installations or modules are required.

External Access to LSOffice

LSOffice objects can be accessed from external programs with the help of automation programs in VB.Net or C#. Through this interface, labels can be opened, the label content can be edited and they can subsequently be printed and saved.

Samples

In this section, the use of the LSOffice automation interface is demonstrated by means of several practical examples in C#, VB.Net and VBScript.

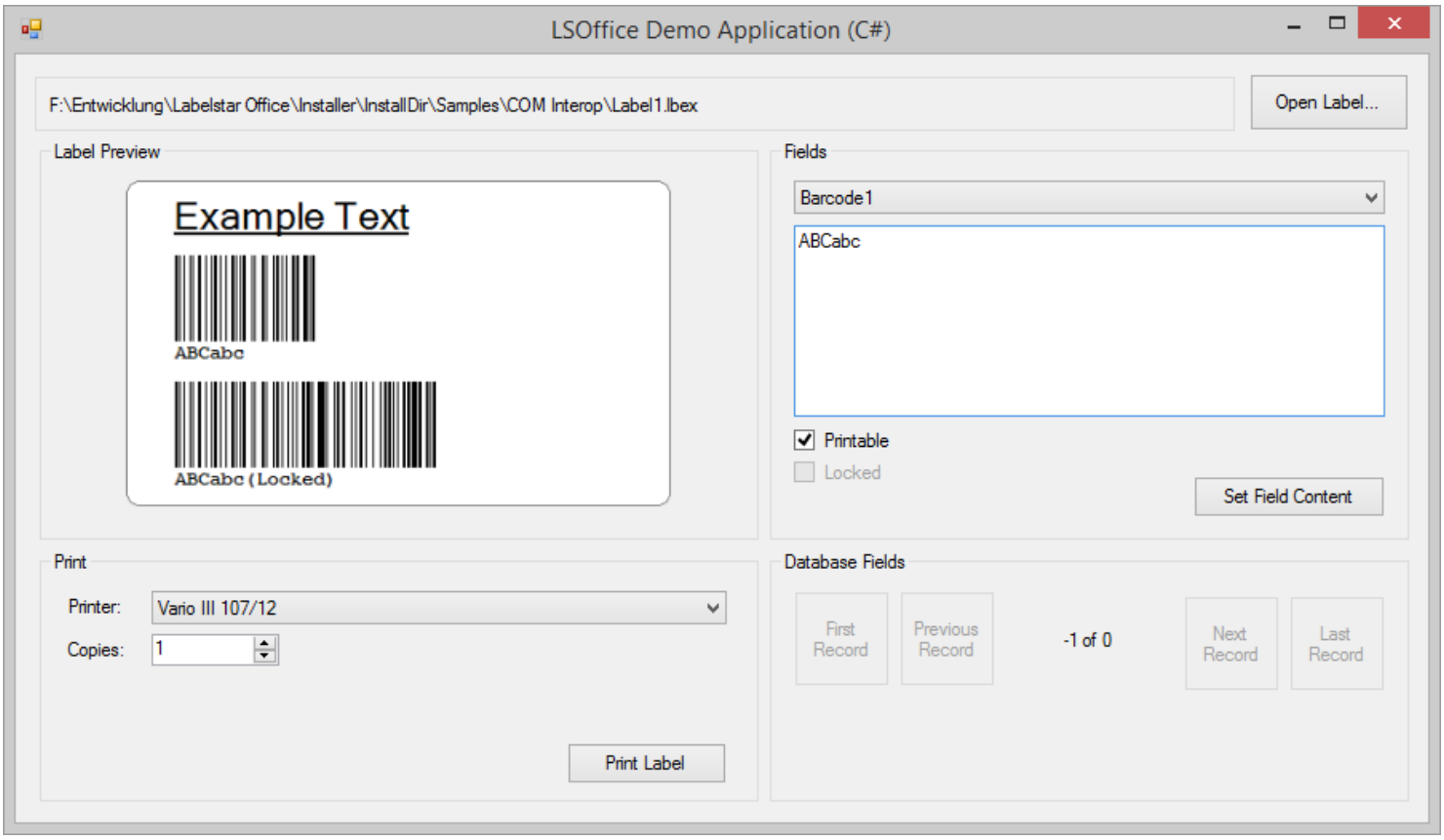
See also

- Sample Project (C#)
- Sample Project (VB.Net)
- VBScript Samples

C#

LSOffice Demo is a Windows Forms application which demonstrates the use of the LSOffice interface by means of several practical examples and is written in C#.

The Visual Studio project (created with Visual Studio 2016) can be found in the following directory: *%InstallDir%\Samples\COM Interop\C#*.



Project Settings

Before compiling, you still need to check the references to external libraries. The most important here is LSOffice.dll, which can be found in the **Labelstar Office** installation folder.

In addition, the following properties must be set:

Property	Setting	
Target framework	.Net Framework 4.6	**Labelstar Office** was compiled with the .Net 4.6 target framework, so the libraries used must also be suitable for this version and this version must be set in the project properties.
Platform target	x86	The automation interface for **Labelstar Office** is a true 32 Bit interface. For 64 Bit systems, the application must be generated as a 32 Bit application.

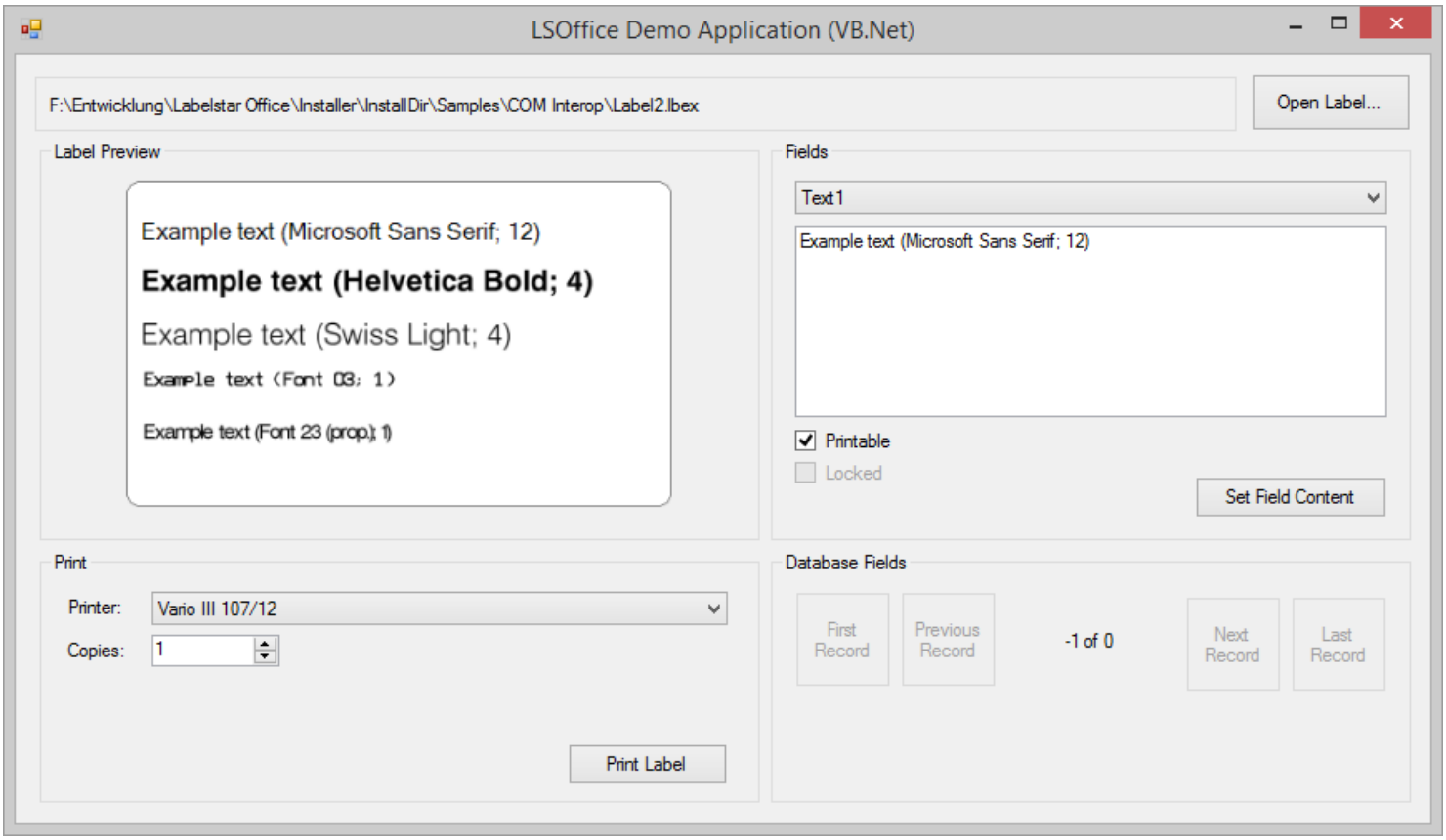
See also

- Sample Project (VB.Net)

VB.Net

LSOffice Demo is a Windows Forms application which demonstrates the use of the LSOffice interface by means of several practical examples and is written in VB.Net.

The Visual Studio project (created with Visual Studio 2016) can be found in the following directory: *%InstallDir%\Samples\COM Interop\VB.Net*.



Project Settings

Before compiling, you still need to check the references to external libraries. The most important here is LSOffice.dll, which can be found in the **Labelstar Office** installation directory.

