DYMO®

Software Developer's Kit v. 7.8

Reference Manual

For Windows

Table of Contents

Chapter 1
Getting Started

Introduction

The DYMO Label Software (DLS) Software Development Kit (SDK) provides a simple, inexpensive, and reliable means of integrating specialized label printing into your application. You can quickly add professional quality label printing to your application using any of the following:

- C/C++
- C#
- Visual Basic
- Visual Basic for Applications
- VBScript
- Delphi
- Microsoft Access
- FoxPro
- Power Builder
- JavaScript
- any other application that supports COM, DDE, or direct DLL calls

The SDK provides the following features:

- Fully integrated printer control so you can forget about paper type selection, printer resolutions, margins, custom page sizes, and all the other complexities that combine to complicate printing from Windows.
- Advanced text handling, including rotation, curved text and shrink-to-fit.
- POSTNET barcodes for faster delivery of mail.
- Built-in support for photo ID applications.
- Built-in support for reading, re-sizing, and printing JPG, TIF, PNG, WMF, EMF and BMP graphics file formats.
- Built-in support for UPC, EAN, EAN 128, ITF-14, Code-128, Code 39, Interleaved 2 of 5, POSTNET, PLANET and other barcode symbologies.

And that's just the beginning. Using the capabilities of the SDK, you can enhance your applications or create entirely new ones. In fact, the DLS application was written using the SDK.

The LabelWriter printer can be used to automate many common labeling tasks, such as:

- Printing shipping labels from your corporate order-entry application.

- Printing barcoded inventory labels from your accounting application.
- Printing patient file folder labels and barcoded medical labels from your medical office automation application.
- Printing address labels and seminar badges for your meeting management program.

Whatever your needs, we are confident that the DYMO Label Software SDK and DYMO Label Software will enhance your product's capabilities and perceptions.

Before you continue

You should have an understanding of COM, DDE, and DLLs before attempting to use this SDK. This reference manual is not intended to teach you everything there is to know about COM, DDE or calling DLL functions.

What's in the SDK?

The SDK includes many samples showing how to print labels using the DLS SDK. Samples are provided in C++, C#, Visual Basic, Delphi, FoxPro, Microsoft Access and Power Builder, among others. In addition, since there are several ways to use the DLS SDK to print labels, you'll find samples illustrating each of the different approaches. All necessary files are supplied as part of the SDK.

If you are using a language other than those for which we provide samples, don't give up. The samples are designed to be easily understood and, with a good understanding of the technique being demonstrated (COM, DLL Calls, or DDE), you should be able to glean all the information you need to apply the techniques to your application.

Installing the DYMO Label Software SDK

You install the SDK in two steps: install the DYMO Label Software application and then install the SDK. Please see the *Quick Start Card* that came with your LabelWriter printer for instructions on installing the DLS application. If you downloaded the SDK from our web site, run the downloaded file to install the SDK. If you are installing the SDK from the CD that came with your LabelWriter printer, you will find the SDK's installation program (INSTALL.EXE) in the SDK directory on the CD. The default installation folder for the SDK is C:\Program Files\DYMO Label\SDK.

> ***NOTE**: All SDK demo programs assume that the DLS application is installed in a folder called C:\Program Files\DYMO Label. If you install the application in another folder, you need to edit the sample files, change the program paths, and recompile them. Otherwise, you need to manually launch the test application before running a DDE Demo program.*

When you install the SDK on your hard disk, all source code files and sample programs are also installed.

The installation creates folders for the high level COM, low level COM, DLL and DDE interfacing examples, as well as several other folders illustrating the use of the SDK from

Microsoft Internet Explorer and Mozilla Firefox browsers. Under each of these folders, you will find additional subfolders categorized by language (Access, VB, VC++, and so on).

Getting Help

Although we have made every attempt to provide clear, concise examples for integrating DYMO LabelWriter printers with your application, it is possible you may still have a question specific to your application. In this case, several avenues of help are available to you:

- The SDK includes an FAQ document that contains answers to the most common questions about using the SDK.
- DYMO maintains a web site at http://www.dymo.com. Select the “Developer Info” link at the bottom of the page and you will be redirected to a Developer’s Forum that includes the latest SDK samples, an FAQ sheet, and other useful information. While visiting, be sure to register for our infrequent, but informative Developers Mailing List.
- You can also send email to sdkhelp@dymo.com to get an answer by return email, usually in less than 24 hours.

Whichever method you use to submit your question, please be sure to include the following information:

- The DLS software version you are using (as shown on the Help/About dialog box).
- The programming language or application you are using.
- The exact nature of your problem, including excerpts of code where appropriate.

> ***NOTE:*** *Be sure to provide as much detail and be as specific as possible to minimize the response time to your query.*

Chapter 2
SDK Background

DLS Overview

The DLS SDK can be easily adapted to different label printing requirements and programming languages, from the simple needs of address label printing, to the complexities of a full-blown label printing program, similar to DLS. The SDK provides four different DLS programming interfaces grouped into two levels of functionality, high-level and low-level.

High-Level Functionality

The high-level functionality provides the easiest and most common label printing solutions. These functions allow an application to open an existing or customized label file, change text, barcode or address data, and print the label. The COM-based interface to the high-level functions is the most common interface method among those using the SDK.

The high-level functionality is supported by two technologies:

- COM
- DDE (for backwards compatibility only)

Although these two high-level interfaces are interchangeable, the COM interface includes some enhancements and additional functionality.

Low-Level Functionality

Unlike the high-level interfaces, the low-level interfaces provide developers with full control over the size, attributes, and appearance of labels. Everything that can be done through the high-level DLS interfaces can be done through these two low-level DLS interfaces as well, but require a great deal more effort to design, develop and debug. Printing, saving, and loading labels; creating new labels and modifying an existing label; access to the properties of any object on the label – all these are available using the low-level DLS interfaces.

The low-level functionality is supported by two technologies:

- COM
- Direct DLL Calls

Program Architecture

The DYMO Label Software architecture is shown in the figure below:

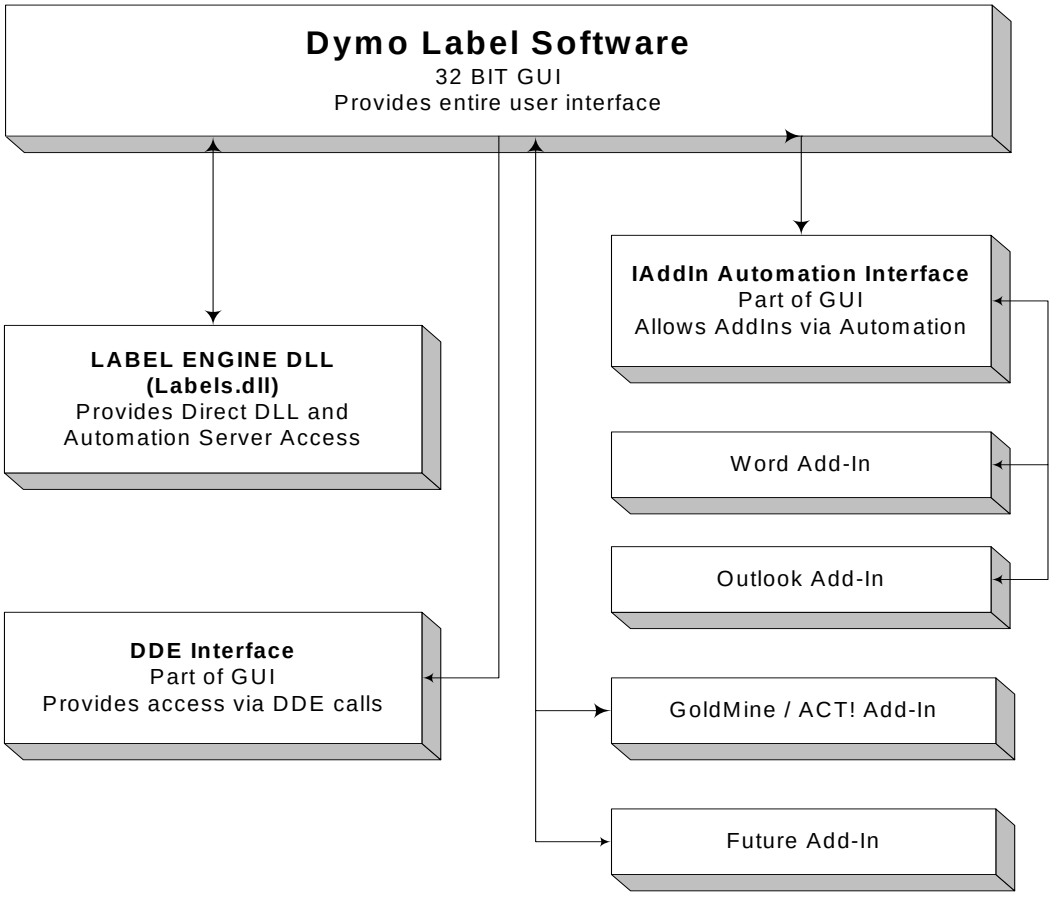

Figure 1

As shown in Figure 1, the low-level control is provided by the Label Engine DLL (for COM interface and DLL calls), whereas the high-level control is fully contained within the DLS executable.

Chapter 3
DLS Interface Descriptions

High-Level Interfaces

Purpose and Capabilities

The two high-level DLS interfaces (COM interface and DDE) provide very simple and often-used functions to interact with the DYMO Label Software application, such as printing labels, loading or saving labels, changing text in the label objects, and so on. Because of this, the main focus of these DLS interfaces is for writing small, simple programs. In addition to the samples included in this SDK, the Word and Outlook Add-In's are examples of programs that use the high-level interface.

With the introduction of the latest LabelWriter printer models, the SDK has been updated to support the added functionality provided by the LabelWriter Twin Turbo and LabelWriter Duo printers.

The COM Interface

As shown Figure 2, the COM interface is part of the DLS application itself (the GUI). It operates as a universal out-of-process COM server that provides integration with any COM-capable application. Its primary use is for simple label printing needs, such as on demand address or bar code label printing.

Program ID Information

The COM interface exports three main COM interfaces: **IDYMOAddIn**, **IDYMOLabels,** and **IDYMOTape** The COM class names for these three interfaces are:

DYMO.DYMOAddIn
DYMO.DYMOLabels
DYMO.DYMOTape

The **DYMOAddIn** COM object implements functions that affect the program operation, the **DYMOLabels** COM object implements functions that pertain to the printing of die-cut labels, and the **DYMOTape** COM object implements functions pertaining to the printing of tape labels using a LabelWriter DUO or other DYMO printer using D1 tape cassettes.

In addition to those interfaces described above, DLS also exports three updated versions of the **IDYMOAddIn** interface: **IDYMOAddIn2**, **IDYMOAddIn3** and **IDYMOAddIn4.** These interfaces are described below.

IDYMOAddIn Properties and Methods

The **IDYMOAddIn** interface provides the following program control functions. As noted above, these functions and properties pertain to the application itself, and in many cases are equivalent to built-in menu commands:

`FileName` This property returns the name of the currently open file.

