



LABELSTAR OFFICE

User Manual

Version 4.30 Build 1010
May 15, 2015

Contents

Labelstar Office.....	7
Variables.....	8
System Variables	9
Date/Time Variables.....	10
Current Date/Time	11
Current Date	12
Current Time	13
Date/Time (System).....	14
Parse Date/Time	16
Calendar Week	17
Day of Year.....	19
Day of Week.....	21
Field Variables	23
Database Field.....	24
Get Field Content	25
Get Field Name	26
Path Variables.....	27
\$AppDataDir.....	28
\$AppDir	29
\$AppPath	30
\$Dir.....	31
\$Ext	32
\$FileName.....	33
\$ImageDir.....	34
\$InstallDir	35
\$LabelDir.....	36
\$LabelPath	37
String Variables.....	38
Get Leftmost Characters of a String	39
Get Rightmost Characters of a String.....	40
Get Middle Characters of a String	41
Remove Characters.....	42
Replace String	43
Replace Pattern.....	44
Regular Expression Language	46
Pad String from Left.....	51
Pad String from Right.....	52
Reverse String	53
Convert String to Lowercase.....	54
Convert String to Uppercase.....	55
Truncate String	56
Trim Leading Characters	57
Trim Trailing Characters	58
Trim Leading and Trailing Characters.....	59
Convert ASCII String to HEX String.....	60
Convert HEX String to ASCII String.....	61
Calculate Text Length.....	62
Counter (System).....	63

Global Counter.....	65
User Input (System)	66
Input Mask	67
Math Variables	68
Absolute Value	69
Minimum Value	70
Maximum Value	71
Calculate Mathematical Formula	72
Mathematical Operators	73
Check Digit Calculation	74
Check Digit (System).....	75
Append Check Digit.....	76
Misc Variables.....	77
Number of Copies	78
If..Then..Else Statement	79
Shift Definition.....	80
Define Shift Times	81
Label Number	82
Page Number.....	83
Printer Name.....	84
User Name.....	85
User Domain Name.....	86
Format Value	87
Formatting Types	89
Standard Numeric Format Strings.....	90
Custom Numeric Format Strings.....	92
Standard Date and Time Format Strings	93
Custom Date and Time Format Strings	95
Text Format Strings	97
Country Codes.....	99
Format Text.....	100
Printer Variables	101
Date/Time (Printer)	102
Printer-specific Date and Time Format Strings	103
Field Link (Printer).....	105
User Input (Printer).....	106
Counter (Printer).....	108
Extended Counter (Printer).....	110
Check Digit (Printer).....	112
Bar Codes.....	113
1D Bar Codes.....	117
Codabar	118
Code 128.....	119
Code 128 (Subset A).....	120
Code 128 (Subset B).....	121
Code 2 of 5 Industrial.....	122
Code 2 of 5 Interleaved	123
Code 39.....	124
Code 39 (Full ASCII).....	125

Code 93	126
Code 93 (Full ASCII)	127
Deutsche Post Identcode	128
Deutsche Post Leitcode	129
EAN-13, GTIN-13	130
EAN-13 + 2 Digits	131
EAN-13 + 5 Digits	132
EAN-8, GTIN-8	133
ITF-14, SCC-14	134
Pharmacode	135
PZN	136
UPC-A, GTIN-12	137
UPC-E	138
2D Bar Codes	139
Aztec Code	140
Aztec Runes	141
Codablock F	142
DataMatrix	143
MaxiCode	144
Structured Carrier Message	145
PDF417	146
QR Code	147
What are the different types of QR Codes?	148
GS1 Bar Codes	149
GS1 DataBar	150
GS1 DataMatrix	151
GS1-128	152
Check Digit Calculation	153
Modulo 10	154
Modulo 10 (Luhn Algorithm)	155
Modulo 11	156
Global Trade Item Number (GTIN)	157
Databases	158
New Data Connection	159
Create a Database Label	160
Logging	161
Activate and Deactivate Logging	162
Log File Location	163
Markup Tags	164
Food Allergen Labelling	166
Sample	167
Supported Graphic and Vector Formats	169
Program Options	171
«General» Tab	172
«Printing» Tab	173
«Label Preview» Tab	174
«Memory Card» Tab	175
«Logging» Tab	176
«File Locations» Tab	177

Print Only	178
Tools	179
Program Settings	180
Language Settings	181
OLE Automation	182
Operating Requirements	184
Register Assembly for COM Interop	185
Your First Application	186
VBScript Samples	189
Object Reference	190
Application Class	191
Application Properties	192
ActivePrinter Property	193
HasError Property	194
Info Property	195
IsInitialized Property	196
LabelDir Property	197
LastError Property	198
License Property	200
Application Methods	201
Initialize Method	202
GetOpenFilename Method	203
OpenLabel Method	205
Error Class	206
Error Properties	207
Details Property	208
ErrorCode Property	209
ErrorType Property	210
Message Property	211
ErrorType Enumeration	212
Field Class	213
Field Properties	214
FieldName Property	215
Locked Property	216
Printable Property	217
Field Methods	219
GetContent Method	220
GetPropertyvalue Method	222
SetContent Method	225
SetPropertyValue Method	226
ImageFormat Enumeration	228
Label Class	229
Label Properties	230
ActivePrinter Property	231
CurrentRecord Property	234
FieldCount Property	235
FieldNames Property	236
IsDataAvailable Property	238
LabelPath Property	239

MaxRecord Property	240
Modified Property	241
PageName Property	242
Label Methods	243
GetFieldByIndex Method	244
GetFieldByName Method	245
GetPreview Method	246
GetPropertyValue Method	247
Print Method	248
PrintToFile Method	249
Save Method	250
SaveAs Method	251
SavePreview Method	252
SelectRecord Method	253
Filter Expression Syntax	255
Filter Expression Functions	259
SetPropertyValue Method	261
LicenseInfo Class	262
LicenseInfo Properties	263
IsTrialVersion Property	264
LicenseKey Property	265
LicenseType Property	266
PrintOptions Enumeration	267
VersionInfo Class	268
VersionInfo Properties	269
CompanyName Property	270
CompiledVersion Property	271
Copyright Property	272
DisplayVersion Property	273
ProductName Property	274
Error Codes and Messages	275
Program Variants	277
Licensing	278
Software Update	279
Contacts	280
System Requirements	281
Imprint	282

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, e.g. 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. Note: If the character is to be used directly, "\$\$" must be entered.
(Indicates the start of parameter list.
)	Indicates the end of parameter list.
"	Text identification
,	Parameter separator
=	Parameter value separator
\	Escape sign

See also

- [System Variables](#)
- [Printer 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](#)
- › [Path Variables](#)
- › [String Variables](#)
- › [Counter \(System\)](#)
- › [User Input \(System\)](#)
- › [Math Variables](#)
- › [Check Digit Calculation](#)
- › [Misc Variables](#)

Date/Time Variables

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

Supported Date/Time Variables

- › [Current Date/Time](#)
- › [Current Date](#)
- › [Current Time](#)
- › [Date/Time \(System\)](#)
- › [Parse Date/Time](#)
- › [Calendar Week](#)
- › [Day of Year](#)
- › [Day of Week](#)

Current Date/Time

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

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

Syntax

\$CurrentDateTime

Return value

Current date and current time according to the system settings.

Examples

```
$CurrentDateTime -> "15.10.2014 11:03:59"
```

```
\$Format ($CurrentDateTime, "yyMMdd") -> "141015"
```

```
\$Format ($CurrentDateTime, "hhmmss") -> "110359"
```

See also

- > [Current Date](#)
- > [Current Time](#)
- > [Date/Time \(System\)](#)
- > [Date/Time \(Printer\)](#)

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 -> "15.10.2014"

[\\$Format](#) (\$CurrentDate, "yyMMdd") -> "141015"

See also

- > [Current Date/Time](#)
- > [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"

`\$Format ($CurrentTime, "hhmmss")` -> "110359"

See also

- > [Current Date/Time](#)
- > [Current Date](#)
- > [Date/Time \(System\)](#)
- > [Date/Time \(Printer\)](#)

Date/Time (System)

Defines a date and time variable and converts the value with the indicated format in the corresponding 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 String](#)), 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 each change of data record

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="15.06.2009", MonthOffset=2) -> "15.08.2009"
```

```
$DateTime ("D", UpdateInterval=0, DayOffset=2, Language="fr-Fr", StartDate=$ParseDateTime ("131012", "yyMMdd")) -> "samedi 12 octobre 2013"
```

```
$DateTime ("HH:mm:ss") -> "13:20:35"
```

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

```
ID01 = "260514"
```

```
$DateTime ("D", UpdateInterval=0, DayOffset=2, StartDate=$ParseDateTime (<<ID01>>, "ddMMyy")) ->  
"Montag, 26. Juni 2014"
```

See also

- [Current Date/Time](#)
- [Current Date](#)
- [Current Time](#)
- [Date/Time \(Printer\)](#)

Parse Date/Time

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Converts the specified string representation 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 occurred.

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 formatting codes, 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") -> "10.09.2013 00:00:00"
```

```
$ParseDateTime (<<ID01>>, "MMyydd") -> "10.09.2014 00:00:00"
```

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, from 1 to 53
ww	Calendar week, from 01 to 53
www	Calendar week, from 001 to 053
wwww	Calendar week, from 0001 to 0053
\	Escape character
Any other character	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 each change of data record!

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 is used for formatting the output. For more information, see [Country Codes](#).

Return value

Formattes calendar week.

Examples

Current date: 01.02.2014

`$WeekOfYear ("w") -> "5"`

\$WeekOfYear ("ww") -> "05"

\$WeekOfYear ("www", DayOffset=5) -> "006"

\$WeekOfYear ("Calendar \week: ww", StartDate="01.03.2014") -> "Calendar week: 09"

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, Language=language])`

Parameters

format

Indicates how the day of year is to be formatted.

Format identifier	Description
d	Day of year, from 1 to 366
dd	Day of year, from 01 to 366
ddd	Day of year, from 001 to 366
dddd	Day of year, from 0001 to 0366
\	Escape character
Any other character	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: After each change of data record

monthOffset (optional, standard = 0)

Month offset (is added to the current data)

dayOffset (optional, standard = 0)

Day offset (is added to the current date)

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

Defines the start date.

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 day of year.

Examples

Current date: 01.02.2014

`$DayOfYear ("d") -> "5"`

\$DayOfYear ("dd") -> "05"

\$DayOfYear ("ddd", DayOffset=5) -> "006"

\$DayOfYear ("Day of year: dd", StartDate="01.03.2014") -> "Day of year: 09"

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, Language=language])`

Parameters

format

Indicates how the weekday is to be formatted.

Format identifier	Description
d	Day of week, from 0 to 6
dd	Day of week, from 00 to 06
ddd	Day of week, from 000 to 006
dddd	Day of week, from 0000 to 0006
\	Escape character
Any other character	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 each change of data record

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 and time according to the system settings is used)

Defines the start date.

sunday (optional, standard = 0)

Defines which value is to be used for Sunday.

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 day of week.

Examples

Current date: 01.02.2014 (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="05.03.2014", Sunday=10) -> "Day of week: 14"

Field Variables

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

Supported Field Variables

- [Database Field](#)
- [Field Link \(System\)](#)
- [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

Note: Upper and lower case is considered

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	Copies
7	Germany	Berlin	357114	82220000	Deutschland		3

```
$DbField ("Europe", "Area") -> "357114"
```

```
$DbField ("Europe", "Area", Format="0000000000") -> "0000357411"
```

```
$DbField ("Europe", "Capital", Format="LLLL") -> "Berl"
```

```
\$ToUpper ($DbField ("Europe", "Capital")) -> "BERLIN"
```

Check whether a database field is empty

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

See also

› [Databases](#)

Get Field Content

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Defines a field link.

Syntax

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

or

```
<<fieldName>>
```

Parameters

fieldName

Field name

Note: Upper and lower case is considered

displayText (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"
```

```
<<ID02>> -> "abcABC"
```

See also

➤ [Field Link \(Printer\)](#)

Get Field Name

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Returns the field name.

Syntax

`$FieldName`

Return value

Field name

Path Variables

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

Supported Path Variables

- › [Application Data Folder](#)
- › [Application Folder](#)
- › [Application Path](#)
- › [Folder Name](#)
- › [File Extension](#)
- › [File Name](#)
- › [Image Folder](#)
- › [Label Folder](#)
- › [Label Path](#)
- › [Installation Folder](#)

\$AppDataDir

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

\$AppDir

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](#)

\$AppPath

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](#)

\$Dir

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\Label1.lbex") -> "C:\Labels"
```

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

See also

- > [Get file name](#)
- > [Get file extension](#)

\$Ext

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

- [Get file name](#)
- [Get folder name](#)

\$FileName

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 or 1: Returns file name with extension

false or 0: Returns file name without extension

Return value

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

Examples

```
$FileName ("C:\Labels\Label1.lbex") -> "Label.lbex"  
$FileName ($AppPath) -> "LabelDesigner.exe"  
$FileName ($AppPath, Ext=false) -> "LabelDesigner"  
$FileName ($LabelPath) -> "Label.lbex"  
$FileName ($LabelPath, Ext=0) -> "Label"
```

See also

- [Get folder name](#)
- [Get file extension](#)

\$ImageDir

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](#)
- [Get current label folder](#)

\$InstallDir

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](#)

\$LabelDir

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](#)
- [Get current image folder](#)

\$LabelPath

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

See also

- [Get current image folder](#)
- [Get current label folder](#)

String Variables

With the help of this variables strings can be edited.