In addition, the following project properties must be set:

Property	Settings	
Target framework	.Net Framework 4.6	**Labelstar Office** was compiled with the .Net 4.6 target framework, so the libraries used must also be suitable for this version and this version must be set in the project properties.
Target CPU	x86	The automation interface for **Labelstar Office** is a true 32 Bit interface. For 64 Bit systems, the application must be generated as a 32 Bit application.

See also

- Sample Project (C#)

VBScript

Open the **32-bit version** of the Visual Basic Script Editor (e.g. VbsEdit 32-Bit)

Note

Labelstar Office is a 32-bit application. This means the project is intended to run only as a 32-bit process.

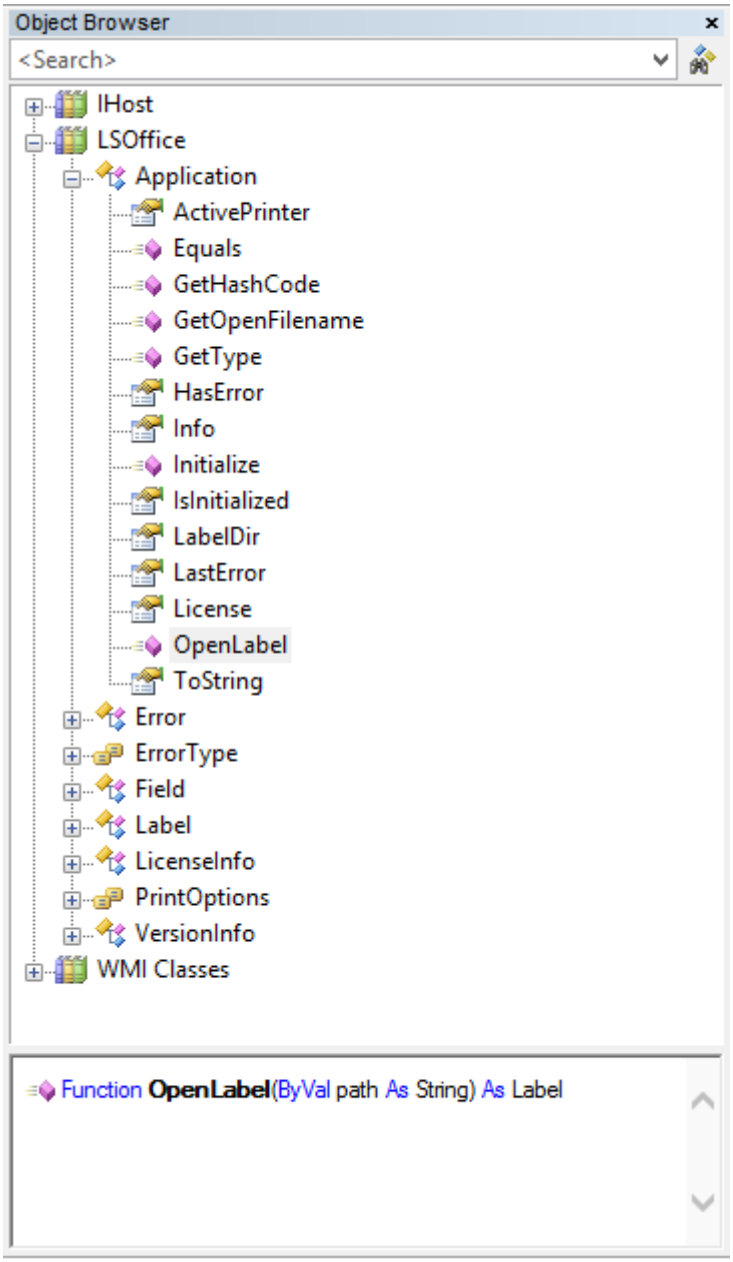
Referencing the Labelstar Office Type Library

To use LSOffice objects in a Visual Basic project, you must first add a reference to the LSOffice type library to the project.

To reference the LSOffice type library, proceed as follows:

1. From the **Tools** menu, select **Reference**.
2. If the interface is already listed as an available refrence, select it, otherwise click **Add/Browse**.
3. Select the LSOffice.tlb file which is located in the installation folder and click **Open**.

Now you can view the LSOffice objects, their methods, properties, parameters, and constant values online using the Visual Basic object browser.



Launch Labelstar Office runtime

Labelstar Office Automation is exposed by the way of a COM object named LSOffice.Application. To call **Labelstar Office**, you first have to create an LSOffice.Application object.

```
Dim objApp
objApp = CreateObject ("LSOffice.Application")
```

The first thing to do now is to initialize the **Labelstar Office** runtime.

```
objApp.Initialize()
```

See also

- VBScript Samples

VBScript Samples

Note

On 64 Bit systems, the example scripts (*.vbs) must be executed as 32 Bit processes and not as 64 Bit processes. By default, 64 Bit Windows starts the 64 Bit version of wscript.exe (script engine) when you double-click on a VBS file. That leads to the "800a01ad Active X component cannot create object" error message.

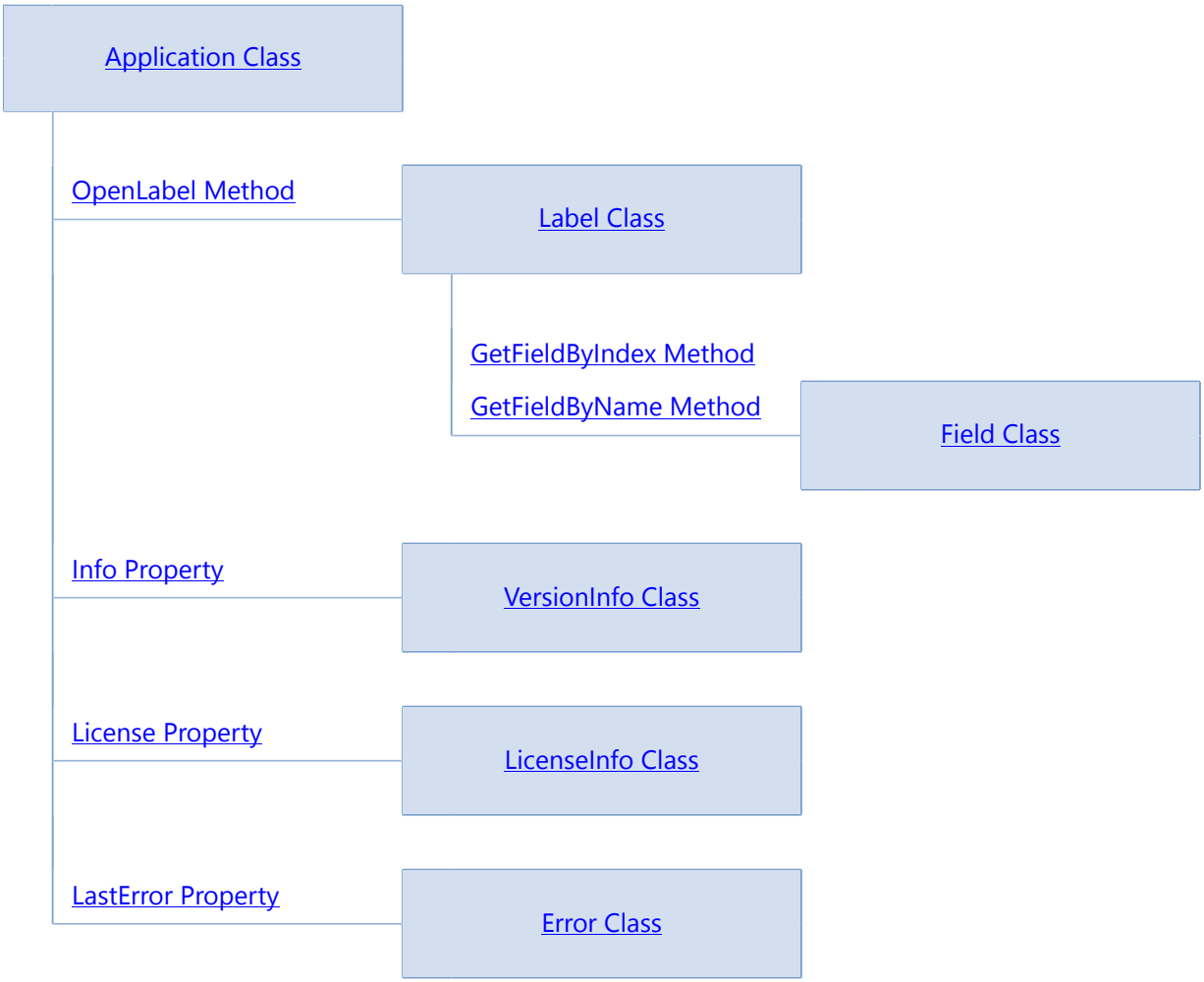
The script samples demonstrate how to open, modify and print labels. The scripts are located in the following directory: *%InstallDir%\Samples\COM Interop\VBScript* and the labels used in the script samples can be found in the following directory: *%InstallDir%\Samples\COM Interop*.

Sample Script	Description	
Open and print label.vbs	Shows a file dialog to choose a label, opens and prints it.	GetOpenFilename method OpenLabel method Print method
Change printer name.vbs	Opens label *label1.lbex* and shows a message box to choose the active printer.	ActivePrinter property
Change field content.vbs	Opens label *label1.lbex* and shows a message box to change the field content of *Text1*.	GetFieldByName method GetContent method SetContent method
Set printable property.vbs	Opens label *label1.lbex* and shows a message box to select if the field *Barcode1* is printed or not.	Printable property
Change text alignment.vbs	Opens label *label4.lbex* and shows a message box to select text alignment of field *Text1*.	SetPropertyValue method
Display field names.vbs	Displays a message box to show the fields defined on *label1.lbex*, *label2.lbex* and *label3.lbex*.	FieldNames property
Display last error.vbs	Demonstrates error handling.	LastError property
Print record.vbs	Opens database label *label3.lbex* and shows a message box to enter search string. **Note:** Data connection *Europe* must be defined in **Labelstar Office**.	SelectRecord method

Programming Interface

The API interface for **Labelstar Office** provides the classes, interfaces and value types which are included in LSOffice.dll. These properties and methods can be used in customer applications, for controlling label printing.