`Open (const FileName: WideString)`

Opens a label file. Returns TRUE on success, FALSE on error.

`Save ()`

Saves the current label. Returns TRUE on success, FALSE on error.

`SaveAs (const FileName: WideString)`

Saves the current label under a new name. Returns TRUE on success, FALSE on error.

`Print (Copies: Integer; bShowDialog: WordBool)`

Prints the current label. Copies is the number of copies to print. bShowDialog controls the display of the print-progress dialog. If TRUE, then the dialog is displayed. Returns TRUE on success, FALSE on error.

> *Note: When the currently selected printer is a LabelWriter Twin Turbo, this command defaults to use the left roll. If you wish to control the roll from which to print, use the Print2() function provided in the **IDYMOAddIn3** interface (see below).*

`Hide ()`

Hides the program, removing it from the task bar.

`Quit ()`

Shuts down the program. Use this with caution, this can cause Windows errors. It is best to let Windows start/stop the program.

`Show ()`

Shows the program, bringing it to the foreground.

`GetDymoPrinters ()`

Returns a list of DYMO printer names. The vertical-bar '¦' character separates each printer name in the list.
Example:
DYMO LabelWriter 400 Turbo-USB¦DYMO LabelWriter 330-USB

`SelectPrinter (const PrinterName: WideString)`

Redirects output to the selected printer. `PrinterName` is of the form "Printer name" on "Port." Returns TRUE on success, FALSE on error.
Example:
To select the LabelWriter EL60 on COM3, you would use the command:
SelectPrinter(DYMO LabelWriter EL60 on COM3:)

`SysTray (State: WordBool)`

Places the program in the system tray when State is True, and removes it from the system tray if State = False.

IDYMOAddIn2 Properties and Methods

The **IDYMOAddIn2** interface inherits directly from the **IDYMOAddIn** interface and provides the following additional program control functions:

`Open2 (const FileName: WideString)`

Opens a label file. If the specified file name is not found, opens the Label File Open dialog box. Returns TRUE on success, FALSE on error.

`GetMRULabelFiles ()`

Returns a list of the Most Recently Used (MRU) label file names. The files names include the full path and file extension. The vertical-bar '¦' character separates each file name in the list.
Example:
C:\Program Files\DYMO Labels\Label Files\Address (30252, 30320).LWL¦C:\Program Files\DYMO Labels\Label Files\Shipping (30256).LWL

`GetMRULabelFileCount ()`

Returns the number of files in the MRU label file list.

`GetMRULabelFileName (Index: Integer)`

Returns a label file name from the MRU label files list. The Index parameter identifies which file name in the MRU to return. The index is zero-based and the file name DOES NOT include the file path or extension. Example: "*Address (30252, 30320)*"

`OpenMRULabelFile (Index: Integer)`

Opens a label file in the MRU label file list. The Index parameter identifies which file in the MRU to open. The index is zero-based. The Label File Open dialog box appears if the index is out of range or if the MRU label file no longer exists. Returns TRUE on success, FALSE on error.

IDYMOAddIn3 Properties and Methods

The **IDYMOAddIn3** interface inherits directly from the **IDYMOAddIn2** interface and provides the following additional program control functions. The new functions were added to support the LabelWriter Twin Turbo printer.

> ***Note:*** *For consistency with Windows printer drivers, the LabelWriter Twin Turbo printer uses the paper tray selection methods to select the roll being printed. It supports two "Trays", the Left Tray (or Roll) and the Right Tray (or Roll).*

`Print2 (Copies: Integer; bShowDialog: WordBool; Tray: Integer)`

In addition to providing the same functionality as the Print() function in the IDYMOAddIn interface, Print2() function allows the caller to specify which paper tray or roll to print from. Possible values for the "Tray" parameter include:

0 - Left Roll
1 - Right Roll
2 - Auto Switch - This puts the printer in a mode where it starts to print from the last printed roll and automatically switch over to the other roll when the starting roll runs out of paper. It continues to toggle back and forth between rolls as long as the user refills rolls once they become empty. This mode of printing is useful when the user is printing a large number of labels.

`GetCurrentPaperTray ()`

Returns the current active paper tray.

Further Information: When the currently selected printer is a LabelWriter Twin Turbo, DLS attempts to associate a paper tray with the currently opened label file. As an example, if the last label

printed on the left tray was "Address 30252" and the last label printed on the right tray was "Shipping 30323," then when the user opens a label file that uses the "Address 30252" paper size, DLS automatically sets the tray selection to left tray.

Possible return values include:

-1 = Unknown Tray (program user must specify)
0 = Left Tray (Roll)
1 = Right Tray (Roll)
2 = Auto Switch (See description for Print2, above)

IDYMOAddIn4 Properties and Methods

The **IDYMOAddIn4** interface inherits directly from the **IDYMOAddIn3** interface and provides the following additional program control functions. These new functions were added to support the LabelWriter Twin Turbo printer and improve LabelWriter print job handling.

`GetCurrentPrinterName ()`

Returns the name of the currently selected printer.

`IsTwinTurboPrinter (PrinterName: WideString)`

Returns TRUE if the specified printer ("PrinterName") is a LabelWriter Twin Turbo, False otherwise.

`StartPrintJob (), EndPrintJob ()`

These optional commands can be wrapped around calls to the Print() or Print2() functions to optimize print speed, particularly when using the LabelWriter 400, 400 Turbo, Twin Turbo or Duo printers.

IDYMOAddIn5 Properties and Methods

The **IDYMOAddIn5** interface inherits directly from the **IDYMOAddIn4** interface and provides the following additional program control functions. These new functions were added to support the use of the DYMO SmartPaste functionality. The SmartPaste functionality processes a block of data in either the Clipboard or a file and turns it into a collection of records. For more information regarding the use of SmartPaste in DLS, refer to the DLS help file and user manual.

`SmartPasteFromClipboard ()`

Parses text data in the Clipboard into records and prints a label for each record.

Returns TRUE if the operation was successful, FALSE if the operation failed.

`SmartPasteFromFile (FileName: WideString)`

Parses comma or tab delimited data in a file into records and prints a label for each record.

Returns TRUE if the operation was successful, FALSE if the operation failed.

`SmartPasteFromString (StrBuf: WideString)`

Parses text data in StrBuf into records and prints a label for each record.

Returns TRUE if the operation was successful, FALSE if the operation failed.

`OpenURL (URLFileName: WideString)`

Opens a label file using a URL. The URL can start with http, https, ftp, or file, etc.

Return TRUE if the file was opened successfully, FALSE if the file does not exist.

`OpenStream (Buffer: VARIANT)`

Reads a label file from a buffer (vs. from a file or URL). This is useful if you intend to manage the binary data yourself. The VARIANT must be a “byte array” filled with the binary data of the label file.

Returns TRUE if the label file was read from the buffer correctly. FALSE if the buffer is invalid.

`SaveStream ()`

Returns a VARIANT, which is actually a “byte array” containing the binary data of the label file currently open in DLS.

IDYMOLabels Properties and Methods

The IDYMOLabels COM object is used for interacting with the contents of labels. It allows modification of the contents of a label opened using the IDYMOAddIn object or its variants.

The **IDYMOLabels** interface provides the following functions for controlling the appearance or contents of a label:

`GetObjectNames (bVariableOnly: WordBool)`

Returns a list of objects on the label. If bVariableOnly is TRUE, then the list contains only those objects that can be pasted into. This includes address, barcode, and text objects with the bVariable property set to TRUE. If bVariableOnly is FALSE, then all objects on the label are returned. The vertical bar ‘¦’ character separates each object in the list. Example: “*Text*¦*Address*¦*Logo*.”

`GetText (const Field: WideString)`

Given an object name (Field) returns the contents of the object. This operation only applies to address, barcode, and text objects.

`SetAddress (AddrIdx: Integer; const Address: WideString)`

Given an index of an address object, places the text in the object. The index is normally 1, but for designs with more than one address object, the index can be greater than one to select other address objects. Returns TRUE on success, FALSE on error.

`SetField (const Field, Text: WideString)`

Given an object name, and some text, changes the text of the object to have the new text. This operation only applies to address, barcode, and text objects. Returns TRUE on success, FALSE on error.

`POSTNET (Index: Integer; const Position: WideString)`

Given an index of an address object, changes the POSTNET barcode setting for the object. The index is normally 1, but for designs with more than one address object, the index can be greater than one to select other address objects. Position can be “NONE,” “TOP,” or “BOTTOM.”

`AddressFieldCount`

This property returns the number of address objects on the current label. Used to determine possible values for the index parameter of the POSTNET and SetAddress functions.

`PasteFromClipboard (const ObjectName: WideString)`

Paste text from the clipboard to an object by the given ObjectName. Returns TRUE on success, FALSE on error.

`SetImageFile (const ObjectName, FileName: WideString)`

Load an image file to a graphic object by the given ObjectName. Returns TRUE on success, FALSE on error.

IDYMOLabels2 Properties and Methods

The **IDYMOLabels2** interface inherits from the IDYMOLabels interface and provides the following additional function:

`PasteImageFromClipboard (const ObjectName: WideString)`

Paste an image from the clipboard to an object by the given ObjectName. Returns TRUE on success, FALSE on error.

IDYMOTape Properties and Methods

Tape labels are permanent plastic, polyester,or nylon labels. The LabelWriter Duo printer, as well as the LMPC, ExecuLabel LM450 and ExecuLabel LP350 printers, can all print labels using DYMO D1 tape cassettes. The **IDYMOTape** interface provides the following program properties and functions to support printing tape labels:

`FileName`

This read only property indicates the name of the currently open label file. Returns an empty string if tape printing is not supported (for example, no LabelWriter Duo or other DYMO tape printer installed).

`LabelOrientation`

This read/write property indicates the tape label's orientation. Possible values:

0 = Horizontal
1 = Vertical

`TextAlignment`

This read/write property indicates the text alignment on the tape label. Possible values:

0 = Left (default)
1 = Right
2 = Center
3 = Center Block

`LabelLength`

This read/write property indicates the tape label's length. Possible values:

0 = automatic length (default)
# = label length in TWIPS (1 in. = 1440 TWIPS)

`BorderMode`

This read/write property controls the label's border. Possible values:

0 = None = No border
1 = Thin line = thin border
2 = Medium line = medium thick border
3 = Thick line = thick border
4 = Long dash = long dash lines
5 = Short dash = short dash lines

`SysTray`

This boolean read/write property controls whether DLS runs from the system tray.

`Open (const FileName: WideString)`

Opens a label file. Returns TRUE on success, FALSE on error.

`New ()`

Creates a new label file. Returns TRUE on success, FALSE on error.

`Save ()`

Saves the current label. Returns TRUE on success, FALSE on error.

`SaveAs (const FileName: WideString)`

Saves the current label under a new file name. Returns TRUE on success, FALSE on error.

`Print (Copies: Integer)`

Prints the current label. Copies is the number of copies to print.