Supported String Variables

- › [Get Leftmost Characters of a String](#)
- › [Get Rightmost Characters of a String](#)
- › [Get Middle Characters of a String](#)
- › [Remove Characters](#)
- › [Replace String](#)
- › [Replace Pattern](#)
- › [Pad String from Left](#)
- › [Pad String from Right](#)
- › [Reverse String](#)
- › [Convert String to Lowercase](#)
- › [Convert String to Uppercase](#)
- › [Truncate String](#)
- › [Trim Leading Characters](#)
- › [Trim Trailing Characters](#)
- › [Trim Leading and Trailing Characters](#)
- › [Convert ASCII String to HEX String](#)
- › [Convert HEX String to ASCII String](#)
- › [Calculate String Length](#)
- › [Format String](#)

Get Leftmost Characters of a 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 a String](#)
- [Get Middle Characters of a String](#)

Get Rightmost Characters of a 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 a String](#)
- › [Get Middle Characters of a String](#)

Get Middle Characters of a 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, index, [Length=length])
```

Parameters

text

String expression from which characters are returned.

index

Starting position of the characters to return. If *index* is greater than the number of characters in *text*, the function returns a zero-length string ("").

Note: The starting position is one 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 a String](#)
- [Get Rightmost Characters of a String](#)

Remove 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, index, [Length=length])
```

Parameters

text

The string to be changed.

index

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 (string, pattern/filename, replacement, [IgnoreCase=ignorecase, RightToLeft=rightToLeft])
```

Parameters

string

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](#).

Sample: List of keywords you search (separated by |) or the file name of a text file that contains the keywords you search.

replacement

The replacement string.

ignorecase (optional, default = false)

true or 1: Specifies case-insensitive matching.
false or 0: Specifies case-sensitive matching.

righttoleft (optional, default = false)

Change the search direction.
true or 1: Specifies that the search will be from right to left.
false or 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
<code>\a</code>	Matches a bell character, \u0007.	<code>\a</code>	"Error!" + '\u0007'	"\u0007"
<code>\b</code>	In a character class, matches a backspace, \u0008.	<code>[\b]{3,}</code>	"\b\b\b\b"	"\b\b\b\b"
<code>\t</code>	Matches a tab, \u0009.	<code>(\w+)\t</code>	"item1\titem2\t"	"item1\t", "item2\t"
<code>\r</code>	Matches a carriage return, \u000D. (<code>\r</code> is not equivalent to the newline character, <code>\n</code> .)	<code>\r\n(\w+)</code>	"\r\nThese are \ntwo lines."	"\r\nThese"
<code>\v</code>	Matches a vertical tab, \u000B.	<code>[\v]{2,}</code>	"\v\v\v"	"\v\v\v"
<code>\f</code>	Matches a form feed, \u000C.	<code>[\f]{2,}</code>	"\f\f\f"	"\f\f\f"
<code>\n</code>	Matches a new line, \u000A.	<code>\r\n(\w+)</code>	"\r\nThese are \ntwo lines."	"\r\nThese"
<code>\e</code>	Matches an escape, \u001B.	<code>\e</code>	"\x001B"	"\x001B"
<code>\ nnn</code>	Uses octal representation to specify a character (<i>nnn</i> consists of two or three digits).	<code>\w\040\w</code>	"a bc d"	"a b", "c d"
<code>\x nn</code>	Uses hexadecimal representation to specify a character (<i>nn</i> consists of exactly two digits).	<code>\w\x20\w</code>	"a bc d"	"a b", "c d"
<code>\c X</code> <code>\c x</code>	Matches the ASCII control character that is specified by <i>X</i> or <i>x</i> , where <i>X</i> or <i>x</i> is the letter of the control character.	<code>\cC</code>	"\x0003" (Strg-C)	"\x0003"
<code>\u nnnn</code>	Matches a Unicode character by using hexadecimal representation (exactly four digits, as represented by <i>nnnn</i>).	<code>\w\u0020\w</code>	"a bc d"	"a b", "c d"
<code>\</code>	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, <code>*</code> is the same as <code>\x2A</code> , and <code>\.</code> is the same as <code>\x2E</code> . This allows the regular expression engine to disambiguate language elements (such as <code>*</code> or <code>?</code>) and character literals (represented by <code>*</code> or <code>\?</code>).	<code>\d+[\+-x*]\d+\d</code> <code>+[\+-x*\d+</code>	"(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
<code>[character_group]</code>	Matches any single character in <i>character_group</i> . By default, the match is case-sensitive.	<code>[ae]</code>	"gray" "lane"	"a" "a", "e"

<code>[^ <i>character_group</i>]</code>	Negation: Matches any single character that is not in <i>character_group</i> . By default, characters in <i>character_group</i> are case-sensitive.	<code>[^aei]</code>	"reign"	"r", "g", "n"
<code>[<i>first</i> - <i>last</i>]</code>	Character range: Matches any single character in the range from <i>first</i> to <i>last</i> .	<code>[A-Z]</code>	"AB123"	"A", "B"
<code>.</code>	Wildcard: Matches any single character except <code>\n</code> . To match a literal period character (<code>.</code> or <code>\u002E</code>), you must precede it with the escape character (<code>\.</code>).	<code>a.e</code>	"nave" "water"	"ave" "ate"
<code>\p{ <i>name</i> }</code>	Matches any single character in the Unicode general category or named block specified by <i>name</i> .	<code>\p{IsCyrillic}</code>	"ДЖем"	"Д", "Ж"
<code>\P{ <i>name</i> }</code>	Matches any single character that is not in the Unicode general category or named block specified by <i>name</i> .	<code>\P{IsCyrillic}</code>	"ДЖем"	"e", "m"
<code>\w</code>	Matches any word character.	<code>\w</code>	"ID A1.3"	"I", "D", "A", "1", "3"
<code>\W</code>	Matches any non-word character.	<code>\W</code>	"ID A1.3"	" ", "."
<code>\s</code>	Matches any white-space character.	<code>\w\s</code>	"ID A1.3"	"D "
<code>\S</code>	Matches any non-white-space character.	<code>\s\S</code>	"int _ctr"	" _"
<code>\d</code>	Matches any decimal digit.	<code>\d</code>	"4 = IV"	"4"
<code>\D</code>	Matches any character other than a decimal digit.	<code>\D</code>	"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
<code>^</code>	The match must start at the beginning of the string or line.	<code>^\d{3}</code>	"901-333-"	"901"
<code>\$</code>	The match must occur at the end of the string or before <code>\n</code> at the end of the line or string.	<code>-\d{3}\$</code>	"-901-333"	"-333"
<code>\A</code>	The match must occur at the start of the string.	<code>\A\d{3}</code>	"901-333-"	"901"
<code>\Z</code>	The match must occur at the end of the string or before <code>\n</code> at the end of the string.	<code>-\d{3}\Z</code>	"-901-333"	"-333"
<code>\z</code>	The match must occur at the end of the string.	<code>-\d{3}\z</code>	"-901-333"	"-333"
<code>\G</code>	The match must occur at the point where the previous match ended.	<code>\G(\d\)</code>	"(1)(3)(5)[7](9)"	"(1)", "(3)", "(5)"
<code>\b</code>	The match must occur on a boundary between a <code>\w</code> (alphanumeric) and a <code>\W</code> (nonalphanumeric) character.	<code>\b\w+\s\w+\b</code>	"them theme them them"	"them theme", "them them"
<code>\B</code>	The match must not occur on a <code>\b</code> boundary.	<code>\Bend\w*\b</code>	"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
<code>(subexpression)</code>	Captures the matched subexpression and assigns it a one-based ordinal number.	<code>(\w)\1</code>	"deep"	"ee"
<code>(?< name > subexpression)</code>	Captures the matched subexpression into a named group.	<code>(?<double>\w)\k<double></code>	"deep"	"ee"
<code>(?< name1 - name2 > subexpression)</code>	Defines a balancing group definition.	<code>((?'Open'\([\^\(\)]*\))+('Close'Open'\))[\^\(\)]*\)(?'Open')(?!))\$</code>	"3+2^((1-3)*(3-1))"	"((1-3)*(3-1))"
<code>(?: subexpression)</code>	Defines a noncapturing group.	<code>Write(?:Line)?</code>	"Console.WriteLine"	"WriteLine"
<code>(?imnsx-imnsx: subexpression)</code>	Applies or disables the specified options within subexpression.	<code>A\d{2}(?:\w+)\b</code>	"A12xl A12XL a12xl"	"A12xl", "A12XL"
<code>(?= subexpression)</code>	Zero-width positive lookahead assertion.	<code>\w+(?=\.)</code>	"He is. The dog ran. The sun is out."	"is", "ran", "out"
<code>(?! subexpression)</code>	Zero-width negative lookahead assertion.	<code>\b(?:un)\w+\b</code>	"unsure sure unity used"	"sure", "used"
<code>(?<= subexpression)</code>	Zero-width positive lookbehind assertion.	<code>(?<=19)\d{2}\b</code>	"1851 1999 1950 1905 2003"	"99", "50", "05"
<code>(?! subexpression)</code>	Zero-width negative lookbehind assertion.	<code>(?<!19)\d{2}\b</code>	"1851 1999 1950 1905 2003"	"51", "03"
<code>(?> subexpression)</code>	Nonbacktracking (or "greedy") subexpression.	<code>[13579](?>A+B+)</code>	"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
<code>*</code>	Matches the previous element zero or more times.	<code>\d*\.</code>	"0", "19.9", "219.9"	"0", "19.9", "219.9"
<code>+</code>	Matches the previous element one or more times.	<code>"be+"</code>	"been" "bent"	"bee" "be"
<code>?</code>	Matches the previous element zero or one time.	<code>"rai? n"</code>	"ran", "rain"	"ran", "rain"
<code>{ n }</code>	Matches the previous element exactly <i>n</i> times.	<code>",\d{3}"</code>	"1,043.6" "9,876,543,210"	",043" ",876", ",543", ",210"
<code>{ n , }</code>	Matches the previous element at least <i>n</i> times.	<code>"\d{2,}"</code>	"166", "29", "1930"	"166", "29", "1930"
<code>{ n , m }</code>	Matches the previous element at least <i>n</i> times, but no more than <i>m</i> times.	<code>"\d{3,5}"</code>	"193024"	"19302"
<code>*?</code>	Matches the previous element zero or more times, but as few times as possible.	<code>\d*? \.</code>	"0", "19.9", "219.9"	"0", "19.9", "219.9"
<code>+?</code>	Matches the previous element one or more times, but as few times as possible.	<code>"be+?"</code>	"been" "bent"	"be" "be"
<code>??</code>	Matches the previous element zero or one time, but as few times as possible.	<code>"rai?? n"</code>	"ran", "rain"	"ran", "rain"

{ n }?	Matches the preceding element exactly <i>n</i> times.	" , \d{3}?"	"1,043.6" "9,876,543,210"	",043" ",876", ",543", ",210"
{ n , }?	Matches the previous element at least <i>n</i> times, but as few times as possible.	"\d{2, }?"		"166", "29", "1930"
{ n , m }?	Matches the previous element between <i>n</i> and <i>m</i> 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 <i>yes</i> if the regular expression pattern designated by expression matches; otherwise, matches the optional <i>no</i> part. expression is interpreted as a zero-width assertion.	(?(A)\d{2}\b \b\d{3}\b)	"A10 C103 910"	"A10", "910"
(?(name) yes no)	Matches <i>yes</i> if name, a named or numbered capturing group, has a match; otherwise, matches the optional <i>no</i> .	(?<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 <i>number</i> .	\b(\w+)(\s)(\w+)\b	\$\$2\$1	"one two"	"two one"
\$(name)	Substitutes the substring matched by the named group <i>name</i> .	\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.	<code>\\$? \d*\.\.? \d+</code>	<code>**\$&**</code>	"\$1.30"	"***\$1.30**"
\$`	Substitutes all the text of the input string before the match.	<code>B+</code>	<code>\$`</code>	"AABBCC"	"AAAACC"
\$'	Substitutes all the text of the input string after the match.	<code>B+</code>	<code>\$'</code>	"AABBCC"	"AACCCC"
\$+	Substitutes the last group that was captured.	<code>B+(C+)</code>	<code>\$+</code>	"AABBCCDD"	AACDD
\$_	Substitutes the entire input string.	<code>B+</code>	<code>\$_</code>	"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.	<code>\b(?:i)a(?:-i)a\w+\b</code>	"aardvark AAAuto aaaAuto Adam breakfast"	"aardvark", "aaaAuto"
m	Use multiline mode. <code>^</code> and <code>\$</code> 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.	<code>\b(?:x) \d+ \s \w+</code>	"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.	<code>\bA(?:i)b\w+\b</code>	"ABA Able Act"	"ABA", "Able"
(?#comment)	Inline comment. The comment ends at the first closing parenthesis.	<code>\bA(?:#Matches words starting with A)\w+\b</code>		
# [bis Zeilenende]	X-mode comment. The comment starts at an unescaped # and continues to the end of the line.	<code>(?x)\bA\w+\b#Matches words starting with A</code>		

Pad String from Left

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Increase 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 = spaces)

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](#).

Increase 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 = spaces)

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](#)

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"
```

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](#)