LSOffice Object Hierachy



Application Class

An **Application** object represents the **Labelstar Office** application. This is the top level object from which all other objects will originate. Its members usually apply to **Labelstar Office** as a whole. You can use its properties and methods to control the **Labelstar Office** environment.

You create an **Application** object from scratch like this:

```
Dim objApp
Set objApp = CreateObject("LSOffice.Application")

objApp.Initialize()
```

You must make sure that its lifetime exceeds the lifetimes of all other OLE Automation objects because all the other objects belong to the **Application** object. This means that you almost always delcare the **Application** object at project global scope.

There is no reason for any other project to use more than one **Application** object because any number of other OLE Automation objects can share a single **Application** object.

Properties

	Name	Description
	ActivePrinter	Returns the name of the active printer.
	HasError	Gets a value that indicates whether an error occurred during the last call to a method or property.
	Info	Refers to the VersionInfo object representing the version information.
	IsInitialized	Gets a value that indicates whether the Initialize method has been called.
	LabelDir	Returns the path of the current label folder.
	LastError	Retrieves the calling method's or property's last-error code value.
	License	Refers to the LicenseInfo object representing the license information.

Methods

	Name	Description
	Initialize ()	Initialises the Application object on the basis of the current program settings.
	Initialize (string, string)	Initialises the Application object with the specified license settings.
	GetOpenFilename (string, int, string)	Opens a dialog box for selecting a file.
	GetOpenFilename (string, string, int, string)	Opens a dialog box for selecting a file. In addition, the directory which should be shown when the dialog box opens can be specified.
	OpenLabel	Opens the specified label.

See also

- Programming Interface

Application Properties

The Application type exposes the following members.

Properties

	Name	Description
	ActivePrinter	Returns the name of the active printer.
	HasError	Gets a value that indicates whether an error occurred during the last call to a method or property.
	Info	Refers to the VersionInfo object representing the version information.
	IsInitialized	Gets a value that indicates whether the Initialize method has been called.
	LabelDir	Returns the path of the current label folder.
	LastError	Retrieves the calling method's or property's last-error code value.
	License	Refers to the LicenseInfo object representing the license information.

See also

- Application Class
- Programming Interface

ActivePrinter Property

Returns the name of the active printer. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.ActivePrinter
```

Type

```
String
```

Example (VBScript)

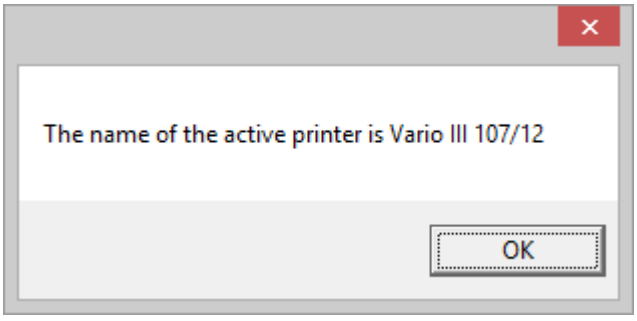
This example displays the name of the active printer.

```
Dim objApp
Set objApp = CreateObject("LSOffice.Application")

objApp.Initialize()

If (objApp.HasError) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

MsgBox "The name of the active printer is " & objApp.ActivePrinter
```



See also

- Application Class
- Programming Interface

HasError Property

Gets a value that indicates whether an error occurred during the last call to a method or property. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.HasError
```

Type

```
Boolean
```

Remarks

For more information, see Application.LastError property.

See also

- Application Class
- Programming Interface

Info Property

Refers to the VersionInfo object representing the version information. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.Info
```

Type

`LSOffice.VersionInfo`

See also

- VersionInfo Class
- Application Class
- Programming Interface

IsInitialized Property

Gets a value that indicates whether the Initialize method has been called. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.IsInitialized
```

Type

```
Boolean
```

Remarks

Use this property to check whether the Application object has already been initialized. The Initialize method should be called once and only once.

See also

- Application Class
- Programming Interface

LabelDir Property

Returns the path of the current label folder. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.LabelDir
```

Type

```
String
```

See also

- Application Class
- Programming Interface

LastError Property

Retrieves the calling method's or property's last-error code value. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.LastError
```

Type

LSOffice.Error

Remarks

After method or property calls on all Automation objects, this property is updated with the return status information of the call. To check whether the request has been successful, check Application.HasError property.

For more information, see Error Codes and Messages.

Example (VBScript)

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Display Last Error Sample Code
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Object variables
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim objApp
Dim objLabel

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Constants LSOffice.ErrorType
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const ErrorType_Success = 0
Const ErrorType_Warning = 1
Const ErrorType_Error = 2

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' DisplayLastError
' Purpose:
'    Shows a message box displaying the last error.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub DisplayLastError()

     If (objApp.LastError.ErrorType = ErrorType_Success) Then
          Exit Sub
     End If

     Dim title
     title = "Error"
```

```
     If (objApp.LastError.ErrorType = ErrorType_Warning) Then
          title = "Message"
     End If

     MsgBox objApp.LastError.Message, vbOKOnly, title

End Sub

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Open and print label
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Set objApp = CreateObject("LSOffice.Application")

' Application must be initialized before OpenLabel is called
objApp.Initialize()
DisplayLastError()

If (objApp.HasError) Then
     WScript.Quit
End If

' Open label
Set objLabel = objApp.OpenLabel("..\Label10.lbex")

If (objLabel is Nothing) Then
     DisplayLastError()
     WScript.Quit
End If
```

See also

- Error Class
- Application Class
- Programming Interface

License Property

Refers to the LicenseInfo object representing the license information. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.License
```

Type

`LSOffice.LicenseInfo`

Remarks

In the trial version an evaluation mode watermark will be put onto each image and all 'e' are replaced by 'x' and all '5' by '0'.

See also

- LicenseInfo Class
- Application Class
- Programming Interface

Application Methods

The Application type exposes the following members.

Methods

	Name	Description
	Initialize ()	Initialises the Application object on the basis of the current program settings.
	Initialize (string, string)	Initialises the Application object with the specified license settings.
	GetOpenFilename (string, int, string)	Opens a dialog box for selecting a file.
	GetOpenFilename (string, string, int, string)	Opens a dialog box for selecting a file. In addition, the directory which should be shown when the dialog box opens can be specified.
	OpenLabel	Opens the specified label.

See also

- Application Class
- Programming Interface

Initialize Method

Initialises the Application object on the basis of the current program settings.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.Initialize ()
```

Remarks

This method ensures that the current instance is properly initialized before it is used to open labels. It should only be called once. Check IsInitialized property before calling the method. This method sets the IsInitialized property to **true**.

Check LastError property to see if the function was completed successfully,

For more information and detailed examples, see here.

See also

- Initialize (string, string) Method
- Application Class
- Programming Interface

Initialize (string, string) Method

Initialises the Application object with the specified license settings.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 5.00.1010

Usage

```
objApp.Initialize (licenseKey, [upgradeCode = null])
```

Parameter

licenseKey
Typ: `String`
License key

upgradeCode (optional)
Typ: `String`
Upgrade code

Remarks

This method ensures that the current instance is properly initialized before it is used to open labels. It should only be called once. Check IsInitialized property before calling the method. This method sets the IsInitialized property to **true**.

Check LastError property to see if the function was completed successfully,

For more information and detailed examples, see here.

See also

- Initialize Method
- Application Class
- Programming Interface

GetOpenFilename Method

Opens a dialog box for selecting a file.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.GetOpenFilename (filter, [filterIndex], [title])
```

Parameters

filter

Type: `String`
A string specifying file filtering criteria.

This string consists of pairs of file filter strings followed by the MS-DOS wildcard file filter specification, with each part and each pair separated by a vertical bar (|). Each separate pair is listed in the **Files of type** drop-down list box. For example, the following string specifies two file filters: "Labels|*.lbex|All Files|*.*".

To use multiple MS-DOS wildcard expressions for a single file filter type, separate the wildcard expressions with semicolons; for example, "Visual Basic Files|*.bas;*.txt".

Note: Do not put spaces before or after the vertical bars in the filter string. This will cause incorrect behavior in the filter.

filterIndex (optional)

Type: `Integer`
Specifies the index numbers of the default file filtering criteria, from 1 to the number of filters specified in *filter*. If this argument is omitted or greater than the number of filters present, the first file filter is used.

title (optional)

Type: `String`
Specifies the title of the dialog box. If this argument is omitted, the title is "Open".

Return Type

`String`

Remarks

Returns the selected file name or the name entered by the user. The returned name may include a path specification. Returns an empty string ("") if the user cancels the dialog box.

This method may change the current drive or folder.

For more information and detailed examples, see VBScript Samples.

See also

- GetOpenFilename (string, string, int, string) Method
- Application Class
- Object Reference

GetOpenFilename (string, string, int, string) Method

Opens a dialog box for selecting a file. In addition, the directory which should be shown when the dialog box opens can be specified.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 5.00.1010

Usage

```
objApp.GetOpenFilename (filter, initialDir, [filterIndex = 1], [title = null])
```

Parameters

filter

Type: `String`
A string specifying file filtering criteria.