`Hide ()`

Hides the program, removing it from the task bar.

`Quit ()`

Shuts down the program. Use this with caution, this can cause Windows errors. It is best to let Windows start/stop the program.

`Show ()`

Shows the program, bringing it to the foreground.

`GetDymoPrinters ()`

Returns a list of DYMO printers installed. The vertical-bar '¦' character separates each printer name in the list.
Example:
DYMO LabelWriter DUO-USB¦DYMO LabelManager 450-USB

`SelectPrinter (const PrinterName: WideString)`

Redirects output to the selected printer. PrinterName is of the form "Printer name" on "Port." Returns TRUE on success, FALSE on error.
Example:
SelectPrinter("DYMO LabelWriter Duo-USB")

`AddText (Text: WideString; Format: WideString; OutlineEffect: Boolean; ShadowEffect: Boolean)`

Appends the specified text to the current text on the tape label. The font format of the text is specified by the Format parameter. Returns TRUE if text is appended, FALSE on error.

Fonts are represented by strings in the following form: “<Font Name>, <Size>, <Style(s)>”

Example: “*Times New Roman, 12, Bold*”

Styles can be any combination of Bold, Italic, Underline, and Strikeout.

`AddBarcode (Text: WideString; Format: Integer; Size: Integer; TextPosition: Integer)`

Appends a barcode representation of the specified text to the current text on the tape label.

Possible Values for the Format parameter:

1 = Code 39
2 = Code 39 with Checksum
3 = Code 128 Auto
4 = Code 128-A
5 = Code 128-B
6 = Code 128-C
7 = Interleaved 2 of 5
8 = UPC A
9 = UPC E
10 = EAN 8
11 = EAN 13
12 = Codabar
13 = Code 3 of 9 Lib, R-L
14 = Code 3 of 9 Lib, L-R
15 = Codabar Lib, R-L
16 = Codabar Lib, L-R
17 = UCC/EAN 128

Possible Values for the Size parameter:

0 = Small
1 = Medium
2 = Large

Possible Values for the TextPosition parameter:

0 = None
1 = Above
2 = Below

The function returns TRUE if the barcode object is inserted correctly, FALSE on error.

`AddPicture (FileName: WideString)`

Appends a graphic object specified by the FileName parameter to the current text on the tape label. The function returns TRUE if text is appended, FALSE on error.

Currently supported graphic formats are *.jpg, *.bmp, *.tif, *.ico, *.wmf, *.emf.

`PasteText ()`

Paste text from the system Clipboard and append it to the current text on the tape label. The function returns TRUE if text is appended, FALSE on error.

`PastePicture ()`

Paste a picture object from the system Clipboard and append it to the current text on the tape label. The function returns TRUE if text is appended, FALSE on error.

`AddTabStop (Position: Integer; TabStyle: Integer)`

Inserts a tab stop at the specified position on the tape label (the position’s unit is in TWIPS (1440 twips = 1 in.)).

Possible values for the TabStyle Parameter:

0 = Left Align
1 = Right Align
2 = Center Align
3 = Decimal Point Align

The function returns a unique TabID of the inserted tab, -1 on error.

`DeleteTabStop (TabID: Integer)`

Deletes the tab identified by the TabID. The function returns TRUE if text is appended, FALSE on error.

`IsTapeWidthSupported (Width: Integer)`

The function returns TRUE if the specified tape width is supported by the printer, FALSE otherwise.

Possible values for the Width parameter:

0 = 1/4 inch or 6mm
1 = 3/8 inch or 9mm
2 = 1/2 inch or 12mm
3 = 3/4 inch or 19mm
4 = 1 inch or 24mm

`SetTapeWidth (Width: Integer)`

Set the width of the current tape. Returns TRUE if successful, FALSE on error.

Possible values for the Width parameter:

0 = 1/4 inch or 6mm
1 = 3/8 inch or 9mm
2 = 1/2 inch or 12mm
3 = 3/4 inch or 19mm
4 = 1 inch or 24mm

IDYMOTape2 Properties and Methods

The **IDYMOAddIn2** interface inherits directly from the **IDYMOTape** interface and provides the following additional program control functions. These new functions were added to support the use of the DYMO SmartPaste functionality. The SmartPaste functionality processes a block of data in either the Clipboard or a file and turns it into a collection of records. For more information regarding the use of SmartPaste in DLS, refer to the DLS help file and user manual.

`SmartPasteFromClipboard ()`

Parses text data in the Clipboard into records and prints a label for each record.

`SmartPasteFromFile (FileName: WideString)`

Parses comma or tab delimited data in a file into records and prints a label for each record.

Dynamic Data Exchange

Introduction

Dynamic Data Exchange (DDE) always involves two applications: a client (your own application) and a server (DLS). Using DDE, your application sends data and commands to the DLS application, in effect operating DLS through remote control. In all cases, you will be using the DLS application as the server and adding DDE client capabilities to your application.

The DLS interface using DDE provides label printing capabilities similar to those provided by the high-level COM interface. Using DDE is primarily for simple label printing needs, such as on demand address or barcode label printing.

> *Note: The DDE functionality of the SDK is provided for backwards compatibility and should not be used for new development unless absolutely necessary.*

Using DDE With DLS

DDE commands are all executed using DDE execute commands. Prior to sending an execute command to the server, the Status item must be in the 'Ready' state. If the Status is not in the 'Ready' state, all commands sent to the server are ignored.

Multiple DDE commands can be combined, up to a maximum string length of 255 characters. Multiple commands can be separated by commas. For example, the command string:

Open(MYFILE.LWL), Paste(Address), print, hide

opens the label file MYFILE.LWL, pastes the clipboard's contents into an object on the label called 'Address', prints the label, and then hides the application.

DDE commands sent to DLS can use carriage return/line feed characters (decimal 13,10) or "Pipe" characters ('¦', decimal 124) to indicate line breaks. For example, to set an object called Address to contain the text:

Acme Industries
123 Main St.
Anytown, CA 99812-1234

You would send the command:

SetObjectText (Address, Acme Industries¦123 Main St.¦Anytown, CA 99812-1234)

System/Topic/Items Information

To establish a DDE conversation, the DDE client must specify an Application or Service Name and a Topic. If the server application is running, the conversation is established. If not, the application must launch the server application and try again.

Executable Name:	**DYMOLBL.EXE**
DDE Application/Service Name	**DYMO**
DDE Topic Name	**SYSTEM**

The path to the executable can be found in the Registry at:

HKEY_LOCAL_MACHINE\SOFTWARE\DYMO\LabelWriter\InstallPath

DLS responds to the following DDE items:

Status

Returns current DDE status as a string. The returned string is either ‘Busy’ or ‘Ready’.

The Status item is used to check the current status of the DLS program. When the program is ready to accept new commands, a DDE request of the Status item returns Ready. When the program is processing a command, a DDE request of the Status item returns Busy. When the program is done processing a command, the DDE request once again returns Ready.

Fields

Returns a list of the names of all objects on a label as a null-terminated string. The object names are separated by the pipe character, (‘¦’, decimal 124) and the last character is followed by a null (decimal 0). For example, a typical shipping label with a return address and logo would return the following when it receives a request of the Fields item:

RETURN ADDRESS¦LINE¦LOGO¦ADDRESS

LastError

When a command fails, the LastError item returns a descriptive (but brief) textual error message and the Status reverts to Ready. If the LastError item is read while the status is Busy or Ready, it returns No error. Other possible error codes include:

Error Message	Meaning
Command error	The format of the command string was incorrect.
Open file error	The Open command specified a file that was not found.
Print error	An error occurred while trying to print the label.
Object not found	A Paste or SetObjectText command specified an object that was not found on the label.
Unknown Command	The command was not found or was syntactically wrong.

> *NOTE: All strings (label object names, text, number of copies, filenames, etc.) are represented by standard C-style null-terminated strings. If you are using Delphi, you may need to use the StrPCopy() procedure to convert a Pascal string to a null-terminated string.*

DDE Commands

The following list represents all available DDE commands. The commands are all performed using the DDE Execute command of the client application.

`Exit`

Shuts down DLS. If the template has been modified through DDE calls, the user is NOT prompted to save changes to the file.

`Maximize`

Maximizes DLS to full screen

`Minimize`

Minimizes DLS to an ICON.

`MiniTool`

Minimizes DLS to the System Tray.

`Restore`

Restores DLS to normal size.

`Hide`

Hides the DLS program.

`Show`

Shows the DLS program if it was hidden.

`Print`

Prints the current label as is.

`Copies(nCopies)`

Sets the number of copies of a template to be printed.
Example:
Copies(4)

`Open(FileName)`

Opens the specified label file.
Example:
Open(C:\LABELS\\MYLABEL.LWL)

`Paste(ObjectName)`

Pastes the text from the clipboard into the specified object's text property. If the object has no text property, the command is ignored.
Example:
Paste(Address)

`SetObjectText(Object Name, Text)`

Similar to the Paste command, except the text in the object is specified as part of the command.
Example:
SetObjectText(Name, John Doe)

`SetImageFile(ObjectName, FileName)`

Replaces the specified graphic object's image with an image read from the specified file. The file name should contain the complete drive and path information for the file.
Example:
SetImageFile(Logo, C:\CLIPART\MYLOGO.BMP)

`PastePicture(ObjectName)`

Copies an image from the clipboard into the specified picture object. Example:
PastePicture(Photograph)

`SelectPrinter(Name)`

Redirects output to the selected printer. Name is of the form *Printer name on Port*.
Example:
To select the LabelWriter EL60 on COM3, you would use the command:
SelectPrinter(DYMO LabelWriter EL60 on COM3:)

`POSTNET(nSetting)`

Modifies the POSTNET barcode setting for all addresses on a label. nSetting can be 0, 1, or 2, and sets the position of the POSTNET bar code as follows:
0 = No barcode
1 = Print barcode above the address
2 = Print barcode below the address
Example:
Postnet(1)

Low-Level Interfaces

Purpose and Capabilities

The low-level DLS interfaces provide full control of the label printing process. Printing, saving, loading labels, creating new labels, changing the properties of any object on the label – all of this can be done using the low-level DLS interfaces. This type of DLS interface is used to write the user-interface components of DLS.

ILabelEngine COM Interface

Introduction

As is shown in Figure 1 on page 6, the ILabelEngine COM interface is implemented in the label engine DLL (LABELS.DLL). This DLL operates as an in-process (in-proc) COM server that provides integration with any COM-capable application. ILabelEngine automation is far more powerful (and trickier to use) than the high-level automation. It is targeted for those needing to have complete control over the design, printing, and preview of labels. If your application requires on-the-fly label design, then this DLS interface is appropriate. If all you need to do is

open a label file change the contents of an object, and print it, then use the high level automation instead.

Program ID Information

Because of the high level of control provided by this automation server, a number of COM interfaces are exported as shown in the table below. All of these are accessible through the ILabelEngine Interface.