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"
```


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

- [Remove leading and trailing characters](#)
- [Remove 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

- [Remove leading and trailing characters](#)
- [Remove leading characters](#)

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

- [Remove leading characters](#)
- [Remove trailing characters](#)

Convert ASCII String to HEX String

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Converts an ASCII string in 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 HEX String to ASCII String

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Converts a HEX string in 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](#)

Calculate Text Length

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Gets the number of characters in the current string.

Syntax

`$Length (string)`

Parameter

string

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.")`

Counter (System)

Inserts a 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 maximum 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 or 1: Remove leading zeros at output

false or 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 overflow)

2: Reset start value manually

3: Reset start value manually (automatic overflow)

radix (optional, standard = 10)

Radix, basis of counter (1-36)

1: Alphabetical (A-Z)

2: Binäre (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](#)
- > [Counter \(Printer\)](#)

Global Counter

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

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

To define a global counter, please proceed as follows:

1. Select a text or bar code field.
2. Open the **Variable Editor**.
3. Define a new customized variable.

4. Define [Counter \(System\)](#) and insert as start value the new defined variable.

Examples

Start = "0010"

[\\$Counter](#) (\$Start, MinValue="0000", MaxValue="0009", Increment=1, Radix=10) -> 0010, 0011, 0012, 0013, 0014, 0015, 0016, 0017, 0018, 0019, 0020, ...

User Input (System)

Inserts a user input (system) on the label.

Syntax

`$UserInput`

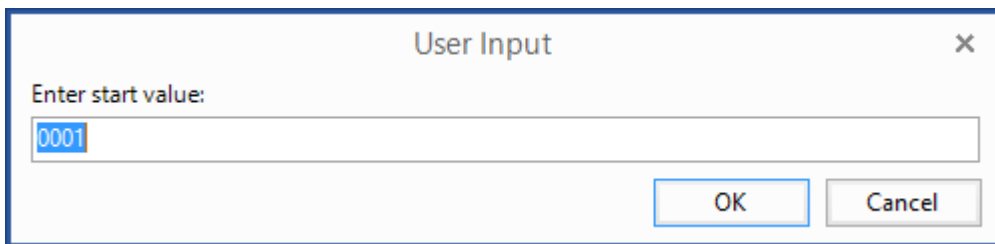
Return value

The entered text.

Examples

Prompt text = "Enter start value:"

Start text = "0001"



See also

➤ [User Input \(Printer\)](#)

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
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 that follows to lowercase.
>	Shift up. Converts all characters that follows to uppercase.
	Disable previous shift up or shift down.
\	Escape. 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.

Math Variables

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

Supported Math Variables

- › [Absolute Value](#)
- › [Minumum Value](#)
- › [Maximum Value](#)
- › [Calculate Mathematical Formula](#)

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"`

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

➤ [\\$Min](#)

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

➤ [\\$Min](#)

Calculate Mathematical Formula

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Defines a mathematic field.

Syntax

`$MathField (formula)`

Parameters

formula

Formular which is to be calculated. For more information, see [Mathematical Operators](#).

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"`

ID	Name	Capital	Area	Population	NativeName	Flag	Copies
7	Germany	Berlin	357114	82220000	Deutschland		3

`$MathField ($DbField ("Europe", "Population") * 2.00) -> "164440000"`

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 oder 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)

Check Digit Calculation

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

Supported Variables

- › [Check Digit \(System\)](#)
- › [Append Check Digit](#)

Check Digit (System)

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Calculates a check digit.

Syntax

```
$CheckDigit (data, checkDigitMethod)
```

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 algorithm)
MOD11	Modulo 11
MOD43	Modulo 43
MOD47_15	Modulo 47 (weighting 15)
MOD47_20	Modulo 47 (weighting 20)
MOD103	Modulo 103

Return value

Calculated check digit.

Examples

```
NVE = "34012345123456789"
```

```
$CheckDigit ("12345", MOD10) -> 7
```

```
$CheckDigit (<<NVE>>, MOD10) -> 5
```

See also

- > [Append Check Digit](#)
- > [Check Digit \(Printer\)](#)

Append Check Digit

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Calculates a check digit.

Syntax

```
$AppendCheckDigit (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 algorithm)
MOD11	Modulo 11
MOD43	Modulo 43
MOD47_15	Modulo 47 (weighting 15)
MOD47_20	Modulo 47 (weighting 20)
MOD103	Modulo 103

appendTo (optional, standard = Right)

Specifies where the calculated check digit is to be appended to the data.
 Left: Indicates that the check digit is inserted at the beginning of the data.
 Right: Indicates that the check digit is appended at the end of the data.

Return value

Data with the check digit appended to the start or the end.

Examples

```
NVE = "34012345123456789"
```

```
$AppendCheckDigit ("12345", MOD10) -> 123457
```

```
$AppendCheckDigit (<<NVE>>, MOD10) -> 340123451234567895
```

```
$AppendCheckDigit (<<NVE>>, MOD10, AppendTo=Left) -> 534012345123456789
```

See also

➤ [Check Digit \(System\)](#)

Misc Variables

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

Supported Variables

- › [Number of Copies](#)
- › [If..Then..Else Statement](#)
- › [Shift Definition](#)
- › [Label Number](#)
- › [Page Number](#)
- › [Printer Name](#)
- › [User Name](#)
- › [User Domain Name](#)
- › [Format Value](#)
- › [Format Text](#)

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

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	Copies
7	Germany	Berlin	357114	82220000	Deutschland		3

```
$If ($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.")
```

Shift Definition

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Displays the current shift definition on the label.

Syntax

`$Shift`

Return value

Shift definition of printer-specific variable definition.

Examples

Early shift -> 06:00 - 13:59

Late shift -> 14:00 - 21:59

Night shift -> 22:00 - 05:59

System variable (TrueType font)

`$Shift` -> "Early shift" (08:20)

`$Shift` -> "Late shift" (15:30)

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 then click **Shift Definitions**.
The **Shift Definitions** dialog box opens.
2. Click on:
 -  to define a new shift.
 -  or  to delete the selected shift.
 -  or double-click the selected shift to change the shift settings.
 -  or  +  to move the selected shift one position upward.
 -  or  +  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 - 14:00
 Late shift -> 14:00 - 22:00
 Night shift -> 22:00 - 06:00

Right

Early shift -> 06:00 - 13:59
 Late shift -> 14:00 - 21:59
 Night shift -> 22:00 - 05:59

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

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

User Name

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Returns the current user name.

Syntax

\$UserName

Return value

User name

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

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 output. For more information, see [Country Codes](#).

type (optional)

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

Number: *value* is interpreted as number

Date/Time: *value* is interpreted as date/time

Text: *value* is interpreted as text

Return value

Formatted text.

Examples

Show number with leading zeros

```
$Format (15, "0000") -> "00015"
$Format (-15, "0000") -> "-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"
```

See also

➤ [Format Text](#)

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. Precision specifier: Number of digits in the result string.	255 ("X") -> FF -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 ("0000") -> 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
Any other character		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 string. 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 ρ (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 ρ (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
yyyy	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
Any other character	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.

Format specifier	Name	Description	Examples
0	Digit	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.	
&	Character	Replaces the placeholder by an appropriate existing character; otherwise a space is indicated in the result string.	abc-123-DEF (&&&&&) -> abc-1
C	Character (optional)	Replaces the placeholder by an appropriate existing character; otherwise no character is indicated in the result string.	
A	Alphanumeric character	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	Alphanumeric character (optional)	Replaces the placeholder by an appropriate existing alphanumeric character (0-9, a-Z); 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
^	Deactivate right-aligned output	Deactivates a preceding right-aligned output.	
\	Escape character	The character which follows the escape character is interpreted as literal and not as customized format identifier.	abcDEF (LLL\LLLL) -> abcLDEF
Any other character	Literale		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 Text

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Formats a text.

Syntax

`$FormatText (value, format)`

Parameters

value

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 ("123456", "####,##") -> "1234,56"
```

```
$FormatText ("1234", "0000-00") -> 0012-34
```

See also

➤ [Format Value](#)

Printer Variables

With the help of these variables printer-internal variables can be defined on the label. In contrast to [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 \(Printer\)](#)
- › [Field Link \(Printer\)](#)
- › [User Input \(Printer\)](#)
- › [Counter \(Printer\)](#)
- › [Extended Counter \(Printer\)](#)
- › [Check Digit \(Printer\)](#)
- › [Shift Definition](#)

Date/Time (Printer)

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
false or 0: Change into next month
true or 1: Retain current month

Return value

Printer-specific variable definition.

Examples

Aktuelle Druckereinstellung: 25.02.2014 14:21:25

```
$PrnDateTime ("DD.MO.YYYY") -> "=CL(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)	15.06.2009 01:45:30 -> 01 15.06.2009 13:45:30 -> 13
HE	Hour, from 00 to 23 (12-hours format)	15.06.2009 01:45:30 -> 01 15.06.2009 13:45:30 -> 01
MI	Minute, from 00 to 59	15.06.2009 01:09:30 -> 09 15.06.2009 13:09:30 -> 09
SS	Second, from 00 to 59	15.06.2009 13:45:09 -> 09
AM, Am or am	AM/PM output	15.06.2009 13:45:09 -> PM (AM) 15.06.2009 13:45:09 -> p.m. (Am) 15.06.2009 13:45:09 -> pm (am)
DD	Day of month, from 01 to 31	01.06.2009 13:45:30 -> 01 15.06.2009 13:45:30 -> 15
MO	Month, from 01 to 12	15.06.2009 13:45:30 -> 06
YYYY	Year (four-digit)	15.06.2009 13:45:30 -> 2009
YY	Year, from 00 to 99	15.06.2009 13:45:30 -> 09
Y	Year, from 0 to 9	15.06.2009 13:45:30 -> 9
WW	Calendar week	15.06.2009 13:45:30 -> 25
DW	Day of week, from 0 (Sunday) to 6 (Saturday)	15.06.2009 13:45:30 -> 1
DW1	Day of week, from 1 (Sunday) to 7 (Saturday)	15.06.2009 13:45:30 -> 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	15.06.2009 13:45:30 -> 166
DY	Day of year, from 000 to 364	15.06.2009 13:45:30 -> 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	15.06.2009 13:45:30 -> JN (CMO) 15.06.2009 13:45:30 -> JUN (DMO) 15.06.2009 13:45:30 -> GUI (IMO)
xSO	Full name of month	15.06.2009 13:45:30 -> June (ESO) 15.06.2009 13:45:30 -> Juin (FSO) 15.06.2009 13:45:30 -> Junio (SSO)
xSD	Shortened name of weekday	15.06.2009 13:45:30 -> MO (GSD) 15.06.2009 13:45:30 -> MA (NSD) 15.06.2009 13:45:30 -> LUN (SSD)
xLD	Full name of weekday	15.06.2009 13:45:30 -> Monday (ELD)

		15.06.2009 13:45:30 -> Montag (GLD)
		15.06.2009 13:45:30 -> 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 (Printer)

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

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

See also

➤ [Field Link \(System\)](#)

User Input (Printer)

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Defines a printer-internal user input.

Syntax

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

Parameters

prompt

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

text

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 *text* length is used as start position.

allowableChars (optional, Standard = 0)

Indicates which characters are allowed for the input.

0: Numeric

1: Alphanumeric

skipSpecialChars (optional, Standard = 0)

Indicates whether special characters are to be retained at the input or not.

0: Do not skip special characters

1: Skip special characters

printAlignment (optional, Standard = 0)

Print alignment

0: Right-aligned

1: Left-aligned

inputAlignment (optional, Standard = 0)

Input alignment

0: Right-aligned

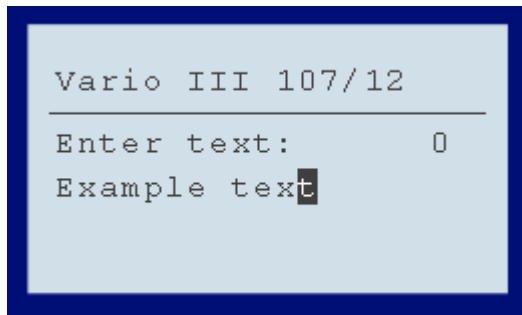
1: Left-aligned

Return value

Printer-specific variable definition.

Examples

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



See also

- [User Input \(System\)](#)

Counter (Printer)

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Defines a printer-internal 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 digits specifies the output format (maximum "999999999").

prompt (optional, Standard = leer)

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 overflow)

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)

Time to which the start value is to be reset.

Format: "HH:MM"

resetValue (optional, for operating mode 6 and 7 only)

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 (Printer)

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

Defines a printer-internal 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 = leer)

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.

minValue (optional, Standard = 0)

Minimum value.

maxValue (optional, Standard = leer)

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 or 1: Enable leading zeros at output

false or 0: Show leading zeros at output

mode (optional, Standard = 5)

Operating mode

0: Reset start value manually

1: Reset start value manually (automatic overflow)

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

See also

- [Counter \(Printer\)](#)
- [Counter \(System\)](#)

Check Digit (Printer)

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	Beschreibung
MOD10	Modulo 10
MOD11	Modulo 11
MOD43	Modulo 43
MOD47_15	Modulo 47 (weighting 15)
MOD47_20	Modulo 47 (weighting 20)
MOD103	Modulo 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 \(System\)](#)




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	 123456	Numeric bar code, encoded digits and special characters.
Codablock F		2D bar code based on the Code 128 .
Code 128	 ABCabc	Alphanumeric bar code, encoded ASCII character set.
Code 128 (Subset A)	 ABCDEF	Alphanumeric bar code, encoded digits, capital letters and special characters.
Code 128 (Subset B)	 ABCabc	Alphanumeric bar code, encoded digits, letters and special characters.
Code 2 of 5 Industrial	 123456	Numeric bar code.