This string consists of pairs of file filter strings followed by the MS-DOS wildcard file filter specification, with each part and each pair separated by a vertical bar (|). Each separate pair is listed in the **Files of type** drop-down list box. For example, the following string specifies two file filters: "Labels|*.lbex|All Files|*.*".

To use multiple MS-DOS wildcard expressions for a single file filter type, separate the wildcard expressions with semicolons; for example, "Visual Basic Files|*.bas;*.txt".

Note: Do not put spaces before or after the vertical bars in the filter string. This will cause incorrect behavior in the filter.

initialDir

Type: `String`
Initial directory displayed by the file dialog box.

filterIndex (optional)

Type: `Integer`
Specifies the index numbers of the default file filtering criteria, from 1 to the number of filters specified in *filter*. If this argument is omitted or greater than the number of filters present, the first file filter is used.

title (optional)

Typ: `String`
Specifies the title of the dialog box. If this argument is omitted, the title is "Open".

Return Type

`String`

Remarks

Returns the selected file name or the name entered by the user. The returned name may include a path specification. Returns an empty string ("") if the user cancels the dialog box.

For more information and detailed examples, see VBScript Samples.

See also

- GetOpenFilename Method

- Application Class
- Programming Interface

OpenLabel Method

Opens the specified label.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objApp.OpenLabel (path)
```

Parameters

path
Type: `String`
The path of the label.

Return Type

`LSOffice.Label`

Remarks

Returns a Label object if opened successfully, otherwise **null**. To get extended error information, check the value of the LastError property.

For more information and detailed examples, see here.

See also

- Label Class
- Application Class
- Programming Interface

Error Class

An **Error** object holds information about the current error state. You can get a reference to an **Error** object through the Application.LastError property.

Properties

	Name	Description
	Details	Detailed message describing the error occurred.
	ErrorCode	Numeric code indicating which error occurred.
	ErrorType	Code indicating the type of the error occurred.
	Message	Message describing the error occurred.

See also

- Programming Interface

Error Properties

The Error type exposes the following members.

Properties

	Name	Description
	Details	Detailed message describing the error occurred.
	ErrorCode	Numeric code indicating which error occurred.
	ErrorType	Code indicating the type of the error occurred.
	Message	Message describing the error occurred.

See also

- Error Class
- Programming Interface

Details Property

Detailed message describing the error occurred. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objError.Details
```

Type

```
String
```

Remarks

Returns a representation of the current error that is intended to be understood by humans. The detailed message obtains the Message property, further information about the error, and the stack trace. If any of these members is **null**, its value is not included in the return string.

If there is no error or if it is an empty string (""), then no error message is returned.

See also

- Error Class
- Programming Interface

ErrorCode Property

Numeric code indicating which error occurred. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objError.ErrorCode
```

Type

```
Int32
```

Remarks

If this property is 0, it indicates that no error occured during the last call of a method or property. For more information, see Error Codes and Messages.

See also

- Error Class
- Programming Interface

ErrorType Property

Code indicating the type of the error occurred. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objError.ErrorType
```

Type

`LSOffice.ErrorType`

Remarks

If this property is **Success**, it indicates that no error occurred during the last call of a method or property.

For more information and a detailed example, see Application.LastError property.

See also

- Error Class
- ErrorType Enumeration
- Programming Interface

Message Property

Message describing the error occurred. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objError.Message
```

Type

```
String
```

Remarks

Returns a representation of the current error that is intended to be understood by humans. If there is no error or if it is an empty string (""), then no error message is returned. For more information, see Error Codes and Messages.

For more information and a detailed example, see Application.LastError property.

See also

- Error Class
- Programming Interface

ErrorType Enumeration

Specifies the error types.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Members

	Name	Value	Beschreibung
	Success	0 (0x00)	No error occurred.
	Warning	1 (0x01)	The call succeeded, but there is a potentional problem.
	Error	2 (0x02)	The call failed.

Remarks

The Error.ErrorType property use this enumeration.

For more information and a detailed example, see Application.LastError property.

See also

- Error Class
- Programming Interface

Field Class

A **Field** object represents a field on the label. It can be used to get information about the field, and to get and set its content and properties. You can get a reference to a **Field** object through the Label.GetFieldByIndex or Label.GetFieldByName method.

Properties

	Name	Description
	FieldName	Gets the name of the field.
	Locked	Gets a value that indicates whether the field can be modified.
	Printable	Gets or sets a value that indicates whether the field is printed.

Methods

	Name	Description
	GetContent	Gets the content of the field.
	GetPropertyValue	Gets the value of the specified property.
	SetContent	Sets the content of the field.
	SetPropertyValue	Sets the value of the specified property.

See also

- Programming Interface

Field Properties

The Field type exposes the following members.

Properties

	Name	Description
🔓	FieldName	Gets the name of the field.
🔓	Locked	Gets a value that indicates whether the field can be modified.
	Printable	Gets or sets a value that indicates whether the field is printed.

See also

- Field Class
- Programming Interface

FieldName Property

Gets the name of the field. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objField.FieldName
```

Type

```
String
```

See also

- Field Class
- Programming Interface

Locked Property

Gets a value that indicates whether the field can be modified. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objField.Locked
```

Type

```
Boolean
```

See also

- Field Class
- Programming Interface

Printable Property

Gets or sets a value that indicates whether the field is printed.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objField.Printable
```

Type

```
Boolean
```

Example (VBScript)

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Set Printable Property Sample Code
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Object variables
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim objApp
Dim objLabel
Dim objField

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Set printable property
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Set objApp = CreateObject("LSOffice.Application")

' Application must be initialized before OpenLabel is called
objApp.Initialize()

If (objApp.HasError) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Open label
Set objLabel = objApp.OpenLabel("..\Label1.lbex")

If (objLabel is Nothing) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Get field by name
Set objField = objLabel.GetFieldByName("Barcode1")
```

```
    WScript.Quit
End If

' Set property and print label
Dim msg
msg = MsgBox("Print bar code?", vbYesNo, objField.FieldName)

objField.Printable = (msg = vbYes)
objLabel.Print(1)
```

See also

- Field Class
- Programming Interface

Field Methods

The Field type exposes the following members.

Methods

	Name	Description
	GetContent	Gets the content of the field.
	GetPropertyValue	Gets the value of the specified property.
	SetContent	Sets the content of the field.
	SetPropertyValue	Sets the value of the specified property.

See also

- Field Class
- Programming Interface

GetContent Method

Gets the content of the field.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objField.GetContent()
```

Return Type

```
String
```

Remarks

If the method succeeds it returns the content of the field, otherwise an empty string (""). Call Application.LastError for information about possible errors.

Example (VBScript)

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Change Field Content Sample Code
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Object variables
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim objApp
Dim objLabel
Dim objField

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Change field content
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Set objApp = CreateObject("LSOffice.Application")

' Application must be initialized before OpenLabel is called
objApp.Initialize()

If (objApp.HasError) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Open label
Set objLabel = objApp.OpenLabel("..\Label1.lbex")
```

```
' Get field by name
Set objField = objLabel.GetFieldByName("Text1")

If (objField Is Nothing) Then
     WScript.Echo objApp.LastError.Message
    WScript.Quit
End If

' Enter new field content
Dim result
result = InputBox("Field content:", objField.FieldName, objField.GetContent())

' Evaluate the user input
If result <> "" Then
     ' Set field content and print label
     objField.SetContent(result)
     objLabel.Print(1)
End If
```

See also

- Field Class
- Programming Interface

GetPropertyValue Method

Gets the value of the specified property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objField.GetPropertyValue(propertyName)
```

Parameters

propertyName
Type: `String`
The name of the property.

The following table describes some possible property names.

Property name	Description
Printable	Type: `Boolean` false or 0: field is not printed true or 1: field is printed See also: Printable Property
Locked	Type: `Boolean` false or 0: field is not locked true or 1: field is locked See also: Locked Property
TextAlignment	**Text/Barcode only** Type: `Integer` 0: Left 1: Center 2: Right 3: Justify
HumanReadable	**Barcode only** Type: `Boolean` false or 0: Don't show plain text true or 1: Show plain text

Return Type

`Object`

Remarks

If the method succeeds it returns the value of the property, otherwise **null**. Call Application.LastError for information about possible errors.

Example (VBScript)

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Change Text Alignment Sample Code
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Object variables
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim objApp
Dim objLabel
Dim objField

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Select text alignment
' Purpose:
'    Displays a message box to select the text alignment option.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function SelectTextAlignment

     SelectTextAlignment = -1

     Dim text

     text = "Text alignment:" & vbCrLf & vbCrLf
     text = text & "0" & vbTab & "Left aligned" & vbCrLf
     text = text & "1" & vbTab & "Centered" & vbCrLf
     text = text & "2" & vbTab & "Right aligned"

     ' Show all available printers and allow a user selection
     Dim tmp
     tmp = InputBox(text, "Select text alignment", "0")

     If tmp = "" Then
         WScript.Echo "No user input, aborted"
         Exit Function
     End If

     tmp = CInt(tmp)

     If (tmp < 0) Or (tmp > 2) Then
         WScript.Echo "Wrong value, aborted"
         Exit Function
     End If

     ' Set text alignment
     SelectTextAlignment = tmp

End Function

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Change text alignment
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Set objApp = CreateObject("LSOffice.Application")

' Application must be initialized before OpenLabel is called
objApp.Initialize()
```



```
If (objApp.HasError) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Open label
Set objLabel = objApp.OpenLabel("..\Label4.lbex")

If (objLabel is Nothing) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Get field by name
Set objField = objLabel.GetFieldByName("Text1")

If (objField is Nothing) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Select text alignment
Dim textAlignment
textAlignment = SelectTextAlignment()

If (textAlignment < 0) Then
     WScript.Quit
End If

' Set text alignment and print label
objField.SetPropertyValue "TextAlignment", textAlignment
objLabel.Print(1)
```

See also

- Field Class
- Programming Interface

SetContent Method

Sets the content of the field.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objField.SetContent(content)
```

Parameters

content
Type: `String`
The new content.