Interface Name	COM Class Name
ILabelEngine	DYMO.LabelEngine
ILabelEngine2	DYMO.LabelEngine
ILblInfo	DYMO.LblInfo
ILabelList	DYMO.LabelList
IPrintObject	DYMO.PrintObject
IObjectsAtEnum	DYMO.ObjectsAtEnum
IObjectList	DYMO.ObjectList
IVarObjectList	DYMO.VarObjectList
ITextAttributes	DYMO.TextAttributes
ITextAttributes2	DYMO.TextAttributes
ICircularTextAttributes	DYMO.CircularTextAttributes
ICircularTextAttributes2	DYMO.CircularTextAttributes
ILabelObject	DYMO.LabelObject
ITextObj	DYMO.TextObj
ICircularTextObj	DYMO.CircularTextObj
IAddressObj	DYMO.AddressObj
IGraphicObj	DYMO.GraphicObj
IRectObj	DYMO.RectObj
ILineObj	DYMO.LineObj
ICounterObj	DYMO.CounterObj
IBarCodeObj	DYMO.BarCodeObj
IDateTimeObj	DYMO.DateTimeObj

Object Model

The object model for the label engine DLL is shown in the two figures that follow:

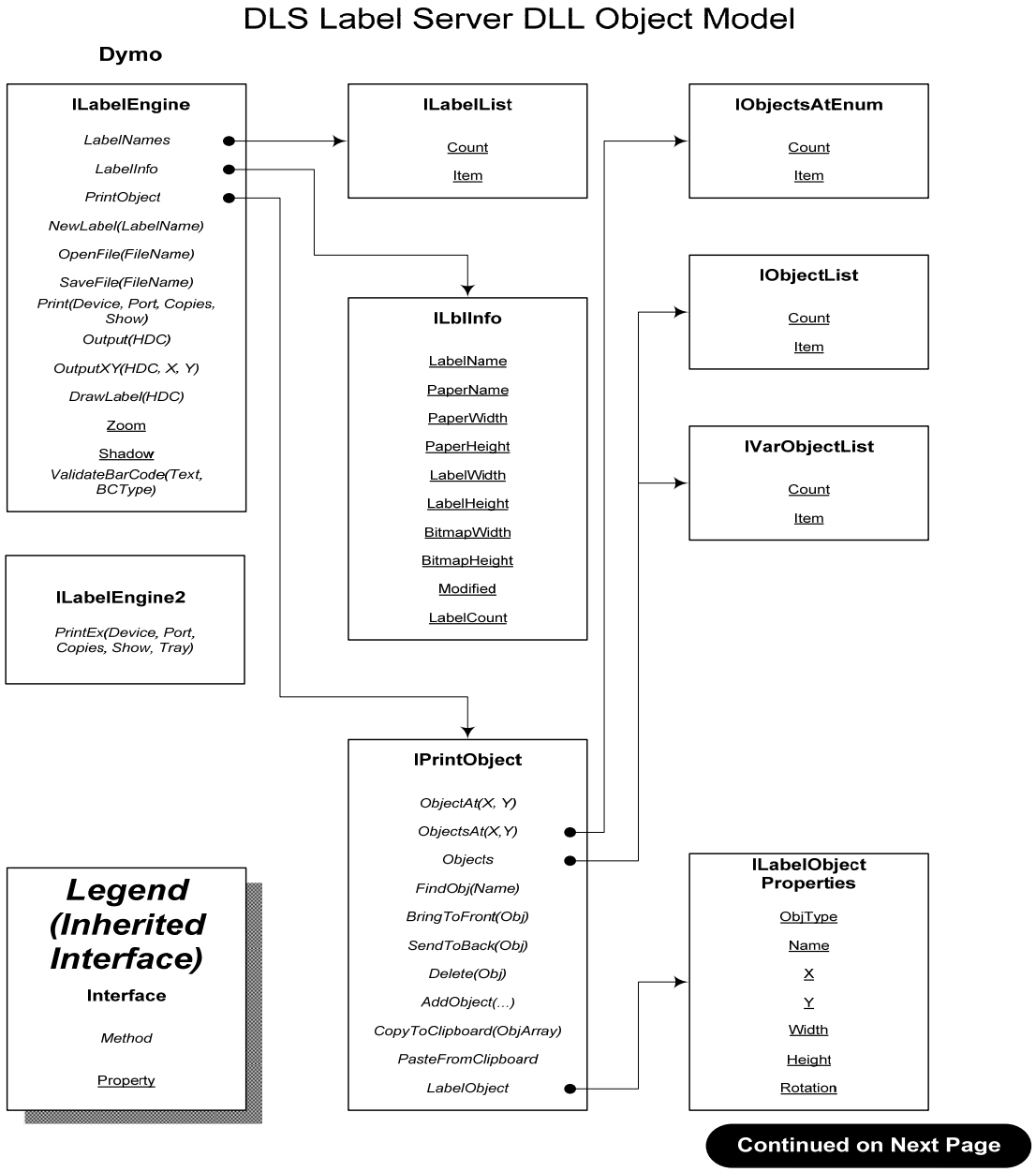

Figure 2

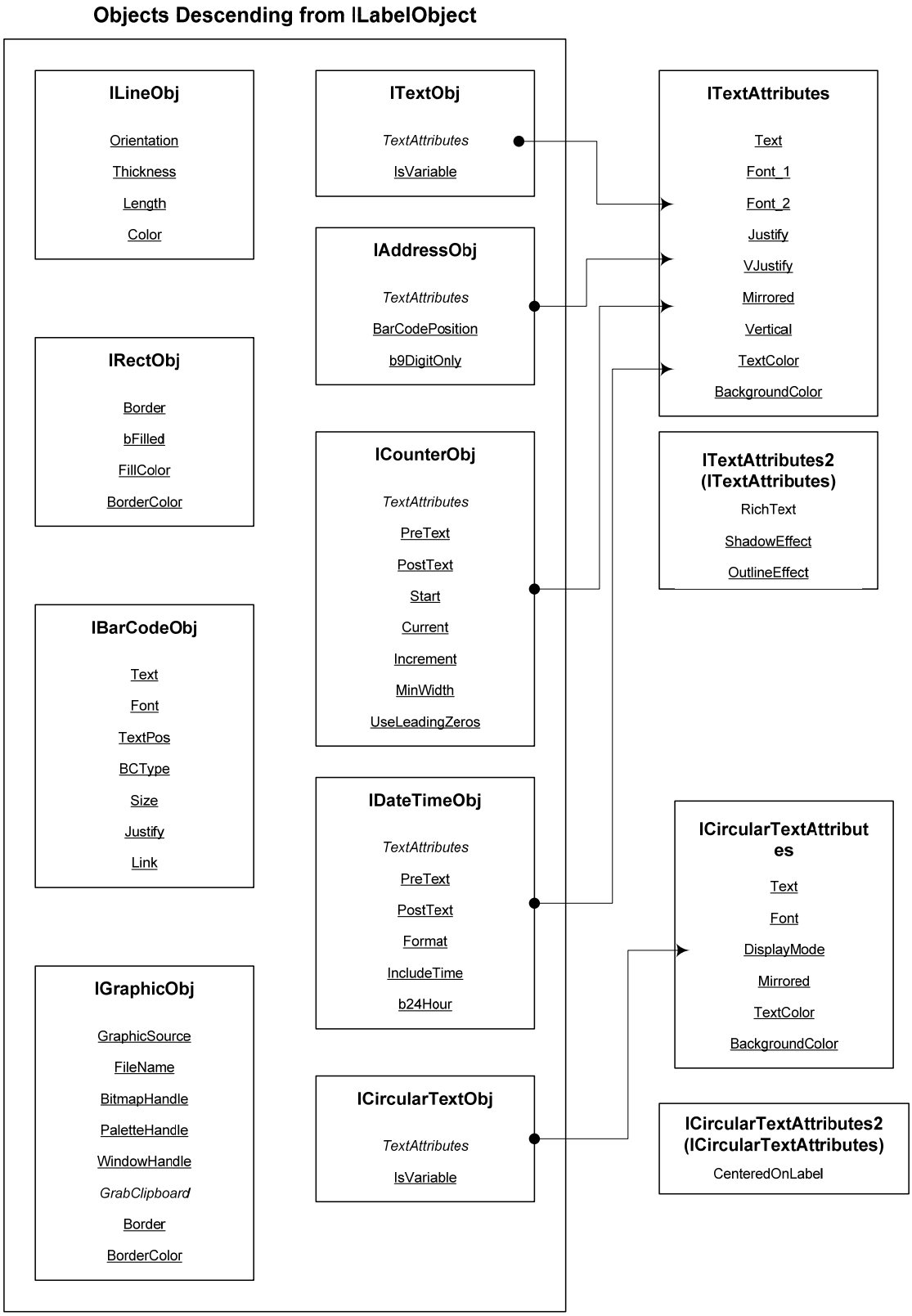
Objects Descending from ILabelObject
ILineObj
Orientation
Thickness
Length
Color
IRectObj
Border
bFilled
FillColor
BorderColor
IBarCodeObj
Text
Font
TextPos
BCType
Size
Justify
Link
IGraphicObj
GraphicSource
FileName
BitmapHandle
PaletteHandle
WindowHandle
GrabClipboard
Border
BorderColor
ITextObj
TextAttributes
IsVariable
IAddressObj
TextAttributes
BarCodePosition
b9DigitOnly
ICounterObj
TextAttributes
PreText
PostText
Start
Current
Increment
MinWidth
UseLeadingZeros
IDateTimeObj
TextAttributes
PreText
PostText
Format
IncludeTime
b24Hour
ICircularTextObj
TextAttributes
IsVariable
ITextAttributes
Text
Font_1
Font_2
Justify
VJustify
Mirrored
Vertical
TextColor
BackgroundColor
ITextAttributes2
(ITextAttributes)
RichText
ShadowEffect
OutlineEffect
ICircularTextAttributes
Text
Font
DisplayMode
Mirrored
TextColor
BackgroundColor
ICircularTextAttributes2
(ICircularTextAttributes)
CenteredOnLabel

Figure 3

The descriptions of the COM Interfaces provided by the label engine DLL are as follows:

ILabelEngine

This is the main interface. The methods and properties supported by this interface apply to entire label designs, and include such things as Open, Save, Print, and so on.

This interface is also used to return the ILabelList interface for the collection object, the ILblInfo interface with label specific information, and the IPrintObject interface for label object level manipulations.

Properties and Methods:

`Zoom` — Read/Write Integer. Used for setting the ZOOM level percentage used when rendering a label on screen using the DrawLabel method. The value set must be between 20 and 400, inclusive.

`Shadow` — Read/Write Boolean. Used for turning the label shadows on and off when drawing a label on-screen. If the value is TRUE, then shadows are added. After this value is written, the bitmap used for rendering the label is resized, so the LabelInfo.BitmapWidth and BitmapHeight values should be reread.

`NewLabel(LabelType: String)`

Creates a new label design. It takes a single parameter, LabelType, which must correspond to one of the strings returned through the LabelList collection object.