Code 2 of 5 Interleaved	 123456	Numeric bar code, with an even number of digits.
Code 39	 *ABCDEF*	Alphanumeric bar code, encoded digits, capital letters, special characters and spaces.
Code 39 (Full ASCII)	 *ABCabc*	Alphanumeric bar code based on Code 39 , encoded ASCII character set.
Code 93	 ABCDEF	Alphanumeric bar code, encoded digits, capital letters, special characters and spaces.
Code 93 (Full ASCII)	 ABCabc	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	 01.234 567.890 5	Numeric bar code based on Code 2/5 Interleaved with different check digit calculation.
Deutsche Post Leitcode	 01234.567.890.12 0	Numeric bar code based on Code 2/5 Interleaved with different check digit calculation.
EAN-13, GTIN-13	 1 234567 890128	Numeric bar code.
EAN-13 + 2 Stellen	 1 234567 890128 12	EAN 13 with 2-digit add on.

EAN-13 + 5 Stellen	 <p>1 234567 890128 12345</p>	EAN 13 with 5-digit add on.
EAN-8, GTIN-8	 <p>1234 5670</p>	Numeric bar code.
GS1 DataBar	 <p>(01) 00614141999996</p>	Alphanumeric bar code.
GS1 DataMatrix		2D bar code.
GS1-128	 <p>(01) 00614141999996</p>	Alphanumeric bar code.
ITF-14, SCC-14	 <p>00614141999996</p>	Numeric bar code based on Code 2/5 Interleaved .
MaxiCode		2D bar code.
PDF417		2D bar code.
Pharmacode	 <p>123456</p>	Numeric bar code.
PZN	 <p>PZN - 12345684</p>	Numeric bar code based on Code 39 .

QR Code		2D bar code.
UPC-A, GTIN-12	 1 23456 78901 2	Numeric bar code.
UPC-E	 0 123456 5	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
Valid characters	Digits 0-9 Special characters - \$: / . +
Check digit	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
Valid characters	ASCII character set including control characters
Check digit	Modulo 103

See also

- > [Code 128 \(Subset A\)](#)
- > [Code 128 \(Subset B\)](#)
- > [GS1-128](#)

Code 128 (Subset A)



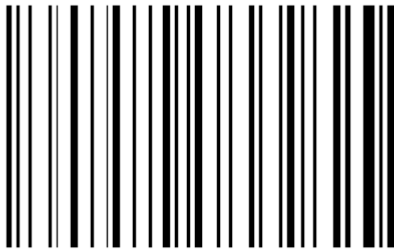
Special type of [Code 128](#).

Length	Variable
Valid characters	Digits 0-9 Upper case letters A-Z Control characters
Check digit	Modulo 103

See also

- › [Code 128](#)
- › [Code 128 \(Subset B\)](#)

Code 128 (Subset B)



ABCabc

Special type of [Code 128](#).

Length	Variable
Valid characters	Digits 0-9 Upper and lower case letters A-z
Check digit	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
Valid characters	Digits 0-9
Check digit	Optional Modulo 10 Modulo 10 (Luhn-Algorithmus)

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 an 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)
Valid characters	Digits 0-9
Check digit	Optional Modulo 10 Modulo 10 (Luhn-Algorithmus)

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 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.

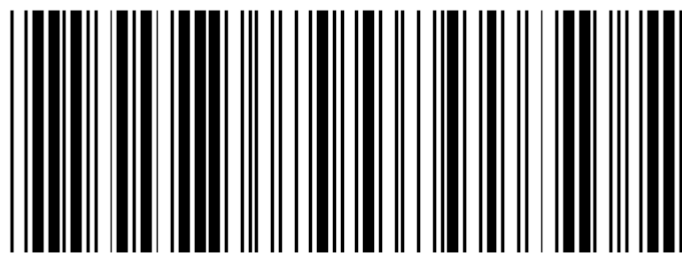
Possibly the most serious drawback of **Code 39** is its low data density: It requires more space to encode data in **Code 39** than, for example, in [Code 128](#).

Length	Variable
Valid characters	Digits 0-9 Upper case letters A-Z Special characters - . \$ / + % Space
Check digit	Optional Modulo 43 Modulo 11 (weighting 7) Modulo 10 (Luhn-Algorithmus)

See also

➤ [Code 39 \(Full ASCII\)](#)

Code 39 (Full ASCII)



ABCabc

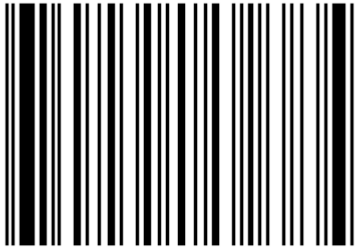
Code 39 (Full ASCII) is an extended version of [Code 39](#) that can encode the complete ASCII character set. The additional characters (e.g. lower case letters) are created using the existing characters of [Code 39](#) by combining two characters each.

Length	Variable
Valid characters	ASCII character set
Check digit	Optional Modulo 43 Modulo 11 (weighting 7) Modulo 10 (Luhn-Algorithmus)

See also

➤ [Code 39](#)

Code 93



ABCDEF

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
Valid characters	Digits 0-9 Upper case letters A-Z Special characters - . \$ / + % Space
Check digit	Modulo 47

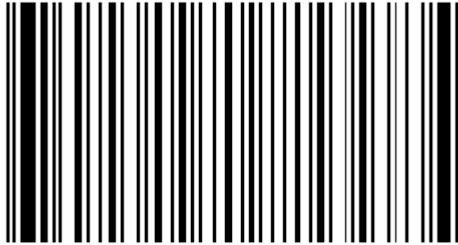
See also

➤ [Code 93 \(Full ASCII\)](#)

Code 93 (Full ASCII)

Note

This barcode is transmitted graphically.



Code 93 (Full ASCII) is an extended version of [Code 93](#) that can encode the complete ASCII character set.

ABCabc

Length	Variable
Valid characters	ASCII character set
Check digit	Modulo 47

See also

➤ [Code 93](#)

Deutsche Post Identcode



The **Identcode** is a variant of [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
Valid characters	Digits 0-9
Check digit	Modulo 10

See also

› [Deutsche Post Leitcode](#)

Deutsche Post Leitcode



The **Leitcode** is a variant of [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
Valid characters	Digits 0-9
Check digit	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
Valid characters	Digits 0-9
Check digit	Modulo 10

See also

- > [EAN-13 + 2 Digits](#)
- > [EAN-13 + 5 Digits](#)
- > [EAN-8, GTIN-8](#)
- > [Global Trade Item Number \(GTIN\)](#)

EAN-13 + 2 Digits



[EAN 13](#) with two additional characters.

Length	15
Valid characters	Digits 0-9
Check digit	Modulo 10

See also

- > [EAN 13, GTIN-13](#)
- > [EAN 13 + 5 Digits](#)

EAN-13 + 5 Digits



[EAN 13](#) with five additional characters.

Length	18
Valid characters	Digits 0-9
Check digit	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.

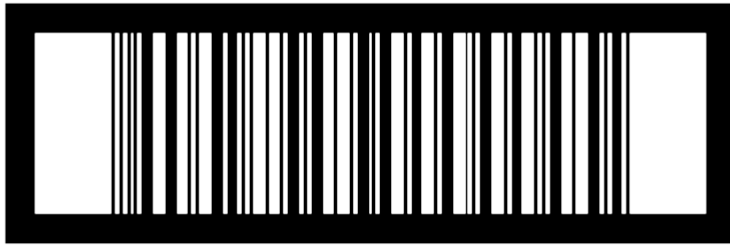
It includes a 2 or 3 digit country code, 4 or 5 data digits (depending on the length of the country code), and a check digit.

Length	8
Valid characters	Digits 0-9
Check digit	Modulo 10

See also

- › [EAN-13, GTIN-13](#)
- › [Global Trade Item Number \(GTIN\)](#)

ITF-14, SCC-14



00614141999996

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/EAN country prefix (GS1 country prefix)
- **4..8:** GS1 Company Prefix (can be longer)
- **9..13:** Item identification number
- **14:** Check digit

Length	14
Valid characters	Digits 0-9
Check digit	Modulo 10

Pharmacode

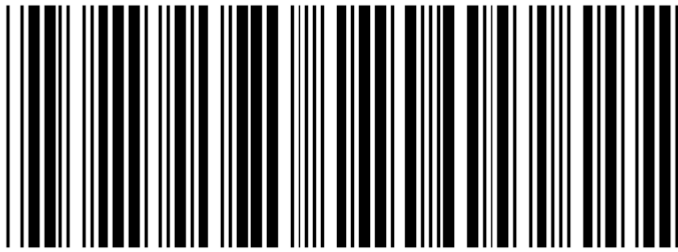


The **Pharmacode** is a simple, numeric bar code placed on the marked from company Laetus. It is used in pharmaceutical industry for the control of packaging means and/or for the control of packaging machines.

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
Valid characters	Digits 0-9
Check digit	None

PZN



PZN - 12345684

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 **PZN8** replaces the old **PZN7** from the 01.01.2013. You will be able to convert old **PZN7** code to **PZN8** by just adding a leading zero.

Length	7-8
Valid characters	Digits 0-9
Check digit	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

Länge	12
Zeichensatz	Ziffern 0-9
Prüfziffer	Modulo 10

Siehe auch

- [UPC-E](#)
- [Global Trade Item Number \(GTIN\)](#)

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
Valid characters	Digits 0-9
Check digit	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](#)
- › [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
Valid characters	ASCII
Check digit	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
Valid characters	An integer between 0 and 255 (including the boundaries).
Check digit	Internal

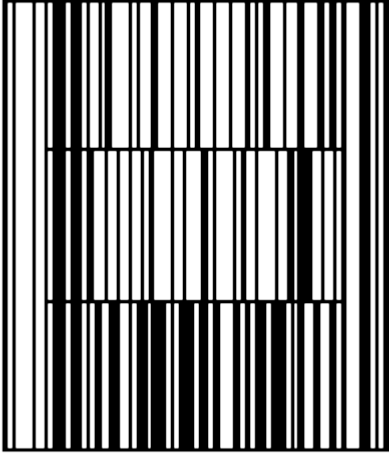
See also

➤ [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](#).

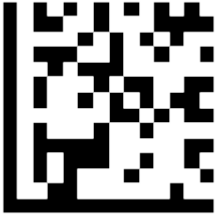
The principle of Codablock bar code is the same like the line break of a text editor - if the line is full it is wrapped to the next line, i.e. to each line the line number is added and to the finished block the number of lines. For the orientation of the reading device, each line contains a line indicator 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.
Valid characters	ASCII
Check digit	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
Valid characters	ASCII
Check digit	Internal

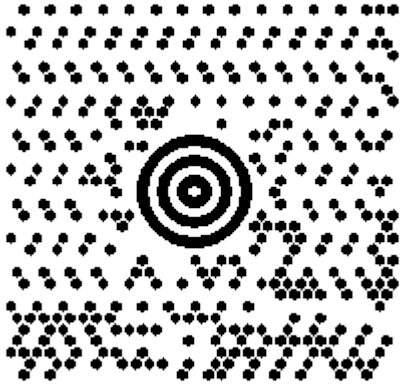
See also

➤ [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 mm²) 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:

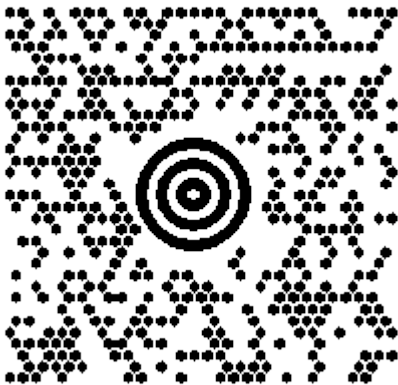
- **Mode 2:** Structured Carrier Message - Numeric Postal Code (up to 9 digits)
- **Mode 3:** Structured Carrier Message - Alphanumeric Postal Code (up to 6 characters)
- **Mode 4:** Raw Data, Standard Error Correction

Length	93 alphanumeric characters 138 numeric characters
Valid characters	ASCII
Check digit	Internal

Structured Carrier Message

Field	Data	Required	Example
Primary Message			
Postal/Zip Code	Mode 2 (US Carrier): 5-digit zip code + 4-digit zip code extension Mode 3 (International Carrier): 6-alphanumeric characters zip code (A through Z or 0 to 9)	yes	000012345
Country Code	numeric, 3 digits	yes	276
Class of Service	numeric, 3 digits	yes	001
Secondary Message			
Tracking Number	alphanumeric, 10 or 11 digits	yes	9A00001234
Standard Carrier Alpha Code	UPSN	yes	UPSN
UPS Account Number	alphanumeric, 6 digits	yes	07X720
Julian Day of Collection	numeric, 3 digits	yes	155
Shipment ID Number	alphanumeric, up to 30 digits	-	
Package n/x	numeric, up to 3 digits/numeric, up to 3 digits	yes	1/1
Package Weight	numeric, up to 3 digits	yes	015
Address Validation	Y or N	yes	Y
Ship To Street Address	alphanumeric, up to 35 digits	-	Muster GmbH
Ship to City	alphanumeric, up to 20 digits	yes	Musterstadt
Ship to State	alpha, 2 digits	yes	DE

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
Valid characters	ASCII
Check digit	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
Valid characters	ASCII
Check digit	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 DataBar](#)
- [GS1 DataMatrix](#)
- [GS1-128](#)

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:

Omnidirectional GS1 DataBar Symbols "PoS compatible"		Small GS1 DataBar Symbols not "PoS compatible"
<p>GS1 DataBar Omnidirectional</p>  <p>(01) 00614141999996</p>	<p>GS1 DataBar Expanded</p>  <p>(01) 00614141999996</p>	<p>GS1 DataBar Truncated</p>  <p>(01) 00614141999996</p>
<p>GS1 DataBar Stacked Omnidirectional</p> 	<p>GS1 DataBar Expanded Stacked</p> 	<p>GS1 DataBar Limited</p>  <p>(01) 00614141999996</p>
		<p>GS1 DataBar Stacked</p> 

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
Check digit	None

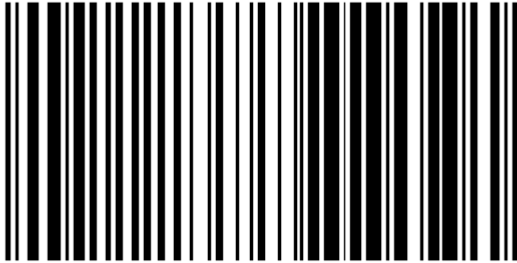
See also

➤ [DataMatrix](#)

GS1-128

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).