Remarks

Call Application.LastError for information about possible errors.

For more information and a detailed example, see Field.GetContent method.

See also

- Field Class
- Programming Interface

SetPropertyValue Method

Sets the value of the specified property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objField.SetPropertyValue(propertyName, value)
```

Parameters

propertyName

Type: `String`
The name of the property.

The following table describes some possible property names.

Property name	Description
Printable	Type: `Boolean` false or 0: field is not printed true or 1: field is printed See also: Printable property
Locked	Type: `Boolean` false or 0: field is not locked true or 1: field is locked See also: Locked property
TextAlignment	**Text/Barcode only** Type: `Integer` 0: Left 1: Center 2: Right
HumanReadable	**Barcode only** Type: `Boolean` false or 0: Don't show plain text true or 1: Show plain text

value

Type: `Object`
The value to set the property to.

Remarks

Call Application.LastError for information about possible errors.

For more information and a detailed example, see Field.GetPropertyValue method.

See also

- Field Class
- Programming Interface

ImageFormat Enumeration

Specifies file save format.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.20.1040

Members

	Name	Value	Description
	Bmp	0 (0x00)	Saves the image as a bitmap (BMP).
	Gif	1 (0x01)	Saves the image using the GIF image format.
	Jpeg	2 (0x02)	Saves the image using the Joint Photographic Experts Group (JPEG) image format.
	Png	3 (0x03)	Saves the image using the W3C Portable Network Graphics (PNG) image format.

See also

- SavePreview Method
- Programming Interface

Label Class

A **Label** object represents an opened label. You can get a reference to a **Label** object by calling the Application.OpenLabel method.

Properties

	Name	Description
	ActivePrinter	Gets or sets the name of the active printer.
	CurrentRecord	Gets or sets the one-based index of the current record to be printed.
🔒	FieldCount	Gets the number of fields defined on the label.
🔒	FieldNames	Gets the list of field names defined on the label.
🔒	IsDataAvailable	Determines if there are database fields defined on the label.
🔒	LabelPath	Gets the path to the opened label.
🔒	MaxRecord	Returns the maximum number of records in the database.
🔒	Modified	Gets a value that indicates that the label has been modified.
	PageName	Gets or sets the current page name.

Methods

	Name	Description
	GetFieldByIndex	Gets the field with the given index.
	GetFieldByName	Searches for the field with the specified name.
	GetPreview	Retrieves a preview image of the current label content.
	GetPropertyValue	Gets the value of the specified property.
	Print	Prints the label.
	PrintToFile	Prints the label to a file.
	Save	Saves changes to the label.
	SaveAs	Saves changes to the label in a different file.
	SavePreview	Saves a preview of the current label.
	SelectRecord	Sets the current record to the first record matching the filter expression.
	SetPropertyValue	Sets the value of the specified property.

See also

- Programming Interface

Label Properties

The Label type exposes the following members.

Properties

	Name	Description
	ActivePrinter	Gets or sets the name of the active printer.
	CurrentRecord	Gets or sets the one-based index of the current record to be printed.
🔓	FieldCount	Gets the number of fields defined on the label.
🔓	FieldNames	Gets the list of field names defined on the label.
🔓	IsDataAvailable	Determines if there are database fields defined on the label.
🔓	LabelPath	Gets the path to the opened label.
🔓	MaxRecord	Returns the maximum number of records in the database.
🔓	Modified	Gets a value that indicates that the label has been modified.
	PageName	Gets or sets the current page name.

See also

- Label Class
- Programming Interface

ActivePrinter Property

Gets or sets the name of the active printer.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.ActivePrinter
```

Type

```
String
```

Example (VBScript)

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Change Printer Name Sample Code
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Object variables
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim objApp
Dim objLabel

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Select printer
' Purpose:
'    Displays a message box to select one of the available printers.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Function SelectPrinter(activePrinter)

     SelectPrinter = ""

     ' Read all printers
     Dim wshNetwork, objPrinters
     Set wshNetwork = WScript.CreateObject("WScript.Network")
     Set objPrinters = wshNetwork.EnumPrinterConnections

     Dim text, i, j, index

     text = "Available printers:" & vbCrLf & vbCrLf
     j = objPrinters.Count
     index = 0

     For i = 0 To j - 1 Step 2

          If (objPrinters(i+1) = activePrinter) Then
               index = i/2
          End If

          text = text & (i/2) & vbTab
```

```
            text = text & objPrinters(i+1) & vbCrLf

        Next

        ' Show all available printers and allow a user selection
        Dim tmp
        tmp = InputBox(text, "Select printer", index)

        If tmp = "" Then
            WScript.Echo "No user input, aborted"
            Exit Function
        End If

        tmp = CInt(tmp)

        If (tmp < 0) Or (tmp > (j/2 - 1)) Then
            WScript.Echo "Wrong value, aborted"
            Exit Function
        End If

        ' Set printer name
        SelectPrinter = objPrinters(tmp*2 + 1)

    End Function

    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Change printer name
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Set objApp = CreateObject("LSOffice.Application")

    ' Application must be initialized before OpenLabel is called
    objApp.Initialize()

    If (objApp.HasError) Then
        WScript.Echo objApp.LastError.Message
        WScript.Quit
    End If

    ' Open label
    Set objLabel = objApp.OpenLabel("..\Label1.lbex")

    If (objLabel is Nothing) Then
        WScript.Echo objApp.LastError.Message
        WScript.Quit
    End If

    ' Select printer
    Dim activePrinter
    activePrinter = SelectPrinter(objLabel.ActivePrinter)

    If (activePrinter = "") Then
        WScript.Quit
    End If

    ' Set active printer and print label
    objLabel.ActivePrinter = activePrinter
    objLabel.Print(1)
```

See also

- Label Class
- Programming Interface

CurrentRecord Property

Gets or sets the one-based index of the current record to be printed.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.CurrentRecord
```

Type

```
Integer
```

Remarks

To check if database fields are defined on the label use IsDataAvailable property.

See also

- Label Class
- Programming Interface

FieldCount Property

Gets the number of fields defined on the label. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.FieldCount
```

Type

```
Integer
```

See also

- Label Class
- Programming Interface

FieldNames Property

Gets the list of field names defined on the label. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.FieldNames
```

Type

```
String[]
```

Example (VBScript)

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Display Field Names Sample Code
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Object variables
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim objApp

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' DisplayFieldNames
' Purpose:
'    A message box is displayed showing all fields defined on the label.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Sub DisplayFieldNames(labelName)

     ' Open label
     Dim objLabel
     Set objLabel = objApp.OpenLabel(labelName)

     If (objLabel is Nothing) Then
          WScript.Echo objApp.LastError.Message
          Exit Sub
     End If

     ' Format field names
     Dim fieldNames
     fieldNames = objLabel.FieldNames

     Dim text, i, j

     j = UBound(fieldNames)

     text = "Available fields:" & vbCrLf & vbCrLf

     For i = 0 To j Step 1
          text = text & i & vbTab
```

```
            text = text & fieldNames(i) & vbCrLf
        Next

        MsgBox labelName & vbCrLf & vbCrLf & text, vbOKOnly, "Field names"

    End Sub

    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Display field names
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Set objApp = CreateObject("LSOffice.Application")

    ' Application must be initialized before OpenLabel is called
    objApp.Initialize()

    If (objApp.HasError) Then
        WScript.Echo objApp.LastError.Message
        WScript.Quit
    End If

    ' Display field names
    DisplayFieldNames "..\Label1.lbex"
    DisplayFieldNames "..\Label2.lbex"
    DisplayFieldNames "..\Label3.lbex"
```

See also

- Label Class
- Programming Interface

IsDataAvailable Property

Determines if there are database fields defined on the label. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.IsDataAvailable
```

Type

```
Boolean
```

See also

- Label Class
- Programming Interface

LabelPath Property

Gets the path to the opened label. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.LabelPath
```

Type

```
String
```

See also

- Label Class
- Programming Interface

MaxRecord Property

Returns the maximum number of records in the database. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.MaxRecord
```

Type

```
Integer
```

Remarks

Returns the maximum number of records in the database, or 0 if no database fields are defined on the label. To check if database fields are defined on the label use IsDataAvailable property.