`OpenFile(FileName: String)`

Reads a pre-existing label file. FileName is the name (with optional drive and path information) of the file to be read. If no drive or path is given, the current directory is used. Returns TRUE on success, FALSE otherwise.

`SaveFile(FileName: String)`

Saves the current label. Filename is the name (with optional drive and path information) of the file to be read. If no drive or path is given, the current directory is used. Returns TRUE on success, FALSE otherwise.

`PrintLabel(DeviceName: String, Port: String, Quantity: Integer, bShowDialog: Boolean)`

Prints the current label. DeviceName is the name of the device (such as “DYMO LabelWriter Turbo”). Port is the port on which the printer is connected. If Port is File:, then output is directed to a file. Quantity is the number of copies to be printed, and bShowDialog is a boolean used to display (TRUE) or hide (FALSE) the print progress dialog. Returns TRUE on success, FALSE otherwise.

`Output(DC: Integer)` Prints a label to a selected device context (DC). Returns TRUE on success, FALSE otherwise.

`OutputXY(X,Y,DC: Integer)`

Prints the current label on the passed device context, offset by X and Y TWIPS. This is especially useful for “rubber stamping” the same label multiple times on a sheet of labels. To do this, retrieve the DC of the printer, and call BeginDoc. Then, call this method once for each label on a sheet, passing the X and Y coordinates for each of the labels. When done, call EndDoc. Returns TRUE on success, FALSE otherwise.

`DrawLabel(DC: Integer)`

Draws the label on-screen with the current shadow and zoom properties. DC is the device context of the window in which to draw the label.

`LabelNames` Returns an interface to the ILabelList collection object used to retrieve the names of all labels defined in the DEF file. (See Figure 2 on page 27.)

`LabelInfo` Returns an interface to the ILblInfo object. (See Figure 2 on page 27.)

`PrintObject` Returns an interface to the IPrintObject object. (See Figure 2 on page 27.)

ILabelEngine2

This interface was added to support printing to the LabelWriter Twin Turbo printer. It provides a new print function that accepts a paper tray selection.

Properties and Methods:

`PrintLabelEx(DeviceName: String, Port: String, Quantity: Integer, bShowDialog: Boolean, Tray: Integer)`

Prints the current label. DeviceName is the name of the device (such as “DYMO LabelWriter Turbo”). Port is the port on which the printer is connected. If Port is File:, then output is directed to a file. Quantity is the number of copies to be printed, and bShowDialog is a boolean used to display (TRUE) or hide (FALSE) the print progress dialog box. Returns TRUE on success, FALSE otherwise.

Possible Values for the Tray parameter are:

0 = Left Roll
1 = Right Roll
2 = Auto Switch - The printer begins printing from the last printed roll and automatically switches to the second roll when the first roll runs out of paper. The printer continues to toggle back and forth between rolls as long as the user refills the empty rolls. This mode is useful when printing a large number of labels.

ILabelList

This interface to the collection object is used to obtain a list of all supported labels. This list of labels is defined in the LABELS.DEF file. The interface uses the IEnumVariant interface to support the Count and Item properties.

Properties and Methods:

`Count` Read Only. Returns the number of label definitions available.

`Item(Index: Integer)`

Read Only. Returns the name of the given label definition.

ILblInfo

This interface has properties that can be used to obtain information about a given label, including its size, name, paper type, and so on. All properties are read only.

Properties and Methods:

`LabelName` Name of the label (from the DEF file). For example, “Address Label (30252).”

`PaperName` Name of the paper that is selected when this label is being printed.

`Paper Width`	Width of the paper, in TWIPS.
`PaperHeight`	Height of the paper, in TWIPS.
`BitmapWidth`	Width of the bitmap used to render the label on-screen in PIXELS.
`BitmapHeight`	Height of the bitmap used to render the label on-screen, in PIXELS.
`Modified`	TRUE if the label has been modified by the user, else FALSE.

IPrintObject

This interface is used for obtaining information about objects on a label. Many of the properties and methods take an object ID. The object ID can be obtained by the ObjectAt, ObjectsAt, FindObj, or AddObject methods.

Properties and Methods:

`ObjectsAt(x,y: Integer)`

Returns an interface to the IObjectsAtEnum collection object. (*See below*). Used to get the IDs of all objects that include a given point on the label. For example, when the user is trying to select a particular object. X and Y are expressed as offsets from the upper left corner of the label in TWIPS.

`Objects: IObjectList`

Returns an interface to the IObjectList collection object. (*See below*). Used to get the IDs of all objects on a label.

`LabelObject(Obj: Integer)`

Return an ILabelObject interface of an object on the label with the ID of *Obj*. (*See below*).

`FindObj(Name: String)`

Returns the ID of the object with the name *Name*. If the object is not found, returns 0.

`ObjectAt(X,Y: Integer)`

Returns the ID of the top-most object on the label at point X, Y (in TWIPS). If no object is at the point, returns a 0. Not to be confused with ObjectsAt(x,y).

`BringToFront(Obj: Integer)`

Moves the object with the given ID to the top, or in the foreground, of all other objects on the label.

`Delete(Obj: Integer)`

Deletes the object with the ID of *Obj*.

`SendToBack(Obj: Integer)`

Moves the object with the given ID to the back, or in the background, of all other objects on the label.

`AddObject(ObjType: Integer, Name: String; X,Y, Width, Height: Integer; Rotation: Integer`

Creates a new object on the label. ObjType is an integer that specifies the type of object to be created, where:
0 - TEXT
1 - ADDRESS
2 - GRAPHIC
3 - RECTANGLE
4 - LINE
5 - BARCODE
6 - COUNTER
7 - DATE_TIME
8 - CIRCULAR_TEXT

Name is the descriptive name to give the object (such as Return Address, Logo, or Product ID).

X, Y, Width, and Height are the location (X,Y) of the upper left corner of the object and the dimensions of the object, all expressed in TWIPS.

Rotation is the rotation to apply to the object and must be 0, 90, 180, or 270. This value is ignored when adding line or rectangle objects.

After adding an object, the LabelObject method can be used to provide the object-specific settings for the newly created object.

This function returns the ID for the new object, or 0 if the object could not be added.

IObjectsAtEnum

This collection object is used to obtain the IDs of all objects on a label that contain the point X,Y in Twips. It uses the IEnumVariant interface to support the Count and Item properties.

Properties and Methods:

`Count` Read Only. Returns the number of objects that contain the point.

`Item (Index: Integer)`

Read Only. Returns the ID of the object.

IObjectList

This collection Object is used to obtain the IDs of every object on a label. It uses the IEnumVariant interface to support the Count and Item properties.

Properties and Methods:

`Count` Read Only. Returns the number of objects that are on the label.

`Item (Index: Integer)`

Read Only. Returns the ID of the object.

IVarObjectList

This collection Object is used to obtain the IDs of every object on a label that can be pasted into. It uses the IEnumVariant interface to support the Count and Item properties.

Properties and Methods

`Count` Read Only. Returns the number of variable objects on the label.

`Item (Index: Integer)`

Read Only. Returns the ID of the object.

ILabelObject

This interface is used to obtain specific information about an object on a label, including its name, size, rotation, type, and more. It is supported by all object-specific interfaces, and can be obtained through a QueryInterface call on ITextObj, IAddressObj, IGraphicObj, IDateTimeObj, IRectObj, ILineObj, and ICounterObj interfaces. Likewise, after checking the type property of a ILabelObject, you can use QueryInterface to obtain the actual object interface.

Properties and Methods:

`ObjType` Read Only. Returns the type of object the LabelObject corresponds to. Possible values are:
0 – TEXT
1 – ADDRESS
2 – GRAPHIC
3 – RECTANGLE
4 – LINE
5 – BARCODE
6 – COUNTER
7 – DATE_TIME
8 – CIRCULAR_TEXT

`Name` Read/Write. Descriptive name of the object.

`X`	Read/Write. Sets the left edge of the object relative to the paper in TWIPS.
`Y`	Read/Write. Sets the top edge of the object relative to the paper in TWIPS.
`Width`	Read/Write. Sets the width of the object in TWIPS
`Height`	Read/Write. Sets the height of the object in TWIPS.
`Rotation`	Read/Write. Sets the rotation of the object in degrees. Must be 0, 90, 180, or 270. Ignored by Line and Rectangle objects.

ITextAttributes

This interface is available from the Text, Address, DateTime and Counter objects' interface. It is used to manipulate the text, fonts, justification, and other rendering attributes for the objects.

Properties and Methods:

`Text`	Read/Write. String displayed by Text and Address objects. Ignored by the Date/Time and Counter objects.
`Font_1`	Read/Write. Font name, style, and size used for the first line of text in an object. Fonts are represented by strings in the following form: <Font Name>, <Size>, <Style(s)> Example: Times New Roman, 12, Bold Styles can be any combination of Bold, Italic, Underline, and Strikeout
`Font_2`	Read/Write. Used only by the Text and Address objects, provides the font for subsequent lines of text. Ignored by the Date/Time and Counter objects.
`Justify`	Read/Write. Provides access to the justification setting for the text, where: 0 = Left Justify 1 = Center Justify 2 = Right Justify 3 = Center Block
`VJustify`	Read/Write. Provides access to vertical justification. It can be any of:

	0 = Justify to top of bounds 1 = Center between top and bottom 2 = Justify against bottom of bounds
`Mirrored`	Provides access to mirrored text setting for the object, where: FALSE = Print Normally TRUE = Print text mirrored across Horizontal axis (or vertical axis if rotated 90 or 270 degrees).
`Vertical`	Provides access to the Vertical Text setting of the object, where: FALSE = Print Normally TRUE = Print each letter on a line by itself. When TRUE, only the first line of the text is printed. All subsequent text is ignored.
`TextColor`	Read/Write. Provides access to the color of the text
`BackgroundColor`	Read/Write. Provides access to the background color of the object

ITextAttributes2

This interface inherits directly from the ITextAttributes interface. It adds RichText support and additional font rendering effects.

Properties and Methods:

`RichText`	Read/Write. RichText formatted string displayed by Text and Address objects. Ignored by the Date/Time and Counter objects.
`ShadowEffect`	Read/Write. Provides access to the shadow effect setting. FALSE = Text is rendered normally. TRUE = Text is rendered with a shadow behind it.
`OutlineEffect`	Read/Write. Provides access to the outline effect setting. FALSE = Text is rendered normally. TRUE = Only the outline of the text is rendered.

ICircularTextAttributes

This interface is available from the CircularText object's interface. It is used to manipulate the text, fonts, background color, and other rendering attributes for the objects.