(01) 00614141999996

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
Check digit	Modulo 103

See also

➤ [Code 128](#)

Check Digit Calculation

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](#)
- [Modulo 10 \(Luhn Algorithm\)](#)
- [Modulo 11](#)

Modulo 10

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 weighting of 3 from the right.

Example Code



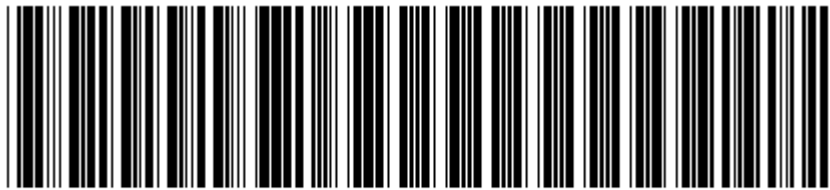
Digit position	13	12	11	10	9	8	7	6	5	4	3	2	1
Raw text	4	0	1	2	3	4	5	9	8	7	6	5	2
Check digit	2												
Digits	4	0	1	2	3	4	5	9	8	7	6	5	
Weights	1	3	1	3	1	3	1	3	1	3	1	3	
Multiply digits by weight	4	0	1	6	3	12	5	27	8	21	6	15	
Add results	$4 + 0 + 1 + 6 + 3 + 12 + 5 + 27 + 8 + 21 + 6 + 15 = 108$												
Find the remainder mod 10	$108 \text{ Mod. } 10 = 8 \text{ (} 108/10 = 10 \text{ Rest } 8\text{)}$												
Subtract from 10	$10 - 8 = 2$												
Check digit	2												

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



455673758689985

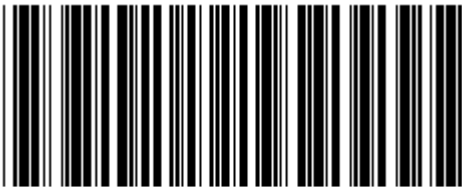
Digit position	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Raw text	4	5	5	6	7	3	7	5	8	6	8	9	9	8	5	5
Check digit	5															
Digits	4	5	5	6	7	3	7	5	8	6	8	9	9	8	5	
Weights	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	
Multiply digits by weight	8	5	10	6	14	3	14	5	16	6	16	9	18	8	10	
Checksum of weighted digits	8	5	1	6	5	3	5	5	7	6	7	9	9	8	1	
Add checksums	$8 + 5 + 1 + 6 + 5 + 3 + 5 + 5 + 7 + 6 + 7 + 6 + 9 + 9 + 8 + 1 = 85$															
Find the remainder mod 10	$85 \text{ Mod. } 10 = 5 \text{ (} 85/10 = 8 \text{ Rest } 5 \text{)}$															
Check digit	5															

Modulo 11

Modulo 11 is e.g. used by [PZN](#).

Calculating the the check digit of PZN: Multiply the first digit by 2, the second digit by 3, ... the sixth 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 - 6319429

Digit position	1	2	3	4	5	6	7
Raw text	6	3	1	9	4	2	9
Check digit	9						
Digits	6	3	1	9	4	2	
Multiply by	2	3	4	5	6	7	
Results	12	9	4	45	24	14	
Add results	$12 + 9 + 4 + 45 + 24 + 14 = 108$						
Find the remainder mod 11	$108 \text{ Mod. } 11 = 9 \text{ (} 108/11 = 9 \text{ Rest } 9\text{)}$						
Check digit	9						

Global Trade Item Number (GTIN)

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

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.

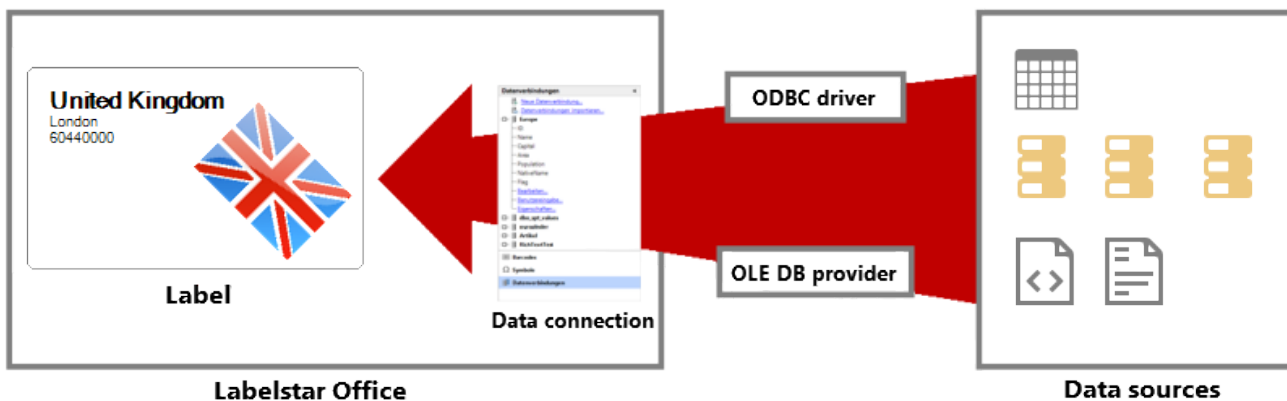
Data on a label can come from two different locations. The data may be stored directly within the label, or it may be stored in an external data source, such as a text file or a database. This external data source is connected to the label through a data connection, which is a set of information that 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 ODBC driver or OLE DB provider 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.



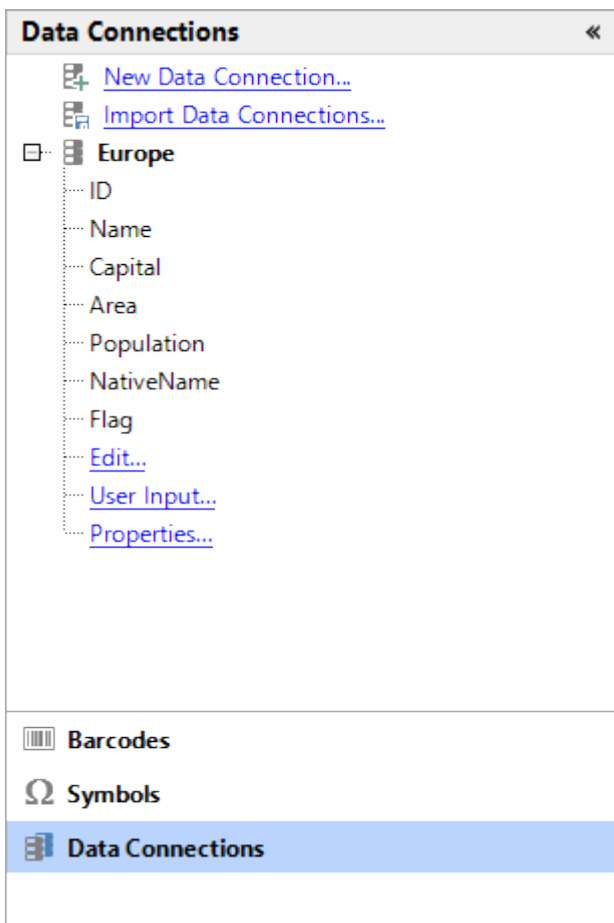
First steps

- [Create a new data connection](#)
- [Create a database label](#)

New Data Connection

To specify a new data connection, proceed as follows:

1. Select **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.



Create a 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)



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**.

Gap length	2,00 mm
Label height	60,00 mm
Label type	Adhesive labels
Label width	100,00 mm
Snap Lines...	
Printing	
Label rotation	180°
Log print job	Marked fields only
Print background image	No
Printer	All fields
Use temporary printer date	Marked fields only
Page Setup...	
Shift Definitions...	
Printing Preferences...	
Settings	
Preview image	<input type="checkbox"/> (None)
Save label preview	<input checked="" type="checkbox"/> Yes
Comment...	

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`).

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.	<code>bold text</code> -> bold text
br	Defines a single line break.	line 1 <code>
</code> line 2 -> line 1 line 2
em	Defines emphasized text (same as italic text).	<code>emphasized text</code> -> <i>emphasized text</i>
font	Defines font, color, and size for text. <i>Attributes:</i> <ul style="list-style-type: none"> • size: Font size • name: Font name • color: Font color 	<code>Example text</code> -> Example text <code>Example text</code> -> Example text
i	Defines italic text.	<code><i>italic text</i></code> -> <i>italic text</i>
rtl	Right-to-left text.	<code><rtl>Example text</rtl></code> -> txeT elpmaxE
shadow	Defines text with a shadow. <i>Attributes:</i> <ul style="list-style-type: none"> • color: Shadow color (Default: Black) • offset: Shadow offset (Default: 1,1) • strength: Shadow size (Default: 1,1) • style: Shadow style (Default: Solid) Solid Blurred 	<code><shadow style='Blurred'>Example text</shadow></code> -> Example text <code><shadow color='Black'>Example text</shadow></code> -> Example text
strike	Defines strikethrough text.	<code><strike>strikethrough text</strike></code> -> strikethrough text
stroke	Defines stroked text. <i>Attributes:</i> <ul style="list-style-type: none"> • width: Border width (Default: 1) • color: Border color (Default: Black) 	<code><stroke color='#FF0000'>Example text</stroke></code> -> Example text
strong	Defines important text.	<code>important text</code> -> important text
sub	Defines subscripted text.	H <code><sub>2</sub></code> O -> H ₂ O
sup	Defines superscripted text.	footer <code><sup>1</sup></code> -> footer ¹
u	Defines underlined text.	<code><u>underlined text</u></code> -> <u>underlined text</u>

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	™

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`.

Allergens.txt This file contains the list of allergens that are being emphasized.

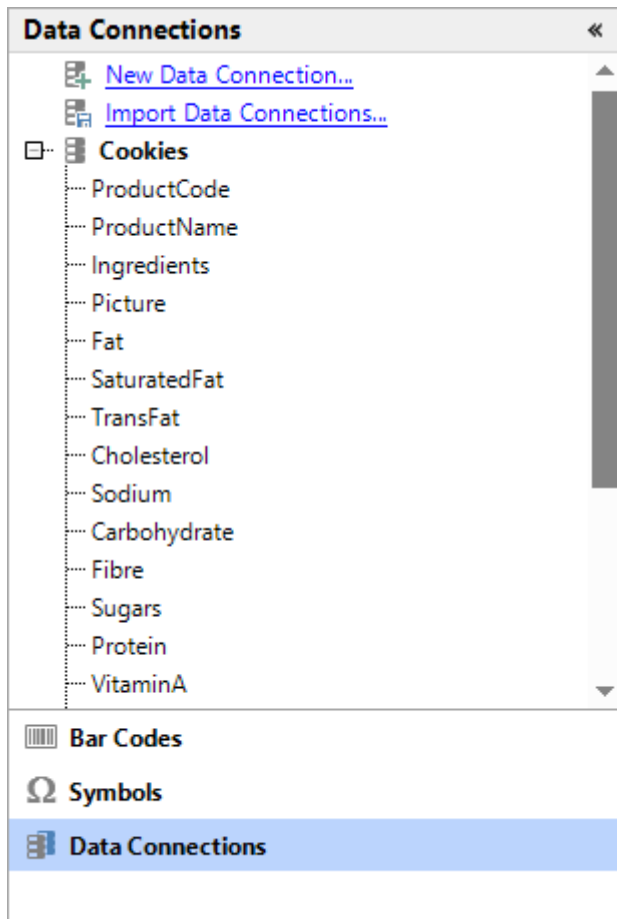
Cookies.accdb Microsoft Access database with the ingredients lists.

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 **Import Data Connections** and browse for the file **Import Cookies data connection.lbdx**.



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

Nutrition Facts

Per 1 cookie (28g)

Amount	% Daily Value
Calories: 139	
Fat 7 g	11%
Saturated Fat 3 g	15%
+ Trans Fat 0 g	
Cholesterol 45 mg	15%
Sodium 75 mg	3%
Carbohydrate 17 g	6%
Fibre 0 g	0%
Sugars 9 g	
Protein 2 g	
Vitamin A 2%	Vitamin C 0%
Calcium 2%	Iron 4%

**Chocolate Cookie**

INGREDIENTS: organic pastry flour (organic whole grain white **wheat**), organic evaporated cane juice, organic butter (cream, salt), organic dark chocolate chips (organic cacao mass, organic evaporated cane juice, organic cacao butter, may contain non-GMO **soy lecithin**), organic whole **eggs**, organic sunflower oil, organic vanilla extract, organic molasses, baking powder, baking soda, sea salt.