See also

- Label Class
- Programming Interface

Modified Property

Gets a value that indicates that the label has been modified. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.Modified
```

Type

```
Boolean
```

See also

- Label Class
- Programming Interface

PageName Property

Gets or sets the current page name.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.20.1040

Usage

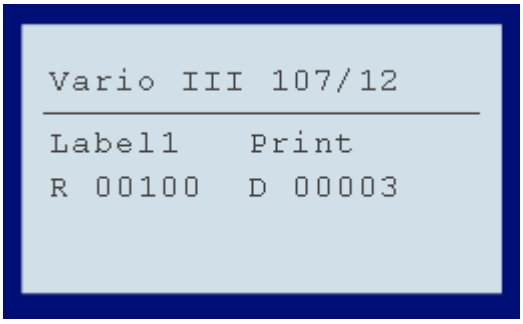
```
objLabel.PageName
```

Type

```
String
```

Remarks

The page name is displayed in the printer display during printing.



See also

- Label Class
- Programming Interface

Label Methods

The Label type exposes the following members.

Methods

	Name	Description
	GetFieldByIndex	Gets the field with the given index.
	GetFieldByName	Searches for the field with the specified name.
	GetPreview	Retrieves a preview image of the current label content.
	GetPropertyValue	Gets the value of the specified property.
	Print	Prints the label.
	PrintToFile	Prints the label to a file.
	Save	Saves changes to the label.
	SaveAs	Saves changes to the label in a different file.
	SavePreview	Saves a preview of the current label.
	SelectRecord	Sets the current record to the first record matching the filter expression.
	SetPropertyValue	Sets the value of the specified property.

See also

- Label Class
- Programming Interface

GetFieldByIndex Method

Gets the field with the given index.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.GetField (index)
```

Parameters

index
Type: `Integer`
Zero-based index of the field.

Return Type

`LSOffice.Field`

Remarks

This method returns a Field object on the indicated field, if found; otherwise, **null**. To get extended error information, check the value of the Application.LastError property.

See also

- GetFieldByName Method
- Label Class
- Programming Interface

GetFieldByName Method

Searches for the field with the specified name.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.GetField (fieldName)
```

Parameters

fieldName
Type: `String`
The string containing the name of the field to get.

Return Type

`LSOffice.Field`

Remarks

Returns a Field object representing the field with the specified name, if found; otherwise, **null**. To get extended error information, check the value of the Application.LastError property.

For more information and a detailed example, see Field.GetContent method.

See also

- GetFieldByIndex Method
- Label Class
- Programming Interface

GetPreview Method

Retrieves a preview image of the current label content.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.GetPreview ()
```

Return Type

```
object
```

Remarks

Returns a **Bitmap** object representing the preview, if a valid label is opened; otherwise, **null**.

See also

- Label Class
- Programming Interface

GetPropertyValue Method

Gets the value of the specified property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.20.1040

Usage

```
objLabel.GetPropertyValue (propertyName)
```

Parameter

propertyName
Type: `String`
The name of the property.

The following table describes some possible property names.

Property name	Beschreibung
LabelRotation	Type: `Integer` 0, 90, 180, 270
LabelType	Type: `Integer` 0 = Adhesive labels 1 = Continuous labels

Return type

`Object`

See also

- Label Class
- Programming Interface

Print Method

Prints the label.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.Print (copies, [options])
```

Parameters

copies
Type: `Integer`
Number of copies to print. If *copies* > 0 the **Print** dialog box will not be shown.

options (optional)
Type: `LSOffice.PrintOptions`
Print options

Remarks

Check Application.LastError property to see if the function was completed successfully,

For more information and detailed examples, see here.

See also

- Label Class
- Programming Interface

PrintToFile Method

Prints the label to a file.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.20.1040

Usage

```
objLabel.PrintToFile (fileName, copies, [options])
```

Parameters

fileName
Type: `String`
File name

copies
Type: `Integer`
Number of copies to print. If *copies* > 0 the **Print** dialog box will not be shown.

options (optional)
Type: `LSOffice.PrintOptions`
Print options

See also

- Label Class
- Programming Interface

Save Method

Saves changes to the label.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.Save ()
```

Remarks

Check Application.LastError property to see if the function was completed successfully,

---

See also

- Label Class
- Programming Interface

SaveAs Method

Saves changes to the label in a different file.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.SaveAs (path)
```

Parameters

path
Type: `String`
The name of the file to be saved.

Remarks

Check Application.LastError property to see if the function was completed successfully,

---

See also

- Label Class
- Programming Interface

SavePreview Method

Saves a preview of the current label.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.20.1040

Usage

```
objLabel.SavePreview (fileName, [format])
```

Parameters

fileName
Type: `String`
File name

format (optional, Standard = Bmp)
Type: `LSOffice.ImageFormat`
File format

See also

- Label Class
- Programming Interface

SelectRecord Method

Sets the current record to the first record matching the filter expression.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLabel.SelectRecord (filterExpression)
```

Parameters

filterExpression
Type: `String`
The criteria to use to filter the records. For examples on how to filter records, see Filter Expression Syntax.

Remarks

Check Application.LastError property to see if the function was completed successfully,

Example (VBScript)

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Print Record Sample Code
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Object variables
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim objApp
Dim objLabel

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Constants LSOffice.PrintOptions
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const PrintOptions_PrintCurrentRecord = 1
Const PrintOptions_PrintAllRecords = 2
Const PrintOptions_Default = 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Print record
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Set objApp = CreateObject("LSOffice.Application")

' Application must be initialized before OpenLabel is called
objApp.Initialize()

If (objApp.HasError) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Open label
```

```
Set objLabel = objApp.OpenLabel("..\Label3.lbex")

If (objLabel is Nothing) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Enter selection string
Dim tmp
tmp = InputBox("Native name starts with:" & vbCrLf & vbCrLf & "de" & vbTab &
"Deutschland" & vbCrLf & "fr" & vbTab & "France" & vbCrLf & "it" & vbTab & "Italia"
& vbCrLf & "es" & vbTab & "Espana" & vbCrLf & "..." & vbCrLf, "Select Record",
"de")

If tmp = "" Then
    WScript.Echo "No user input, aborted"
    WScript.Quit
End If

' Select record and print label
objLabel.SelectRecord("NativeName LIKE '" & tmp & "%'")