Properties and Methods:

`Text`	Read/Write. String displayed by Text and Address objects. Ignored by the Date/Time and Counter objects.
`Font`	Read/Write. Font name, style, and size used for the first line of text in an object. Fonts are represented by strings in the following form: <Font Name>, <Size>, <Style(s)> Example: "Times New Roman, 12, Bold" Styles can be any combination of Bold, Italic, Underline, and Strikeout
`DisplayMode`	Read/Write. Provides access to the justification setting for the text, where: 0 – CircularTextAtTop = Text centered at the top of circle. 1 – CircularTextAtBottom = Text centered at the bottom of circle. 2 – ArcTextAtTop = Text centered at the top of an arc segment. 3 – ArcTextAtBottom. = Text centered at the bottom of an arc segment.
`Mirrored`	Provides access to mirrored text setting for the object, where: FALSE = Print Normally TRUE = Print text mirrored across horizontal axis (or vertical axis if rotated 90 or 270 degrees).
`TextColor`	Read/Write. Provides access to the color of the text.
`BackgroundColor`	Read/Write. Provides access to the background color of the object

ICircularTextAttributes22

This interface is inherited directly from the ITextAttributes interface. It adds RichText support and additional font rendering effects.

Properties and Methods:

`CenteredOnLabel`	Provides access to the centered on label setting for the object, where: FALSE = Positioned normally. TRUE = Always positioned at the center of a label. Position is adjusted automatically if the object size is changed.

ITextObj

This is the interface used to manipulate Text objects. The ITextObj interface must be obtained through a QueryInterface on a ILabelObject interface with the Type property = Text.

Properties and Methods:

`TextAttributes`	Returns an ITextAttributes Interface.
`IsVariable`	Read/Write boolean. Used to tag an object as one that can be pasted into. If True, then the object can be pasted into programmatically. If False, the object can only be changed by direct editing.

IAddressObj

This is the interface used to manipulate Address objects. The IAddressObj interface must be obtained through a QueryInterface on an ILabelObject interface with the Type property = Address.

Properties and Methods:

`TextAttributes`	Returns an ITextAttributes interface.
`BarCodePosition`	Read/Write. Used to set the position for the POSTNET barcode, where: 0 = Suppress POSTNET printing for this object. 1 = Print POSTNET above the address. 2 = Print POSTNET below the address.
`b9DigitOnly`	Read/Write. If this property is TRUE, then POSTNET barcodes are only printed for addresses with full 9-digit (ZIP+4) codes. If FALSE, then 5 and 9-digit POSTNET barcodes are printed. When printing 9-digit ZIP codes, the full 11-digit delivery point barcode (DPBC) is printed.

IGraphicObj

This is the interface used to manipulate Graphic objects. The IGraphicObj interface must be obtained through a QueryInterface on an ILabelObject interface with the Type property = Graphic.

Properties and Methods:

`FileName`	Read/Write. Provides the name of the bitmap file to display.
`BitmapHandle`	Read/Write. Provides the handle of the bitmap to be displayed by the object.
`PaletteHandle`	Read/Write. Provides the handle of the palette to be used in association with the bitmap handle.
`WindowHandle`	Write Only. Provides the handle of a window to be captured for display by the object.
`Border`	Sets the type of border to draw around the image, where: 0 = No Border 1 = Thin Border 2 = Thick Border
`BorderColor`	Read/Write. Provides access to the color of the object's border.
`GraphicSource`	Read/Write. Specifies the type of bitmap being passed to, or returned from the object. Possible values include: 0 = Source image is a file on disk. FileName has the full path to the file 1 = Source image is a bitmap or metafile whose handle is in the Picture field. If Picture corresponds to a bitmap (GetObjectType(Picture) = OBJ_BITMAP) then Palette represents the palette of the bitmap. If the object type is a metafile, then Palette is undefined. 2 = Source image is to be captured from the window whose handle is contained in Window. (Valid only on Setting the attributes for the object. Once captured, the image information is returned as a bitmap handle in the Picture field.) 3 = Source image is in clipboard. Valid only on Setting the attributes for the object. Once captured, the image information is returned as a bitmap (or metafile) handle in the Picture field.

`GrabClipboard`	Loads an image from the clipboard.

ILineObj

This is the interface used to manipulate Line objects. The ILineObj interface must be obtained through a QueryInterface call on a ILabelObject interface with the Type property = Line.

Properties and Methods:

`Orientation`	Read/Write. Provides the orientation of the line, where: 0 = Horizontal Line 1 = Vertical Line
`Thickness`	Read/Write. Provides the thickness of the line, where 0 = No Line – 0 TWIPS 1 = Thin Line – 15 TWIPS 2 = Medium-Thin – 30 TWIPS 3 = Medium Line – 45 TWIPS 4 = Medium-Thick – 80 TWIPS 5 = Thick Line – 115 TWIPS
`LineColor`	Read/Write. Provides access to the color of the line

IRectObj

This is the interface used to manipulate Rectangle objects. The IRectObj interface must be obtained through a Query Interface on an ILabelObject interface with the Type property = Rectangle.

Properties and Methods:

`Border`	Sets the style of border to draw around the image, where: 0 = No Border 1 = Thin Border 2 = Thick Border
`bFilled`	Read/Write. If TRUE, then the rectangle is filled by the color specified by the FillColor property.
`FillColor`	Read/Write. If the bFilled property = TRUE, then this specifies the color to fill the rectangle, otherwise ignored.
`BorderColor`	Read/Write. Provides access to the color of the border.

IBarCodeObj

This is the interface used to manipulate Barcode objects. The IBarCodeObj interface must be obtained through a Query interface on a ILabelObject interface with the Type property = BarCode.

Properties and Methods:

`Text`	Read/Write. Provides the data to be formatted. This string can be up to 255 characters in length.
`Font`	Read/Write. This string represents the font to be used for the human-readable text. For format information, see the Font_1 description for the ITextAttributes interface.
`TextPos`	Read/Write. The position where the human-readable text is to be printed. Possible values include: 0 = No text printed 1 = Above the barcode 2 = Below the barcode
`BCType`	Read/Write. Provides the type of barcode to be printed. Supported types include: 0 = Code 39 (Code 3 of 9) 1 = Code 39 w/Mod 43 Checksum 2 = Code 128 Auto 3 = Code 128A 4 = Code 128B 5 = Code 128C 6 = Code 2 of 5 7 = UPC A 8 = UPC E 9 = EAN 8 10 = EAN 13 11 = Codabar 12 = POSTNET 13 = Code 39 Library Version L – R Checksum 14 = Code 39 Library Version R – L Checksum 15 = Codabar Library Version L – R Checksum 16 = Codabar Library Version R – L Checksum 17 = ITF-14 18 = EAN-128 19 = PLANET
`Size`	Read/Write. Provides the size of the barcode to be printed. The library supports three sizes as follows:

	0 = Small 1 = Medium 2 = Large
`Justify`	Read/Write. Provides the horizontal justification of the barcode within its object bounds. It can one of: 0 = Left Justify 1 = Center Justify 2 = Right Justify
`Link`	If zero, then the data to be barcoded is taken from the Text field of the Barcode object. If non-zero, then the data to be barcoded is taken from the object with an ID that corresponds to the Link value. If non-zero, then the Link value must be the ID of a Text, Address, or Counter object.

ICounterObj

This is the interface used to manipulate Counter objects. The ICounterObj interface must be obtained through a Query Interface on an ILabelObject interface with the Type property = Counter.

Properties and Methods:

`TextAttributes`	Returns an ITextAttributes interface.
`PreText`	Read/Write. Provides the text (if any) to appear *before* the counter value. This must be no more than 31 characters in length.
`PostText`	Read/Write. Provides the text (if any) to appear *after* the counter value. This must be no more than 31 characters in length.
`Start`	Read/Write integer. Provides the value from which the counter starts counting.
`Current`	Read/Write. Current value of the counter. It is incremented each time the label is printed.
`Width`	Read/Write. Specifies the minimum width to format the counter.
`Increment`	Read/Write integer. Specifies the amount by which to increment the Current value after each label is printed. To count down, this value should be negative.

`UseLeadingZeros` Read/Write. If TRUE, then the value is printed with leading zeros added to pad the width to Width. Otherwise, no padding is used.

IDateTimeObj

This is the interface used to manipulate DateTime objects. The IDateTimeObj interface must be obtained through a Query Interface on an ILabelObject interface with the Type property = DateTime.

Properties and Methods:

`TextAttributes` Returns an ITextAttributes interface.

`PreText` Read/Write. Provides the text (if any) to appear *before* the date/time value. This must be no more than 31 characters in length.

`PostText` Read/Write. Provides the text (if any) to appear *after* the date/time value. This must be no more than 31 characters in length.

`Format` Read/Write. Provides the format to be used for the date and time. Available choices include US and international standards. Possible values for Format include:

0 = Blank
1 = Friday, February 6, 1998
2 = Friday, 6 February, 1998
3 = February 6, 1998
4 = 6 February, 1998
5 = 2/6/1998
6 = 6/2/1998
7 = 2/6/98
8 = 6/2/98
9 = 2.6.98
10 = 6.2.98
11 = 1998-02-06
12 = 1998-06-02
13 = 6-Feb-98
14 = Feb 6, 1998

`IncludeTime` Read/Write. If TRUE, the time is added after the date. If False, then only the date is printed.

`b24Hour` Read/Write. If TRUE, then the time is printed as 24-hour time (0-23), otherwise, it is printed as 12 hour (1-12) time.

DLL Calls

Introduction

The LABELS.DLL file provides the DLS interface for using DLL calls. This label engine DLL can be used from any language that supports Windows DLL calls. Like the low-level COM interface, the DLS interface using DLL calls is far more complex than the DLS interface using high-level automation and DDE, and is targeted at those needing to perform complex label printing tasks.

Calling Conventions

All DLL calls use the standard windows calling conventions (_stdcall). Parameters are passed from right to left, and the called function is responsible for cleaning up the stack.

All functions that use string parameters use C-Style, null-terminated strings.

DLL Data Structures

Structures are given in both C and Pascal formats. These data structure definitions, as well as function prototypes and other useful type definitions, are defined in the *lbltypes.h* and *lbltypes.pas* files that are included as part of the SDK.

TLabelInfo

The TLabelInfo structure is used to get information about the current label.

C definition:

```
typedef struct tagLABELINFO {
     char LabelName[64];
     char PaperName[64];
     POINT PaperSize;
     POINT BitmapSize;
     int LabelCount;
     POINT LabelSize;
} TLABELINFO;
typedef TLABELINFO * PLABELINFO;
```

Pascal definition:

```
TLABELINFO = Record
     LabelName : Array[0..63] of char;
     PaperName : Array[0..63] of char;
     PaperSize : TPoint;
     BitmapSize : TPoint;
     LabelCount : Integer;
     LabelSize : TPoint;
End;
PLABELINFO = ^TLabelInfo;
```

Where:

LabelName	Null-terminated string giving the name of the label (for example, ‘Address Label,’ or ‘2-up Address Label’)
PaperName	Null-terminated string giving the name of the paper stock for which this label is designed (for example, ‘Address’). These names correspond to names of paper known by the printer driver.
PaperSize	Gives the X and Y extends of the paper in TWIPS.
BitmapSize	Gives the size of the bitmap required to hold the drawn label at the current Zoom level. The dimensions given are in pixels.
LabelCount	Contains the number of labels per page. Normally 1.
LabelSize	Contains the physical dimensions of the label in TWIPS. PaperSize provides the overall dimensions, but when more than one label per sheet, the label size can’t be determined. This entry provides the actual size of each label.