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
- **PCD** - Kodak Photo-CD file
- **PCT, PICT, PIC** - Macintosh PICT Format
- **PCX** - PC Paintbrush Format
- **PDF/A** - Document Format for long term preservation
- **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
- **PDF Multi-page** - Portable Document Format
- **PFM** - Portable Float Map
- **PGM** - Portable Graymap BINARY
- **PGM RAW** - Protoble 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

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](#)
- › [«Label Preview» Tab](#)
- › [«Memory Card» Tab](#)
- › [«Logging» Tab](#)
- › [«File Locations» Tab](#)

«General» Tab

In this tab, you can change various general settings.

Among other things, you can select how to **Labelstar Office** behaves when the program starts:

- **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 dialog box 'Open File' to choose a label.

«Printing» Tab

In this tab, you can change various printing options.

In this tab, you can change various printing options.

Labelstar Office uses at first the Windows default printer but you can select another default printer for the printouts. The Windows default printer and the **Labelstar Office** default printer are independent. If you change one of the two default printers this does not affect the other printer.

The default printer which you select for **Labelstar Office** is a program setting, i.e. all labels that you print with **Labelstar Office**, were printed on this printer if you do not select another one with the label.

«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 parallel 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.

Gap length	2,00 mm
Label height	60,00 mm
Label type	Adhesive labels
Label width	100,00 mm
Snap Lines...	
▲ Printing	
Label rotation	180°
Log print job	Marked fields only
Print background image	<input type="checkbox"/> No
Printer	(Vario III 107/12)
Use temporary printer date	<input type="checkbox"/> No
Page Setup...	
Shift Definitions...	
Printing Preferences...	
▲ Settings	
Preview image	<input type="checkbox"/> (None)
Save label preview	<input checked="" type="checkbox"/> Yes
Comment...	

«Memory Card» Tab

Required program variant **BASIC, PROFESSIONAL**

For more information, see [Program Variants](#).

In this tab you can change different Memory Card options, e.g. the standard directory for printer drive and system drive.

«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 (e.g. `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 (in MB) Select this option to permit the creation of additional 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.

«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.

Print Only

With this program you can open and print labels.

Labelstar Office Print Only

FILE PRINTERS LABELS VIEW

Printers

- Vario III 107/12 Ready
- Compa 104/8 Ready
- Compa II 106/12 Not available

Label Label Info Printer Info

COOKIES

Search

ProductCode	ProductName
32500	Chocolate Coo...
32505	Double Chocol...
33200	Orange Chocol...
33300	Lemon Chocol...
33500	Peppermint Ch...
34100	Caramel Choco...

Nutrition Facts
Per 1 cookie (28g)

Amount	% Daily Value
Calories: 139	
Fat 7 g	11%
Saturated Fat 3 g	15%
+ Trans Fat 0 g	
Cholesterol 45 mg	15%
Sodium 75 mg	3%
Carbohydrate 17 g	6%
Fibre 0 g	0%
Sugars 9 g	
Protein 2 g	
Vitamin A 2%	Vitamin C 0%
Calcium 2%	Iron 4%

Chocolate Cookie

INGREDIENTS: organic pastry flour (organic whole grain white wheat), organic evaporated cane juice, organic butter (cream, salt), organic dark chocolate chips (organic cacao mass, organic evaporated cane juice, organic cacao butter, may contain non-GMO soy lecithin), organic whole eggs, organic sunflower oil, organic vanilla extract, organic molasses, baking powder, baking soda, sea salt.

NUMBER OF COPIES
1

Print

Version 4.30 Build 1010

Tools

The **LABELSTAR OFFICE** provides various tools to manage and change program data.

Installed Tools



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.

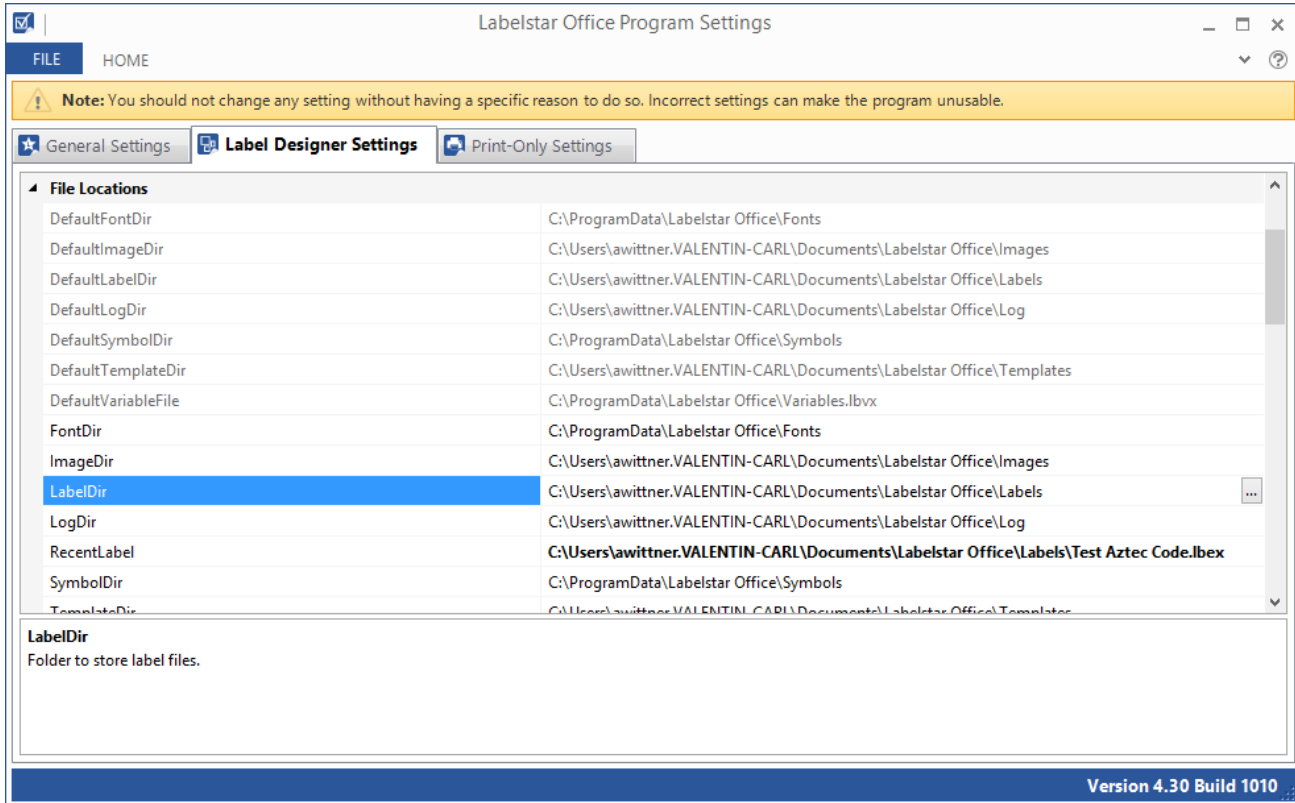


[Language Settings](#)

With this application you can select the language which is used for the user interface (e.g. in the menus, dialog boxes and help files).

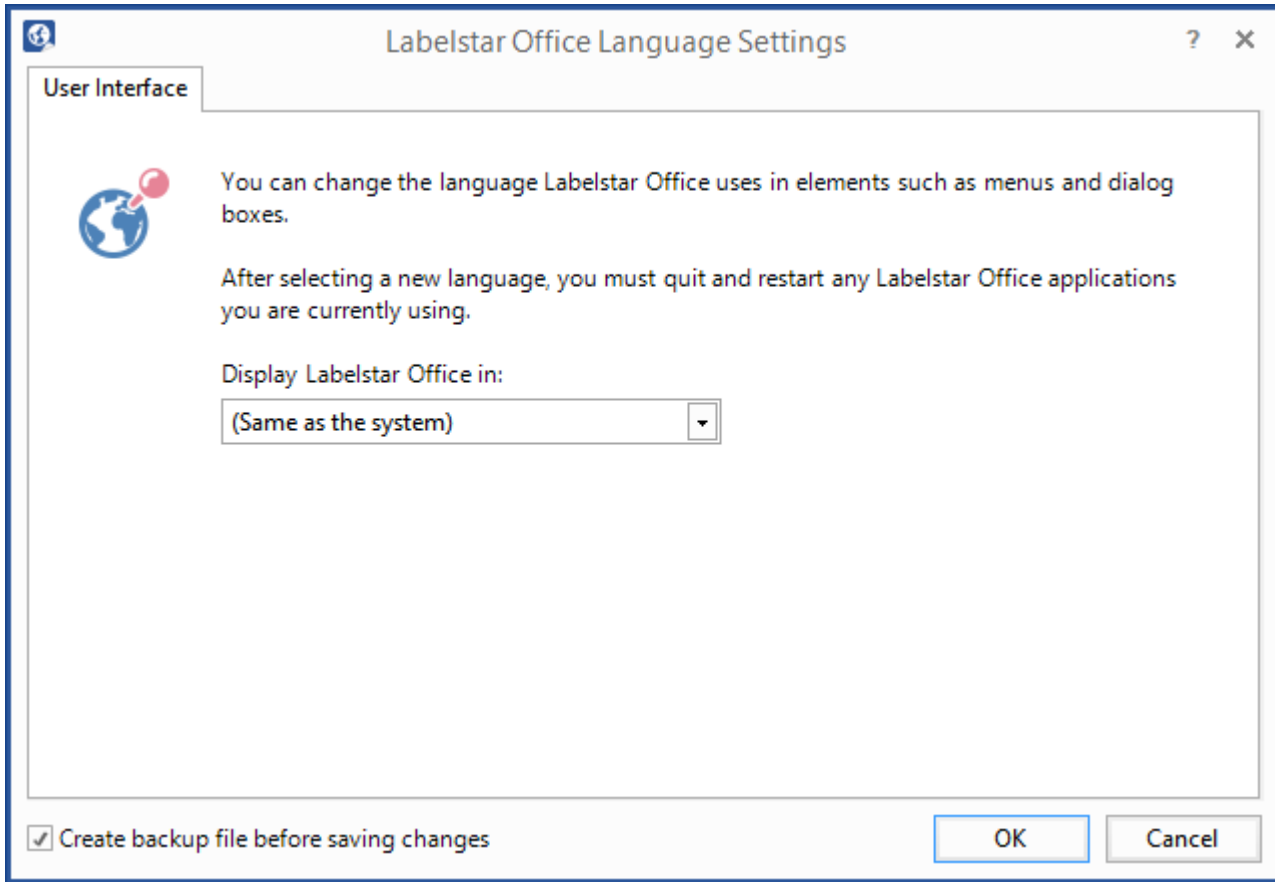
Program Settings

- 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**.



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.

OLE Automation

Required program variant **PROFESSIONAL**

For more information, see [Program Variants](#).

OLE Automation provides a way for OLE-compliant applications to interact with **Labelstar Office**. Using OLE, another program can start a **Labelstar Office** session, open a label, place data into a field, print a label, and save the updated label.

Quick Reference

[Application Class](#)

Represents the **Labelstar Office** application. This is the top level object from which all other objects will originate.

[Label Class](#)

Provides access to an individual label.

[Field Class](#)

Provides access to an individual field on the label.

Note

Your development computer must have a copy of the OLE automation's application before you can test your application. Also, your user must have a copy of the OLE automation application. Without **Labelstar Office** you cannot use OLE automation to open and print a label.

What is OLE Automation?

In Microsoft Windows applications programming, **OLE Automation** (later renamed by Microsoft to just Automation, although the old term remained in widespread use), is an inter-process communication mechanism based on Component Object Model (COM) that is intended for use by Scripting Languages -originally Visual Basic, but now many languages that run on Windows. It provides an infrastructure whereby applications called automation controllers can access and manipulate (i.e. set properties of or call methods on) shared automation objects that are exported by other applications. It supersedes Dynamic Data Exchange (DDE), an older mechanism for applications to control one another. As with DDE, in OLE Automation the automation controller is the "client" and the application exporting the automation objects is the "server".

What is COM?

Component Object Model (COM) is an interface standard for software componentry introduced by Microsoft in 1993. It is used to enable interprocess communication and dynamic object creation in any programming language that supports the technology. The term COM is often used in the software development industry as an umbrella term that encompasses the OLE, OLE Automation, and ActiveX, COM+DCOM technologies.

The essence of COM is a language-neutral way of implementing objects that can be used in environments different from the one they were created in, even across machine boundaries. For well-authored components, COM allows reuse of objects with no knowledge of their internal implementation, as it forces component implementers to provide well-defined interfaces that are separate from the implementation. The different allocation semantics of languages are accommodated by making objects responsible for their own creation and destruction through reference-counting. Casting between different interfaces of an object is achieved through the QueryInterface() function. The preferred method of inheritance within COM is the creation of sub-objects to which method calls are delegated.

Although the interface standard has been implemented on several platforms, COM is primarily used with Microsoft Windows. COM is expected to be replaced at least to some extent by the Microsoft .NET framework, and support for Web Services through the Windows Communication Foundation (WCF). However, COM objects can still be used with all .NET languages without problems. Networked DCOM uses binary proprietary formats, while WCF encourages the use of XML-based SOAP messaging. COM is very similar to other component software interface standards, such as CORBA and Java Beans, although each has its own strengths and weaknesses. It is likely that the characteristics of COM make it most suitable for the development and deployment of desktop applications, for which it was originally designed.