If (objApp.HasError) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

objLabel.Print 1, PrintOptions_PrintCurrentRecord
```

See also

- Label Class
- Programming Interface

Filter Expression Syntax

This example describes syntax of filter expression. It shows how to correctly build aexpression string (without „SQL injection“) using methods to escape values.

Column Names

If a column name contains any of these special characters ~ ( ) # \ / = > < + - * % & | ^ ' " [ ], you must enclose the column name within square brackets [ ]. If a column name contains right bracket] or backslash \, escape it with backslash (\] or \\).

Example	Description
Filter = "id = 10"	No special character in column name "id".
Filter = "$id = 10"	No special character in column name "$id".
Filter = "[#id] = 10"	Special character "#" in column name "#id".
Filter = "[[id\]] = 10"	Special characters in column name "[id]".

Literals

String values are enclosed within single quotes ' '. If the string contains single quote ', the quote must be doubled.

Example	Description
Filter = "Name = 'John'"	String value.
Filter = "Name = 'John ''A'''"	String with single quotes "John 'A'".

Number values are not enclosed within any characters. The values should be formatted in English locale format.

Example	Description
Filter = "Year = 2008"	Integer value.
Filter = "Price = 1199.9"	Float value.

Date values are enclosed within sharp characters # #. The date format is the same as for English culture.

Example	Description
Filter = "Date = #12/31/2008#"	Date value (time is 00:00:00).
Filter = "Date = #2008-12-31#"	Also this format is supported.
Filter = "Date = #12/31/2008 16:44:58#"	Date and time value.

Alternatively you can enclose all values within single quotes ' '. It means you can **use string values** for numbers or date time values. In this case the current culture is used to convert the string to the specific value.

Example	Description
Filter = "Date = '12/31/2008 16:44:58'"	Current culture is English.
Filter = "Date = '31.12.2008 16:44:58'"	Current culture is German.
Filter = "Price = '1199.90'"	Current culture is English.
Filter = "Price = '1199,90'"	Current culture is German.

Comparison Operators

Equal, not equal, less, greater operators are used to include only values that suit to a comparison expression. You can use these operators = <> < <= > >=.

Note: String comparison is culture-sensitive.

Example	Description
Filter = "Num = 10"	Number is equal to 10.
Filter = "Date < #1/1/2008#"	Date is less than 1/1/2008.
Filter = "Name <> 'John'"	String is not equal to 'John'.
Filter = "Name >= 'Jo'"	String comparison.

Operator IN is used to include only values from the list. You can use the operator for all data types, such as numbers or strings.

Example	Description
Filter = "Id IN (1, 2, 3)"	Integer values.
Filter = "Price IN (1.0, 9.9, 11.5)"	Float values.
Filter = "Name IN ('John', 'Jim', 'Tom')"	String values.
Filter = "Date IN (#12/31/2008#, #1/1/2009#)"	Date time values.
Filter = "Id NOT IN (1, 2, 3)"	Values not from the list.

Operator LIKE is used to include only values that match a pattern with wildcards. **Wildcard** character is * or %, it can be at the beginning of a pattern '*value', at the end 'value*', or at both '*value*'. Wildcard in the middle of a patern 'va*lue' is **not allowed**.

Example	Description
Filter = "Name LIKE 'j*'"	Values that start with 'j'.
Filter = "Name LIKE '%jo%'"	Values that contain 'jo'.
Filter = "Name NOT LIKE 'j*'"	Values that don't start with 'j'.

If a pattern in a LIKE clause contains any of these special characters * % [ ], those characters must be escaped in brackets [ ] like this [*], [%], [[] or []].

Example	Description
Filter = "Name LIKE '[*]*'"	Values that starts with '*'.
Filter = "Name LIKE '[[]*'"	Values that starts with '['.

Boolean Operators

Boolean operators **AND, OR** and **NOT** are used to concatenate expressions. Operator NOT has precedence over AND operator and it has precedence over OR operator.

Example	Description
Filter = "City = 'Tokyo' AND (Age < 20 OR Age > 60)"	Operator AND has precedence over OR operator, parenthesis are needed.
Filter = "City <> 'Tokyo' AND City <> 'Paris'"; Filter = "NOT City = 'Tokyo' AND NOT City = 'Paris'"; Filter = "NOT (City = 'Tokyo' OR City = 'Paris')"; Filter = "City NOT IN ('Tokyo', 'Paris')";	These examples do the same.

Arithmetic and String Operators

Arithmetic operators are addition +, subtraction -, multiplication *, division / and modulus %.

Example	Description
Filter = "MotherAge - Age < 20"	People with young mother.
Filter = "Age % 10 = 0"	People with decennial birthday.

There is also one **string** operator **concatenation** **+**.

Parent-Child Relation Referencing

A **parent table** can be referenced in an expression using parent column name with `Parent.` prefix. A column in a **child table** can be referenced using child column name with `Child.` prefix.

The reference to the child column must be in an aggregate function because child relationships may return multiple rows. For example expression `SUM(Child.Price)` returns sum of all prices in child table related to the row in parent table.

If a table has more than one child relation, the prefix must contain relation name. For example expression `Child(OrdersToItemsRelation).Price` references to column Price in child table using relation named OrdersToItemsRelation.

Aggregate Functions

There are supported following aggregate functions **SUM, COUNT, MIN, MAX, AVG** (average), **STDEV** (statistical standard deviation) and **VAR** (statistical variance).

Example	Description
Filter = "Salary > AVG(Salary)"	Select people with above-average salary.
Filter = "COUNT(Child.IdOrder) > 5"	Select orders which have more than 5 items.
Filter = "SUM(Child.Price) >= 500"	Select orders which total price (sum of items prices) is greater or equal $500.

Functions

There are also supported following functions. Detailed description can be found here Filter Expression Functions.

- `CONVERT` – converts particular expression to a specified type
- `LEN` – gets the length of a string
- `ISNULL` – checks an expression and either returns the checked expression or a replacement value
- `IIF` – gets one of two values depending on the result of a logical expression
- `TRIM` – removes all leading and trailing blank characters like \r, \n, \t, , ‘
- `SUBSTRING` – gets a sub-string of a specified length, starting at a specified point in the string

---

See also

- Filter Expression Functions
- SelectRecord Method
- Label Class
- Programming Interface

Filter Expression Functions

The following functions are supported:

CONVERT

Description	Converts particular expression to a specified type.
Syntax	CONVERT (*expression*, *type*)
Arguments	*expression* -- The expression to convert. *type* -- The type to which the value will be converted.
Example	Expression = "Convert (total, 'System.Int32')"

All conversions are valid with the following exceptions: **Boolean** can be coerced to and from **Byte**, **SByte**, **Int16**, **Int32**, **Int64**, **UInt16**, **UInt32**, **UInt64**, **String** and itself only. **Char** can be coerced to and from **Int32**, **UInt32**, **String**, and itself only. **DateTime** can be coerced to and from **String** and itself only. **TimeSpan** can be coerced to and from **String** and itself only.

LEN

Description	Gets the length of a string.
Syntax	LEN (*expression*)
Arguments	*expression* -- The expression to evaluated.
Example	Expression = "Len (item)"

ISNULL

Description	Checks an expression and either returns the checked expression or a replacement value.
Syntax	ISNULL (*expression*, *replacementvalue*)
Arguments	*expression* -- The expression to check. *replacementvalue* -- If expression is **Nothing**, replacementvalue is returned.
Example	Expression = "IsNull (price, -1)"

IIF

Description	Gets one of two values depending on the result of a logical expression.
Syntax	IIF (*expression*, *truepart*, *falsepart*)
Arguments	*expression* -- The expression to evaluated. *truepart* -- The value to return if the expression is true. *falsepart* -- The value to return if the expression is false.
Example	Expression = "IIF (total>1000, 'expensive', 'dear')"

TRIM

Description	Removes all leading and trailing blank characters like \r, \n, \t, ' '.
Syntax	TRIM (*expression*)
Arguments	*expression* -- The expression to trim.

SUBSTRING

Description	Gets a sub-string of a specified length, starting at a specified point in the string.
Syntax	SUBSTRING (*expression*, *start*, *length*)

Arguments	*expression* -- The source string for the substring. *start* -- Integer that specifies where the substring starts. *length* -- Integer that specifies the length of the substring.
Example	Expression = "SubString (phone, 7, 8)"

See also

- Filter Expression Syntax
- SelectRecord Method
- Label Class
- Programming Interface

SetPropertyValue Method

Sets the value of the specified property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.20.1040

Usage

```
objLabel.SetPropertyValue (propertyName, value)
```

Parameter

propertyName
Type: `String`
The name of the property.

The following table describes some possible property names.

Property name	Beschreibung
LabelRotation	Type: `Integer` 0, 90, 180, 270
LabelType	Type: `Integer` 0 = Adhesive labels 1 = Continuous labels

value
Type: `Object`
The value to set the property to.

See also

- Label Class
- Programming Interface

LicenseInfo Class

A **LicenseInfo** object represents the license information. You can get a reference to a **LicenseInfo** object through the Application.License property.

Properties

	Name	Description
	IsTrialVersion	Gets a value that indicates whether the application is in trial mode.
	LicenseKey	Gets the license key used to to activate **Labelstar Office**.
	LicenseType	Gets the license type.
	UpgradeCode	Gets the upgrade code used to activate **Labelstar Office**.

See also

- Programming Interface

LicenseInfo Properties

The LicenseInfo type exposes the following members.

Properties

	Name	Description
	IsTrialVersion	Gets a value that indicates whether the application is in trial mode.
	LicenseKey	Gets the license key used to to activate **Labelstar Office**.
	LicenseType	Gets the license type.
	UpgradeCode	Gets the upgrade code used to activate **Labelstar Office**.

See also

- LicenseInfo Class
- Programming Interface

IsTrialVersion Property

Gets a value that indicates whether the application is in trial mode. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLicense.IsTrialVersion
```

Type

```
Boolean
```

Remarks

In the trial version an evaluation mode watermark will be put onto each image and all 'e' are replaced by 'x' and all '5' by '0'.

See also

- LicenseInfo Class
- Programming Interface

LicenseKey Property

Gets the license key used to to activate **Labelstar Office**. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLicense.LicenseKey
```

Type

```
String
```

See also

- LicenseInfo Class
- Programming Interface

LicenseType Property

Gets the license type. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objLicense.LicenseType
```

Type

```
String
```

Remarks

Possible license types are **TRIAL**, **LITE**, **BASIC** or **PROFESSIONAL**.

For more information, see Program Variants.

See also

- LicenseInfo Class
- Programming Interface

UpgradeCode Property

Gets the upgrade code used to activate **Labelstar Office**. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 5.00.1010

Usaged

```
objLicense.UpgradeCode
```

Type

```
String
```

See also

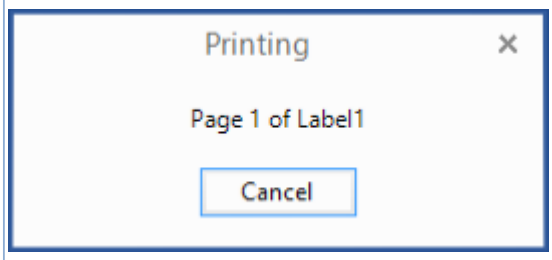
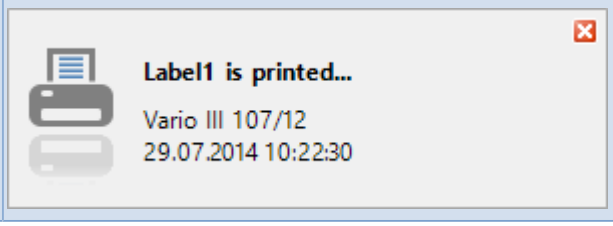
- LicenseInfo Class
- Programming Interface

PrintOptions Enumeration

Specifies the print options.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Members

	Name	Value	Description
	Default	0 (0x00)	Default settings
	PrintCurrentRecord	1 (0x01)	Prints the current selected record. **Select Record** dialog box will no be shown.
	PrintAllRecords	2 (0x02)	Prints all records. **Select Record** dialog box will not be shown.
	ShowPrintingDialog	4 (0x04)	Shows a **Printing** dialog box as long as labels are sent to the printer. If you decide to cancel the printing process - and you're quick enough - you can click the **Cancel** button in this **Printing** dialog box to stop the operation. 
	ShowNotificationMessage	8 (0x08)	Shows a notification at the bottom-right of your screen when a label is printed. 

Remarks

This enumeration allows a bitwise combination of its member values.

The Label.Print method use this enumeration.

For more information and a detailed example, see Label.SelectRecord method.

See also

- Label Class
- Programming Interface

VersionInfo Class

A **VersionInfo** object represents the version information about the API on top of which the application runs. You can get a reference to a **VersionInfo** object through the Application.Info property. The version information is useful to ensure the application is using the proper version of the API.

Properties

	Name	Description
	CompanyName	Gets the company name associated with the API.
	CompiledVersion	Internal version number of the API (this field is for internal use only - see DisplayVersion property for the version string that is displayed to the users).
	Copyright	Gets the copyright notice associated with the API.
	DisplayVersion	Version to be displayed to the users.
	ProductName	Gets the product name associated with the API.

See also

- Programming Interface

VersionInfo Properties

The VersionInfo type exposes the following members.