TObjectID

Object ID is a generic 32-bit handle. Internally, it points to the specific object. It is used when whenever a specific object is being referenced.

C definition:

```
typedef int TObjectID;
```

Pascal definition:

```
TObjectID = TObject;
```

TObjectInfo

This structure returns information about a specific object.

C definition:

```
typedef struct tagObjectInfo {
     TObjectID ObjID;
     Int ObjType;
     char ObjName[64];
     RECT Size;
     int Rotate;
} TOBJECTINFO;
typedef TOBJECTINFO * POBJECTINFO;
```

Pascal definition:

```
TOBJECTINFO = Record
     ObjID : TObjectID;
     ObjType : Integer;
     ObjName : Array[0..63] of char;
     Size : TRect;
```

```
        Rotate : Integer;
    End;
    POBJECTINFO = ^TOBJECTINFO;
```

Where:

ObjID — Handle to the specific object. This can be passed to functions that manipulate objects, to modify, delete, move, or make other modifications.

ObjType — Identifier that corresponds to the type of object. This Read Only field can be one of the following:

Object Type	Identifier
Text	0
Address	1
Graphic	2
Rectangle	3
Line	4
Bar Code	5
Counter	6
Date/Time	7

ObjName — Null-terminated string giving the name of the object (for example, ‘Return Address,’ or ‘Part Number’).

Size — A RECT that provides the X,Y location of the upper left corner of the object, along with its width and height. All dimensions are in TWIPS.

Rotate — One of ROTATE_NONE, ROTATE_90, ROTATE_180, or ROTATE_270. (These are defined as integers of 0, 90, 180, and 270).

TTextBlockAttributes

This structure is used for setting or retrieving information about the text that is displayed by Text and Address objects. With it, you can determine the fonts, text, justification, and other settings:

C definition:

```
    typedef struct tagTTextBlockAttributes {
        char * Text;
        char Font1[64];
        char Font2[64];
        int Justify;
```

```
        int VertJustify;
        BOOL bMirrored;
        BOOL bVerticalText;
        COLORREF TextColor;
        COLORREF BackgroundColor;
        BOOL IsRichText;
        Int  Effects;
    } TTextBlockAttributes;
    typedef TTextBlockAttributes * PTextBlockAttributes;
```

Pascal definition:

```
    TTextBlockAttributes = Record
        Text: PCHAR;
        Font1: Array[0..63] of char;
        Font2: Array[0..63] of char;
        Justify: Integer;
        VertJustify: Integer;
        Mirrored: BOOL;
        VerticalText: BOOL;
        TextColor: COLORREF;
        BackgroundColor: COLORREF;
        IsRichText: BOOL;
        Effects: Integer;
    End;
    PTextBlockAttributes = ^TTextBlockAttributes;
```

Where:

Text	Points to the text to be displayed. This text is null terminated, and has CR/LFs as line terminators.
Font1, Font2	Defines the font used for the first line of text (Font1) and subsequent lines of text (Font2). Fonts are represented by null terminated strings in the following form: <Font Name>, <Size>, <Style(s)> Example: “Times New Roman, 12, Bold” Styles are any combination of Bold, Italic, Underline, and Strikeout.
Justify	Sets the horizontal justification and controls the state of shrink-to-fit. It can be: 0 = Left Justify 1 = Center Justify

	2 = Right Justify 3 = Center Block The high-bit of this field is used to enable or disable shrink-to-fit for the text. If the high-bit is non-zero, then shrink-to-fit is disabled, otherwise, it is enabled. To disable shrink-to-fit, use the OR operator with value 0x80000000.
VertJustify	Sets the vertical justification. It can be: 0 = Justify to top of bounds 1 = Center between top and bottom 2 = Justify against bottom of bounds
Mirrored	Used to force text to print mirrored across axis. FALSE = Print Normally TRUE = Print text mirrored across Horizontal axis (or vertical if rotated 90 or 270 degrees).
VerticalText	Used to print text vertically. FALSE = Print Normally TRUE = Print each letter on a line by itself.
TextColor	Provides the color to be used for the text characters.
BackgroundColor	Provides the color to be used to fill in the background of the object.
IsRichText	Used to identify if the text stored in the text field is RichText formatted.
Effects	Indicates the effects used when render text. Logical-OR the constants to combine rendering effects. 0x00000001 = Shadow Effect 0x00000002 = Ouline Effect

TCircularTextBlockAttributes

This structure is used for setting or retrieving information about the text that is displayed by Text and Address objects. With it, you can determine the fonts, text, justification, and other settings.

C definition:

```
typedef struct tagTCircularTextBlockAttributes {
     char * Text;
     char Font[64];
     BOOL bMirrored;
     BYTE DisplayMode;
     COLORREF TextColor;
     COLORREF BackgroundColor;
     BOOL CenteredOnLabel;
} TCircularTextBlockAttributes;
typedef TCircularTextBlockAttributes *
PCircularTextBlockAttributes;
```

Pascal definition:

```
TDisplayMode = (dmCircleTextAtTop, dmCircleTextAtBottom,
     dmArcTextAtTop, dmArcTextAtBottom);
TCircularTextBlockAttributes = record
     Text: PCHAR;
     Font: array[0..63] of char;
     bMirrored: BOOL;
     DisplayMode: TDisplayMode;
     TextColor: COLORREF;
     BackGroundColor: COLORREF;
     bCenteredOnLabel: BOOL;
end;
PCircularTextBlockAttributes =
^TCircularTextBlockAttributes;
```

Where:

Text	Points to the text to be displayed. This text is null terminated, and all CR/LFs are replaced with a space character.
Font	Defines the font used for the text. Fonts are represented by null terminated strings in the following form: <Font Name>, <Size>, <Style(s)> Example: “Times New Roman, 12, Bold” Styles are any combination of Bold, Italic, Underline, and Strikeout.

Mirrored	Used to force text to print mirrored across axis. FALSE = Print Normally TRUE = Print text mirrored across Horizontal axis (or vertical if rotated 90 or 270 degrees).
DisplayMode	Sets the circular text display mode as one of the following: 0 = Text is centered at the top of the circle 1 = Text is centered at the bottom of the circle 2 = Text is centered at the top of an arc segment 3 = Text is centered at the bottom of an arc segment
TextColor	Provides the color for the text characters.
BackgroundColor	Provides the color to fill in the background of the object.
bCenteredOnLabel	Determines the position of the circular text object. FALSE = Positioned normally TRUE = Always positioned at the center of a label. The position is automatically adjusted if the size of the object is changed.

TAddressAttributes

This structure is used for setting or retrieving information about an Address object:

C definition:

```
typedef struct tagTAddressAttributes {
     PTextBlockAttributes TextInfo;
     int BarCodePos;
     BOOL b9DigitOnly;
     BOOL bFixedAddr;
     BOOL bFormatted;
} TAddressAttributes;
typedef TAddressAttributes * PAddressAttributes;
```

Pascal definition:

```
TAddressAttributes = Record
     TextInfo: PTextBlockAttributes;
     BarCodePos: Integer;
     b9DigitOnly: BOOL;
     bFixedAddr: BOOL;
     bFormatted: BOOL;
```

```
End;
PAddressAttributes = ^TAddressAttributes;
```

Where:

TextInfo	Points to a TextBlockAttributes structure from which text specific attributes can be read or written. If this entry is null, then the text remains unchanged.
BarCodePos	Sets the position of the POSTNET barcode as one of the following: 0 = Suppress POSTNET printing for this object 1 = Print POSTNET above the address 2 = Print POSTNET below the address
b9DigitOnly	If TRUE, then POSTNET barcodes are only printed for addresses with full 9-digit (ZIP+4) codes. If False, then 5-digit POSTNET barcodes are printed.
bFixedAddr	Indicates if the address text can be changed. This flag is reserved for the DLS application and should be set to false for your application.
bFormatted	A read-only variable to indicate if the address has a font/text format for each line of the address text. This flag is reserved for the DLS application and should not be set.

TTextAttributes

This structure is used for setting or retrieving information about a Text object:

C definition:

```
typedef struct tagTTextAttributes {
     PTextBlockAttributes TextInfo;
     BOOL bVariable;
} TTextAttributes;
typedef TTextAttributes * PTextAttributes;
```

Pascal definition:

```
TTextAttributes = Record
     TextInfo : PTextBlockAttributes;
     bVariable : BOOL;
End;
PTextAttributes = ^TTextAttributes;
```

Where:

TextInfo	Points to a TextBlockAttributes structure from which text specific attributes can be read or written. If this entry is null, then the text remains unchanged.
bVariable	If TRUE, then this object can be pasted into. Otherwise, the text can only be changed by a Get/Set call.

TGraphicAttributes

This structure is used for setting or retrieving information about a Graphic object:

C definition:

```
typedef struct tagTGraphicAttributes {
     int GraphicSource;
     char FileName[260];
     HANDLE Picture;
     HPALETTE Palette;
     HWND Window;
     int Border;
     COLORREF BorderColor;
} TGraphicAttributes;
typedef TGraphicAttributes * PGraphicAttributes;
```

Pascal definition:

```
TGraphicAttributes = Record
     GraphicSource : Integer;
     FileName : Array[0..259] of char;
     Picture : THANDLE;
     Palette: HPALETTE;
     Window : HWND;
     Border : Integer;
     BorderColor : COLORREF;
End;
PGraphicAttributes = ^TGraphicAttributes;
```

Where:

GraphicSource	Used to specify the type of bitmap being passed to or returned from the object. Possible values include: 0 = Source image is a file on disk. FileName has the full path to the file. 1 = Source image is a bitmap or metafile whose handle is in the Picture field. If Picture corresponds to a bitmap (GetObjectType(Picture) = OBJ_BITMAP), then Palette represents the palette of the bitmap. If the object type is a metafile, then Palette is undefined.

	2 = Source image is to be captured from the window whose handle is contained in Window. (Valid only on Setting the attributes for the object. Once captured, the image information is returned as a bitmap handle in the Picture field.) 3 = Source image is in clipboard. Valid only on Setting the attributes for the object. Once captured, the image information is returned as a bitmap (or metafile) handle in the Picture field.
FileName	When the image source is a file, this field contains the name of the file represented by the graphic object. Supported file types include BMP. WMF. EMF, PCX, PNG, TIF, and JPG.
Picture	When the image is not from a file, contains the handle of the bitmap or metafile to be displayed (on input) or being displayed (on output).
Palette	Contains the handle of the bitmap palette. If 0, then the system palette is used. When returning attribute information, Attribute represents the palette of the bitmap if Picture represents a bitmap handle. If Picture represents a metafile handle, then this is undefined on output.
Window	Valid on input only. This contains the handle of a window whose image is to be captured.
Border	Sets the type of border that is drawn around the graphic. Possible values include: 0 = No Border 1 = Thin Border 2 = Thick Border
BorderColor	Defines the color of the border to be drawn around the graphic. If Border = 0 then this setting is not used.