Operating Requirements

The OLE Automation Interface of **Labelstar Office** runs on Microsoft Windows operating systems family, it has been proven working on Windows 7, Windows 8 and Windows 8.1 in both 32-bit and 64-bit versions.

Labelstar Office requires the **.NET Framework 4.0 or higher**. Please visit <http://www.microsoft.com/net/> for additional information and download links.

Labelstar Office core is compiled using the **x86 Platform target**. This means the project is intended to run only as a 32-bit process. A 64-bit process will be unable to call into an assembly set as x86. Applications and assemblies marked for x86 can still run on 64-bit Windows. However they run under WOW64.

Register Assembly for COM Interop

Labelstar Office includes an OLE Automation Interface that can be used like a COM component with IntelliSense support in environment supporting this technology such as Visual Basic 6, html pages, Delphi and Visual FoxPro.

This assembly is automatically registered on your computer when installing the **Labelstar Office** package.

The dll is located in the installation folder and can be registered on other computers by using the *regasm* command with admin privileges:

```
%SystemRoot%\Microsoft.NET\Framework\v4.0.30319\regasm.exe LSOoffice.dll /codebase
```

Where *%SystemRoot%* is the path to your Windows installation, what is typically *C:\Windows* or *C:\WINNT*. The above examples assume that *LSOffice.dll* is in current working directory. Otherwise you need to specify an absolute path to the *.dll*.

Note

When deploying the assembly, make sure that **Labelstar Office** is installed along with the assembly on the target system.

Your First Application

Open the 32-bit version of the Visual Basic script editor.

Note

Labelstar Office core is compiled using the **x86 Platform target**. This means the project is intended to run only as a 32-bit process.

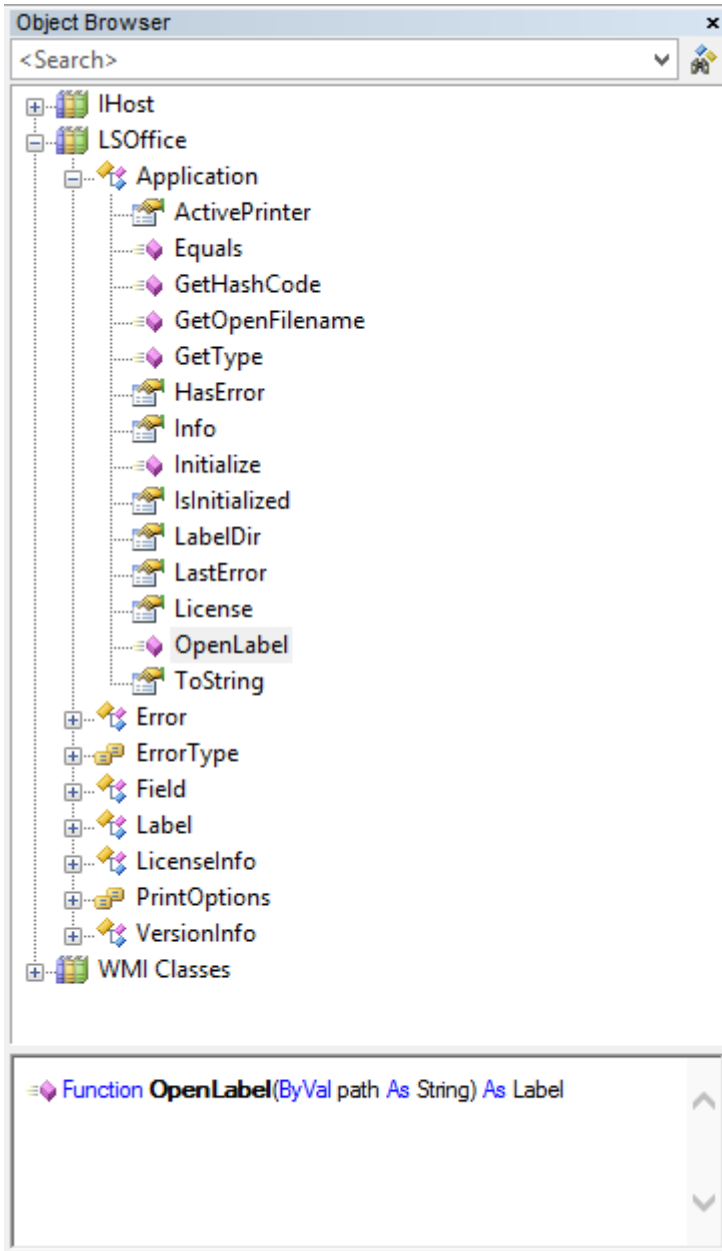
Referencing the Labelstar Office Type Library

To use **Labelstar Office** in a Visual Basic project, you must first add a reference to the **Labelstar Office** type library to the project.

To reference the **Labelstar Office** type library, proceed as follows:

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

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



Launch Labelstar Office runtime

Labelstar Office OLE 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()
```

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")

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

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

VBScript Samples

Note

Labelstar Office core is compiled using the **x86 Platform target**. Running **Labelstar Office** on 64-bit machines requires 32-bit script host, which is located in the **SYSWOW64** folder. By default, Windows 64-bit starts the 64-bit version of wscript.exe (the VBS interpreter). This results in the "800a01ad Active X component can't create object" error message.

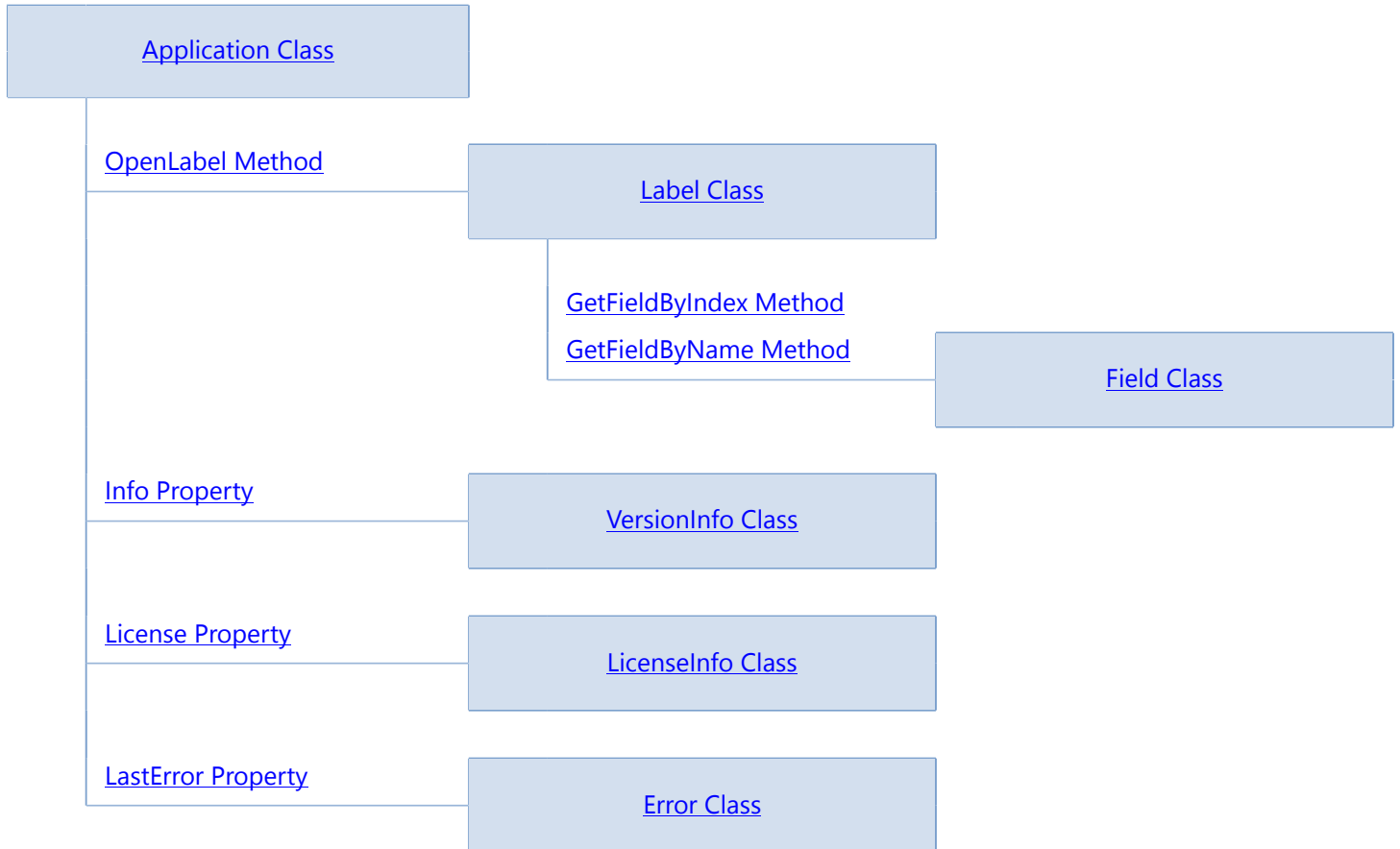
The sample scripts demonstrate how to open, modify and print labels. The scripts are located in the following directory:
`%InstallDir%\Samples\COM Interop\VBScript.`

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

Object Reference

The **Labelstar Office** application programming interface consists of OLE Automation objects that are created and referenced by client applications. These objects provide properties and methods that client applications can utilize.

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 declare 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	Initializes the current instance.
	GetOpenFilename	Displays the standard Open dialog box and gets a file name from the user without actually opening any files.
	OpenLabel	Opens the specified label.








See also

> [Object Reference](#)

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](#)
- > [Object Reference](#)

ActivePrinter Property

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

Namespace: LSOoffice

Assembly: LSOoffice.dll

Version: 4.10.1010

Usage

objApp.ActivePrinter

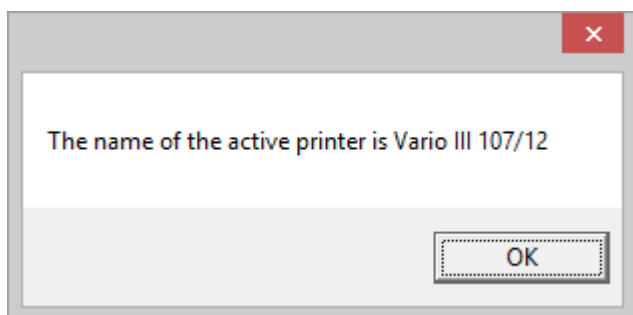
Type

String

Example (VBScript)

This example displays the name of the active printer.

```
1
2 Dim objApp
3 Set objApp = CreateObject ("LSOffice.Application")
4
5 objApp.Initialize()
6
7 If (objApp.HasError) Then
8     MsgBox objApp.LastError.Message
9     WScript.Quit
10 End If
11
12 MsgBox "The name of the active printer is " & objApp.ActivePrinter
13
```



See also

- > [Application Class](#)
- > [Object Reference](#)

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 [LastError](#) property.

See also

- [Application Class](#)
- [Object Reference](#)

Info Property

Refers to the [VersionInfo](#) object representing the version information. Read-only property.

Namespace: LSOoffice
Assembly: LSOoffice.dll
Version: 4.10.1010

Usage

objApp.Info

Type

[LSOoffice.VersionInfo](#)

See also

- [VersionInfo Class](#)
- [Application Class](#)
- [Object Reference](#)

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](#)
- [Object Reference](#)

LabelDir Property

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

Namespace: LSOOffice
Assembly: LSOOffice.dll
Version: 4.10.1010

Usage

objApp.LabelDir

Type

String

See also

- [Application Class](#)
- [Object Reference](#)

LastError Property

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

Namespace: LSOoffice
Assembly: LSOoffice.dll
Version: 4.10.1010

Usage

objApp.LastError

Type

[LSOoffice.Error](#)

Remarks

After method or property calls on all OLE Automation objects, this property is updated with the return status information of the call. To check whether the request has been successful, check [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 LSOoffice.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.lbx")

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

See also

- > [Error Class](#)
- > [Application Class](#)
- > [Object Reference](#)

License Property

Refers to the [LicenseInfo](#) object representing the license information. Read-only property.

Namespace: LSOoffice
Assembly: LSOoffice.dll
Version: 4.10.1010

Usage

objApp.License

Type

[LSOoffice.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](#)
- [Object Reference](#)

Application Methods

The [Application](#) type exposes the following members.

Methods

Name	Description
Initialize	Initializes the current instance.
GetOpenFilename	Displays the standard Open dialog box and gets a file name from the user without actually opening any files.
OpenLabel	Opens the specified label.

See also

- [Application Class](#)
- [Object Reference](#)

Initialize Method

Initializes the current instance.

Namespace: LSOoffice
Assembly: LSOoffice.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 a detailed example, see [OLE Automation -> Your First Application](#).

See also

- > [Application Class](#)
- > [Object Reference](#)

GetOpenFilename Method

Displays the standard **Open** dialog box and gets a file name from the user without actually opening any files.

Namespace: LSOOffice

Assembly: LSOOffice.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 a detailed example, see [OLE Automation -> Your First Application](#).

See also

➤ [Application Class](#)

➤ [Object Reference](#)

OpenLabel Method

Opens the specified label.

Namespace: LSOoffice

Assembly: LSOoffice.dll

Version: 4.10.1010

Usage