Properties

	Name	Description
	CompanyName	Gets the company name associated with the API.
	CompiledVersion	Internal version number of the API (this field is for internal use only - see DisplayVersion property for the version string that is displayed to the users).
	Copyright	Gets the copyright notice associated with the API.
	DisplayVersion	Version to be displayed to the users.
	ProductName	Gets the product name associated with the API.

See also

- VersionInfo Class
- Object Reference

CompanyName Property

Gets the company name associated with the API. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objVersion.CompanyName
```

Type

```
String
```

Example

CompanyName: "Carl Valentin GmbH"

See also

- VersionInfo Class
- Object Reference

CompiledVersion Property

Internal version number of the API (this field is for internal use only - see DisplayVersion property for the version string that is displayed to the users). Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objVersion.CompiledVersion
```

Type

```
String
```

Example

CompiledVersion: "4.10.1010"

See also

- VersionInfo Class
- Object Reference

Copyright Property

Gets the copyright notice associated with the API. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objVersion.Copyright
```

Type

```
String
```

Example

Copyright: "Copyright © Carl Valentin GmbH, 2012-2014"

See also

- VersionInfo Class
- Object Reference

DisplayVersion Property

Version to be displayed to the users. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objVersion.DisplayVersion
```

Type

```
String
```

Example

DisplayVersion: "Version 4.10 Build 1010"

See also

- VersionInfo Class
- Object Reference

ProductName Property

Name of the product. Read-only property.

Namespace: LSOffice
Assembly: LSOffice.dll
Version: 4.10.1010

Usage

```
objVersion.ProductName
```

Type

```
String
```

Example

ProductName: "Labelstar Office"

See also

- VersionInfo Class
- Object Reference

Error Codes and Messages

Labelstar Office follows the standard OLE Automation approach for generating errors.

Runtime Error Handling

When handling errors, determine if the error was caused by using OLE Automation incorrectly, or if the error was returned from an OLE Automation feature API call. When the error was returned from an OLE Automation feature API call use Application.LastError property to get further error information. Each call to an OLE Automation feature API (expect LastError) resets the error information, so that Application.LastError property obtains error information only for the most recent OLE Automation feature API call.

Runtime Error Codes

Error Code	Description	Type
1000	Generic error.	Error
1001	The application is alread initialized.	Warning
1002	Invalid license key.	Warning
1003	Invalid license type; Professional license expected.	Warning
1004	Application not initialized; call Initialize() first.	Error
1005	No label opened.	Error
1006	Invalid field name.	Error
1007	No field defined.	Error
1008	Field index out of range.	Error
1009	Invalid property name.	Error
1010	No database fields defined on the label.	Error
1011	No record found.	Error
1012	Multiple database connections.	Error

Example (VBScript)

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Open And Print Label Sample Code
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Object variables
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim objApp
Dim objLabel

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Open and print label
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Set objApp = CreateObject("LSOffice.Application")
```

```
End If

' Browse file name
Dim fileName
fileName = objApp.GetOpenFilename("Labels|*.lbex|All Files|*.*")

If (Len(fileName) = 0) Then
     WScript.Quit
End If

' Open label
Set objLabel = objApp.OpenLabel(fileName)

If (objLabel is Nothing) Then
     WScript.Echo objApp.LastError.Message
     WScript.Quit
End If

' Print label
objLabel.Print(1)
```


Redistributing

Redistributing programs that have been created using the automation interface of **Labelstar Office**, is super easy.

Before Starting

When you redistribute an application, you need to ensure that the user has the correct versions of all the necessary files. The DLLs should be located in the same folder as the executable file.

All files which are needed for redistribution can be found in the following directory: *%InstallDir%\Redist*.

Since native code is used in some DLLs, **Microsoft Visual C++ 2010 SP1 Redistributable (x86)** must be installed on the computer. In addition, **.Net Framework 4.6** or later must be available on the target system.

Required Files

CommonLib.dll	Standard library with basic classes and controls.
LabelDocument.dll	This library contains classes for label definition and management.
LSOffice.dll	The **Labelstar Office** automation interface.
SettingsLib.dll	This library contains classed which can be used to manage the program settings (incl. licensing).
SHWindows.dll	This library contains methods for generating printer fonts (C++).
ActiproSoftware.Shared.WinForms.dll ActiproSoftware.SyntaxEditor.WinForms.dll	
DevComponents.DotNetBar.SuperGrid.dll DevComponents.DotNetBar2.dll	
GdPicture.NET.12.dll GdPicture.NET.12.filters.dll GdPicture.NET.12.image.gdimgplug.dll	
TBarCode11.dll TECIT.TBarCode.dll	

Program Variants

Labelstar Office is available in three versions. In the LITE version, the software is primarily intended for designing simple labels. For professional requirements, there is the BASIC or PROFESSIONAL version. This makes a broad selection of formats and variables available so that the requirements of almost all industries can be fulfilled.

	LITE	BASIC	PROFESSIONAL
Texts			
TrueType fonts	●	●	●
Printer fonts		●	●
Text formatting (Markup Tags)		●	●
Curved text			●
Bar codes			
1D bar codes	●	●	●
2D bar codes		●	●
GS1 bar codes		●	●
Images	Limited (only BMP)	More than 90 graphic and vector formats (e.g. TIFF, GIF, JPEG, PNG, WMF, BMP, ICO)	●
Variables			
System variables	Limited (only date, time, counter, and user input)	More than 30 variables (e.g. date, time, counter, user input, field link, check digit, If..Then.Else statement)	Complex variable definitions (e.g. custom check digit calculation)
Printer variables		●	●
Databases		●	●
Logging			●
Memory Card Support		●	●
Symbols		●	●
Printing			
Internal printer protocoll (CVPL) (Carl Valentin Printer Drivers Version 2.4.1 or later)		●	●
Print preview		●	●
Printing preferences		●	●
Two-color printing		●	●
Print Only			
Quick Print			●
Print Manager			●
Print Form			●
Folder Monitoring			●
SAPscript ITF Export			●
Automation Interface			●
Import of Labelstar PLUS labels		●	●

Installation

Installing Labelstar Office from CD

1. Insert the program CD. The AutoStart function launches the installer.
2. If the auto-start function is deactivated on your computer, launch the program by double-clicking on the **Start.exe** file on the CD.
3. Click **Install Software** to start the installation.

Downloading and installing Labelstar Office

1. Call the download page in order to download **Labelstar Office**.
2. Start the installation by double-clicking on the file you have just downloaded.

Licensing

The following information should help you to activate your program. If you have problems, please contact **Labelstar Office** Support.

How do I activate my program?

You can activate your program by using the **License Wizard.** Thereto you need a license key which you can find on the license label in your program CD.

How can I notice if my software was already activated?

1. Open **Labelstar Office**.
2. Click **Help** on the **File** tab.
3. See **About Labelstar Office** to find the product information.

Note: To receive further information to licensing click on **Additional Version and Copyright Information**. The **About** dialog box opens. Click on the **Program Information** key and select **Licensing**.

What is a trial version?

A trial version allows you to test the program. In the trial version some functions are limited, all e are replaced by x and all 0 by 5. All images are marked with a watermark. In the trial version certain functions or programs are possibly activated which are not included in the scope of delivery of the product bought by you. After you have entered a valid license key only the programs and features bought by you are shown.

How can I enter, delete or change the license key?

1. Click **Start** > **Programs** > **Labelstar Office** > **License Labelstar Office**
   The **License Wizard** opens.
2. **Enter license key** To convert a trial to a regular program version, enter a valid license key.
3. **Delete license key** You can delete a license key to use it e.g. on another computer. After deleting the license key the program runs as trial version.
4. **Change license key** You can enter another license key to release additional features and programs.
5. **Enter upgrade code** To upgrade from BASIC to PROFESSIONAL version enter a valid upgrade code.

Software Update

To perform a Labelstar Office update, proceed as follows:

1. Open **Labelstar Office**.
2. Click **Help** on the **File** tab, and then click **Check for Updates**.
   The **Update Wizard** opens.
3. Follow the instructions in the wizard.

or visit our Updates website to download the latest program version.

Contacts

Product Website

You may find additional information and the latest program version on or website: www.carl-valentin.de

Email

Technical support: support@carl-valentin.de
Ordering and licensing requests: order@carl-valentin.de
General requests: info@valentin-carl.de

System Requirements

Minimal system requirements

- Microsoft Windows 7/8/8.1/10 x86/x64
- .Net Framework 4.6 or later (download from http://www.microsoft.com/net/)
- Microsoft Visual C++ 2010 Redistributable (x86) (download from https://www.microsoft.com/en-us/download/details.aspx?id=8328)
- Microsoft Access Database Engine 2010 (x86)
- Recommended printer drivers: Carl Valentin Printer Drivers Version 2.4.1 or later

Hinweis

Some components, such as .Net Framework for example, are not included in the installation program by default. During installation, the program searches for components, downloads them from the internet where applicable and installs them. If you do not have internet access, you can find the necessary components on the program CD in the *Utilities* folder.

Imprint

Carl Valentin GmbH
Neckarstrasse 78-86 u. 94
78056 Villingen-Schwenningen

Phone: +49 (0) 7720 9712 - 0
Email: info@carl-valentin.de

Copyright © 2016 Carl Valentin GmbH

Disclaimer

These descriptions however do not constitute any guaranteed characteristics in a legal sense or in the sense of any product liability. The authors reserve the right to make changes to the software, to announce any person without obligation these changes. No warranty is accepted for the correctness of the contents of this document. As errors can never be completely avoided, despite all efforts made, we are grateful for any information regarding errors.

Trademark Notifications

All product names mentioned in this document may be trademarks or registered trademarks of their respective owners.