```
objApp.OpenLabel (path)
```

Parameters

path

Type: String

The path of the label.

Return Type

[LSOoffice.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 a detailed example, see [OLE Automation -> Your First Application](#).





See also

- > [Label Class](#)
- > [Application Class](#)
- > [Object Reference](#)

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

> [Object Reference](#)

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](#)
- > [Object Reference](#)

Details Property

Detailed message describing the error occurred. Read-only property.

Namespace: LSOOffice
Assembly: LSOOffice.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](#)
- [Object Reference](#)

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 occurred during the last call of a method or property. For more information, see [Error Codes and Messages](#).

See also

- [Error Class](#)
- [Object Reference](#)

ErrorType Property

Code indicating the type of the error occurred. Read-only property.

Namespace: LSOoffice
Assembly: LSOoffice.dll
Version: 4.10.1010

Usage

`objError.ErrorType`

Type

[LSOoffice.ErrorType](#)

Remarks

If this property is [ErrorType.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](#)
- [Object Reference](#)

Message Property

Message describing the error occurred. Read-only property.

Namespace: LSOOffice
Assembly: LSOOffice.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](#)
- [Object Reference](#)

ErrorType Enumeration

Specifies the error types.

Namespace: LSOoffice

Assembly: LSOoffice.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 potential 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](#)
- [Object Reference](#)

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

- [Object Reference](#)

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](#)
- [Object Reference](#)

FieldName Property

Gets the name of the field. Read-only property.

Namespace: LSOOffice
Assembly: LSOOffice.dll
Version: 4.10.1010

Usage

```
objField.FieldName
```

Type

String

See also

- [Field Class](#)
- [Object Reference](#)

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](#)
- [Object Reference](#)

Printable Property

Gets or sets a value that indicates whether the field is printed.

Namespace: LSOoffice

Assembly: LSOoffice.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.lbx")  
  
If (objLabel is Nothing) Then  
    WScript.Echo objApp.LastError.Message  
    WScript.Quit  
End If  
  
' Get field by name  
Set objField = objLabel.GetFieldByName("Barcode1")
```

```
If (objField Is Nothing) Then
    WScript.Echo objApp.LastError.Message
    WScript.Quit
End If

' Set property and print label
Dim msg
msg = MsgBox("Print barcode?", vbYesNo, objField.FieldName)

objField.Printable = (msg = vbYes)
objLabel.Print(1)
```

See also

- [Field Class](#)
- [Object Reference](#)

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](#)
- [Object Reference](#)

GetContent Method

Gets the content of the field.

Namespace: LSOOffice
Assembly: LSOOffice.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.lbx")  
  
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

' 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](#)
- [Object Reference](#)

GetPropertyValue Method

Gets the value of the specified property.

Namespace: LSOOffice
Assembly: LSOOffice.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.lbx")

    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](#)
- [Object Reference](#)

SetContent Method

Sets the content of the field.

Namespace: LSOoffice
Assembly: LSOoffice.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](#)
- [Object Reference](#)

SetPropertyValue Method

Sets the value of the specified property.

Namespace: LSOOffice
Assembly: LSOOffice.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](#)

➤ [Object Reference](#)

ImageFormat Enumeration

Specifies file save format.

Namespace: LSOoffice

Assembly: LSOoffice.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](#)
- › [Object Reference](#)

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

➤ [Object Reference](#)

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](#)
- [Object Reference](#)

ActivePrinter Property

Gets or sets the name of the active printer.

Namespace: LSOOffice
Assembly: LSOOffice.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) & vbCrLf
        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](#)
- › [Object Reference](#)

CurrentRecord Property

Gets or sets the one-based index of the current record to be printed.

Namespace: LSOOffice
Assembly: LSOOffice.dll
Version: 4.10.1010

Usage

objLabel1.CurrentRecord

Type

Integer

Remarks

To check if database fields are defined on the label use [IsDataAvailable](#) property.

See also

- [Label Class](#)
- [Object Reference](#)

FieldCount Property

Gets the number of fields defined on the label. Read-only property.

Namespace: LSOOffice
Assembly: LSOOffice.dll
Version: 4.10.1010

Usage

```
objLabel.FieldCount
```

Type

Integer

See also

- [Label Class](#)
- [Object Reference](#)

FieldNames Property

Gets the list of field names defined on the label. Read-only property.

Namespace: LSOOffice
Assembly: LSOOffice.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](#)
- > [Object Reference](#)

IsDataAvailable Property

Determines if there are database fields defined on the label. Read-only property.

Namespace: LSOoffice
Assembly: LSOoffice.dll
Version: 4.10.1010

Usage

objLabel.IsDataAvailable

Type

Boolean

See also

- [Label Class](#)
- [Object Reference](#)

LabelPath Property

Gets the path to the opened label. Read-only property.

Namespace: LSOOffice
Assembly: LSOOffice.dll
Version: 4.10.1010

Usage

```
objLabel.LabelPath
```

Type

String

See also

- [Label Class](#)
- [Object Reference](#)

MaxRecord Property

Returns the maximum number of records in the database. Read-only property.

Namespace: LSOoffice
Assembly: LSOoffice.dll
Version: 4.10.1010

Usage

objLabel1.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](#)
- [Object Reference](#)

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](#)
- [Object Reference](#)

PageName Property

Gets or sets the current page name.

Namespace: LSOOffice
Assembly: LSOOffice.dll
Version: 4.20.1040

Usage

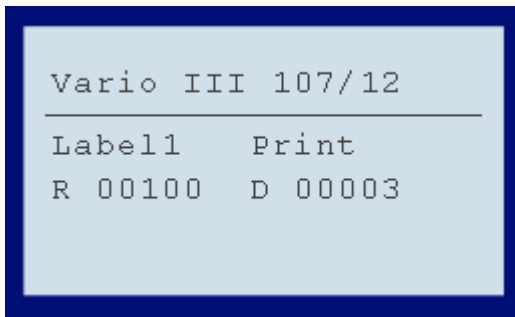
objLabel1.PageName

Type

String

Remarks

The page name is displayed in the printer display during printing.



See also

- [Label Class](#)
- [Object Reference](#)

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](#)
- [Object Reference](#)

GetFieldByIndex Method

Gets the field with the given index.

Namespace: LSOoffice

Assembly: LSOoffice.dll

Version: 4.10.1010

Usage

```
objLabel.GetField (index)
```

Parameters

index

Type: Integer

Zero-based index of the field.

Return Type

[LSOoffice.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](#)
- [Object Reference](#)

GetFieldByName Method

Searches for the field with the specified name.

Namespace: LSOoffice
Assembly: LSOoffice.dll
Version: 4.10.1010

Usage

```
objLabel1.GetField (fieldName)
```

Parameters

fieldName
Type: String
The string containing the name of the field to get.

Return Type

[LSOoffice.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](#)
- > [Object Reference](#)

GetPreview Method

Retrieves a preview image of the current label content.

Namespace: LSOOffice
Assembly: LSOOffice.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](#)
- [Object Reference](#)

GetPropertyValue Method

Gets the value of the specified property.

Namespace: LSOoffice

Assembly: LSOoffice.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](#)
- > [Object Reference](#)

Print Method

Prints the label.

Namespace: LSOoffice
Assembly: LSOoffice.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: [LSOoffice.PrintOptions](#)

Print options

Remarks

Check [Application.LastError](#) property to see if the function was completed successfully,

For more information and a detailed example, see [OLE Automation -> Your First Application](#) or [SelectRecord](#) method.

See also

- [Label Class](#)
- [Object Reference](#)

PrintToFile Method

Prints the label to a file.

Namespace: LSOoffice
Assembly: LSOoffice.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: [LSOoffice.PrintOptions](#)
Print options

See also

- [Label Class](#)
- [Object Reference](#)

Save Method

Saves changes to the label.

Namespace: LSOOffice
Assembly: LSOOffice.dll
Version: 4.10.1010

Usage

```
objLabel1.Save ()
```

Remarks

Check [Application.LastError](#) property to see if the function was completed successfully,

See also

- [Label Class](#)
- [Object Reference](#)

SaveAs Method

Saves changes to the label in a different file.

Namespace: LSOOffice
Assembly: LSOOffice.dll
Version: 4.10.1010

Usage

objLabel1.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](#)
- [Object Reference](#)

SavePreview Method

Saves a preview of the current label.

Namespace: LSOoffice
Assembly: LSOoffice.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](#)
- [Object Reference](#)

SelectRecord Method

Sets the current record to the first record matching the filter expression.

Namespace: LSOoffice
Assembly: LSOoffice.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 LSOoffice.PrintOptions
.....

Const PrintOptions_PrintCurrentRecord = 1
Const PrintOptions_PrintAllRecords = 2
Const PrintOptions_Default = 0

.....
' Print record
.....

Set objApp = CreateObject("LSOoffice.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.lbx")

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 &
"España" & 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](#)
- [Object Reference](#)

Filter Expression Syntax

This example describes syntax of filter expression. It shows how to correctly build a expression 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 pattern '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](#)
- [Object Reference](#)

Filter Expression Functions

The following functions are supported:

CONVERT

Description	Converts particular expression to a specified type.
Syntax	CONVERT (<i>expression</i> , <i>type</i>)
Arguments	<i>expression</i> -- The expression to convert. <i>type</i> -- 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 (<i>expression</i>)
Arguments	<i>expression</i> -- 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 (<i>expression</i> , <i>replacementvalue</i>)
Arguments	<i>expression</i> -- The expression to check. <i>replacementvalue</i> -- 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 (<i>expression</i> , <i>truepart</i> , <i>falsepart</i>)
Arguments	<i>expression</i> -- The expression to evaluated. <i>truepart</i> -- The value to return if the expression is true. <i>falsepart</i> -- 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 (<i>expression</i>)
Arguments	<i>expression</i> -- The expression to trim.

SUBSTRING

Description	Gets a sub-string of a specified length, starting at a specified point in the string.
Syntax	SUBSTRING (<i>expression</i> , <i>start</i> , <i>length</i>)
Arguments	<i>expression</i> -- The source string for the substring. <i>start</i> -- Integer that specifies where the substring starts. <i>length</i> -- Integer that specifies the length of the substring.
Example	Expression = "SubString (phone, 7, 8)"

See also

- > [Filter Expression Syntax](#)
- > [SelectRecord Method](#)
- > [Label Class](#)
- > [Object Reference](#)

SetPropertyValue Method

Sets the value of the specified property.

Namespace: LSOoffice

Assembly: LSOoffice.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](#)
- > [Object Reference](#)

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.




See also

➤ [Object Reference](#)

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.

See also

- > [LicenseInfo Class](#)
- > [Object Reference](#)

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](#)
- [Object Reference](#)

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](#)
- [Object Reference](#)

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

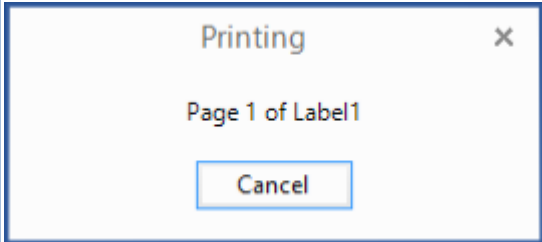

- [LicenseInfo Class](#)
- [Object Reference](#)

PrintOptions Enumeration

Specifies the print options.

Namespace: LSOoffice
Assembly: LSOoffice.dll
Version: 4.10.1010

Members

Name	Value	Description
Default	0 (0x00)	Default settings
PrintAllRecords	1 (0x01)	Prints all records. Select Record dialog box will not be shown.
PrintCurrentRecord	2 (0x02)	Prints the current selected record. Select Record dialog box will no 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](#)
- > [Object Reference](#)

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

➤ [Object Reference](#)

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: LSOOffice
Assembly: LSOOffice.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: LSOOffice
Assembly: LSOOffice.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 (except 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 already 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

Sample Runtime Error Handling (VBScript)

```
27
28 .....
29 ' Open and print label
30 .....
31 Set objApp = CreateObject("LSOffice.Application")
32
33 ' Application must be initialized before OpenLabel is called
34 objApp.Initialize()
35
36 If (objApp.HasError) Then
37     WScript.Echo objApp.LastError.Message
38     WScript.Quit
39 End If
40
41 ' Browse file name
42 Dim fileName
43 fileName = objApp.GetOpenFilename("Labels|*.lbex|All Files|*.*")
44
45 If (Len(fileName) = 0) Then
46     WScript.Quit
47 End If
48
49 ' Open label
50 Set objLabel = objApp.OpenLabel(fileName)
51
52 If (objLabel is Nothing) Then
53     WScript.Echo objApp.LastError.Message
54     WScript.Quit
55 End If
56
57 ' Print label
58 objLabel.Print(1)
59
```

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			•
Barcodes			
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, link field, 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 protocol (CVPL) (Carl Valentin Printer Drivers Version 2.3.1 or higher)		•	•
Print preview		•	•
Printing preferences		•	•
Two-color printing		•	•
Print Only			•
OLE Automation			•
Import of Labelstar PLUS labels		•	•

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.

What is "Converting"?

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. You can also enter another license key to release additional features and programs.

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 our 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 x86/x64
- .Net Framework 4.0 or higher (download from <http://www.microsoft.com/net/>)
- Microsoft Visual C++ 2010 Redistributable (x86)
- Microsoft Access Database Engine 2010 (x86)
- Recommended printer drivers: [Carl Valentin Printer Drivers](#) Version 2.3.1 or higher

Imprint

Carl Valentin GmbH
Neckarstrasse 78-86 u. 94
78056 Villingen-Schwenningen

Phone: +49 (0) 7720 9712 - 0
Email: info@carl-valentin.de

Copyright © 2014 Carl Valentin GmbH

All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission.

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.