TLineAttributes

This structure is used for setting or retrieving information about a Line object:

C definition:

```
typedef struct tagTLineAttributes {
     int Length;
     int Orient;
     int Thickness;
     COLORREF LineColor;
} TLineAttributes;
```

```
typedef TLineAttributes * PLineAttributes;
```

Pascal definition:

```
TLineAttributes = Record
     Length : Integer;
     Orient : Integer;
     Thickness : Integer;
     LineColor : COLORREF;
End;
PLineAttributes = ^TLineAttributes;
```

Where:

Length — Represents the length of the line in TWIPS

Orient — Represents the orientation of the line, where:

0 = Horizontal Line
1 = Vertical Line

Thickness — Represents the thickness of the line, where:

Value	Description	Width (in TWIPS)
0	No Line	0
1	Thin line	15
2	Medium-Thin	30
3	Medium Line	45
4	Medium-Thick	80
5	Thick Line	115

LineColor — Provides the color of the line.

TRectAttributes

This structure is used for setting or retrieving information about a Rectangle object:

C definition:

```
typedef struct tagTRectAttributes {
     BOOL bFilled;
     int Border;
     COLORREF BorderColor;
     COLORREF FillColor;
```

```
} TRectAttributes;
typedef TRectAttributes * PRectAttributes;
```

Pascal definition:

```
TRectAttributes = Record
     bFilled : BOOL;
     Border : Integer;
     BorderColor : COLORREF;
     FillColor : COLORREF;
End;
PRectAttributes = ^TRectAttributes;
```

Where:

bFilled	If TRUE, then the rectangle is drawn as a solid, black rectangle. Otherwise, it is drawn as a rectangle with the specified border.
Border	This is used to set the type of border that is drawn around the rectangle. Possible values include: 0 = No Border 1 = Thin Border 2 = Thick Border
BorderColor	Color of the border (if any).
FillColor	Color to use to fill the interior of the rectangle.

TCounterAttributes

This structure is used for setting or retrieving information about a Counter object

C definition:

```
typedef struct tagTCounterAttributes {
     PTextBlockAttributes TextInfo;
     char PreText[32];
     char PostText[32];
     int Start;
     int Current;
     int Width;
     int Increment;
     BOOL bLeadingZeros;
} TCounterAttributes;
typedef TCounterAttributes * PCounterAttributes;
```

Pascal definition:

```
TCounterAttributes = Record
     TextInfo : PTextBlockAttributes;
     PreText : Array[0..31] of char;
```

```
        PostText : Array[0..31] of char;
        Start : Integer;
        Current : Integer;
        Width : Integer;
        Increment : Integer;
        bLeadingZeros : BOOL;
    End;
    PCounterAttributes = ^TCounterAttributes;
```

Where:

TextInfo	Points to a TextBlockAttributes structure from which text specific attributes can be read or written. If this entry is null, then the text remains unchanged.
PreText	Specifies any text to appear *before* the counter value. This is a null terminated string of up to 31 characters in length.
PostText	Specifies any text to appear *after* the counter value. This is a null terminated string of up to 31 characters in length.
Start	Value from which the counter starts counting.
Current	Current value of the counter. It is incremented each time the label is printed.
Width	Specifies the minimum width to format the counter.
Increment	Specifies the amount by which to increment the Current value after each label is printed. To count down, this value should be negative.
bLeadingZeros	If TRUE, then the value is printed with leading zeros added to pad the width out to Width. Otherwise, no padding is used.

TDateTimeAttributes

This structure is used for setting or retrieving information about a Date/Time object:

C definition:

```
    typedef struct tagTDateTimeAttributes {
        PTextBlockAttributes TextInfo;
        char PreText[32];
        char PostText[32];
        int DateFormat;
        BOOL bIncludeTime;
        BOOL b24Hour;
    } TDateTimeAttributes;
    typedef TDateTimeAttributes * PDateTimeAttributes;
```

Pascal definition:

```
TDateTimeAttributes = Record
     TextInfo : PTextBlockAttributes;
     PreText : Array[0..31] of char;
     PostText : Array[0..31] of char;
     DateFormat : Integer;
     bIncludeTime : BOOL;
     b24Hour : BOOL;
End;
PDateTimeAttributes = ^TDateTimeAttributes;
```

Where:

TextInfo — Points to a TextBlockAttributes structure from which text specific attributes can be read or written. If this entry is null, then the text remains unchanged.

PreText — Specifies any text to appear *before* the Date/Time value. This is a null terminated string of up to 31 characters in length.

PostText — Specifies any text to appear *after* the Date/Time value. This is a null terminated string of up to 31 characters in length.

DateFormat — Specifies the format to be used for the date and time. Available choices include US and international standards. Possible values for DateFormat include:

Value	Sample
0	Blank
1	Friday, February 6, 1998
2	Friday, 6 February, 1998
3	February 6, 1998
4	6 February, 1998
5	2/6/1998
6	6/2/1998
7	2/6/98
8	6/2/98
9	2.6.98
10	6.2.98
11	1998-02-06
12	1998-06-02

13	6-Feb-98
14	Feb 6, 1998

bIncludeTime — If TRUE, then the time is added after the date. If False, then only the date is printed.

b24Hour — If TRUE, then the time is printed as 24-hour time (0-23), otherwise, it is printed as 12 hour (1-12) time.

TBarcodeAttributes

This structure is used for setting or retrieving information about a Barcode object:

C definition:

```
typedef struct tagTBarcodeAttributes {
     char Text[256];
     char Font[64];
     int HRTextPos;
     int BCType;
     int BCRatio;
     int Justify;
     TObjectID Link;
} TBarcodeAttributes;
typedef TBarcodeAttributes * PBarcodeAttributes;
```

Pascal definition:

```
TBarcodeAttributes = Record
     Text : Array[0..255] of char;
     Font : Array[0..63] of char;
     HRTextPos : Integer;
     BCType : Integer;
     BCRatio : Integer
     Justify : Integer;
     Link : TObjectID;
End;
PBarcodeAttributes = ^TBarcodeAttributes;
```

Where:

Text — Provides the data to be formatted. This can be up to 255 characters in length, and is passed as a null terminated string.

Font — A null terminated string that represents the font to be used for the human-readable text. For format information, see the Font1 description for the TextBlockAttributes structure.

HRTextPos

The position where the human-readable text is to be printed. Possible values include:

0 = No text printed
1 = Above the bar code
2 = Below the bar code

BCType

Gives the type of barcode to be printed. Supported types include:

Value	Barcode Symbology
0	Code 39 (Code 3 of 9)
1	Code 39 w/Mod 43 Checksum
2	Code 128 Auto
3	Code 128A
4	Code 128B
5	Code 128C
6	Code 2 of 5
7	UPC A
8	UPC E
9	EAN 8
10	EAN 13
11	Codabar
12	POSTNET
13	Code 39 Library Version L – R Checksum
14	Code 39 Library Version R – L Checksum
15	Codabar Library Version L – R Checksum
16	Codabar Library Version R – L Checksum
17	ITF-14
18	EAN-128
19	PLANET

BCRatio

Sets the size of the barcode to be printed. The library supports three sizes as follows:

0 = Small
1 = Medium
2 = Large

Justify	Sets the horizontal justification of the barcode within its object bounds as follows: 0 = Left Justify 1 = Center Justify 2 = Right Justify
Link	If null, then the data represented is from the Text field of the barcode object. If not null, then it must be an object handle for a Text, Address, or Counter object.

The DLL Functions

Following is a list of the DLL function calls. For each function, the prototypes are given in both C and Pascal. In all cases, the C version is listed first and the Pascal version appears second.

```
BOOL WINAPI ReadLabelFile(LPCSTR FileName);
Function ReadLabelFile(FileName : PChar) : BOOL; stdcall;
```

Clears any existing label file from memory and reads a new label definition from the file name.

FileName	A pointer to the filename to be read. This is the full file name, with drive and path information. If no drive or path is included, it tries to open the file in the current directory.
Returns:	True on success, False if file could not be read, or the file was not a valid label file.

```
BOOL WINAPI WriteLabelFile(LPCSTR FileName);
Function WriteLabelFile(FileName : PChar) : BOOL; stdcall;
```

Saves the current label using the passed filename. If the file already exists, it is overwritten.

FileName	A pointer to the filename to be written. This is the full file name, with drive and path information. If no drive or path is included, tries to save the file in the current directory.
Returns:	True on success, False if file could not be written.

```
BOOL WINAPI WriteLabelFileEx(LPCSTR FileName, BOOL SaveAddress);
Function WriteLabelFileEx(FileName: PChar; SaveAddress: BOOL):
                    BOOL; stdcall;
```

Saves the current label using the passed filename. If the file already exists, it is overwritten.

FileName	A pointer to the filename to be written. This is the full file name, with drive and path information. If no drive or path is included, it tries to save the file in the current directory.

SaveAddress	Set to True to save the content of address. Set to False to save address contents as empty strings.
Returns:	True on success, False if file could not be written.

```
BOOL WINAPI PrintLabel(LPCSTR DeviceName, LPCSTR Port, int
                    Quantity, BOOL bShowDialog);
Function PrintLabel(DeviceName : PChar; Port : PChar; Quantity :
                    Integer; ShowDialog : BOOL) : Bool; stdcall;
```

Prints the current label on the given output device.

DeviceName	Specifies the device on which to print. For example, “DYMO LabelWriter 400 Turbo”.
Port	Specifies the Port on which to print. If null, then the port is taken from Windows. If not null, then the output is forced to the passed port. To print to a file, use ‘File:’.
Quantity	Specifies the number of labels to be printed. The maximum valid value is 2,147,483,647.
bShowDialog	If TRUE, then the printer progress dialog is displayed when printing. If FALSE, then no dialog is displayed.
Returns:	True on success, False if file could not be written.

```
BOOL WINAPI PrintLabelEx(LPCSTR DeviceName, LPCSTR Port, int
                    Quantity, BOOL bShowDialog, int PaperTray);
Function PrintLabelEx(DeviceName : PChar; Port : PChar; Quantity
                    : Integer; ShowDialog : BOOL; PaperTray:
                    Integer) : Bool; stdcall;
```

Same as the PrintLabel() function (see above) with an additional “PaperTray” parameter to specify the paper tray selection when printing on a LabelWriter Twin Turbo.

PaperTray	Specifies the Paper Tray selection when printing on a LabelWriter Twin Turbo. Possible values for this parameter are: 0 - Left Roll 1 - Right Roll 2 - Auto Switch - This puts the printer in a mode where it starts to print from the last printed roll and automatically switch over to the other roll when the starting roll runs out of paper. It continues to toggle back and forth between rolls as long as the user refills rolls once they become empty. This mode of printing is useful when the user is printing a large number of labels.

```
BOOL WINAPI Output(HDC hDC);
Function Output(DC: HDC) : BOOL; stdcall;
```

Similar to PrintLabel, but without BeginDoc/EndDoc, and simply prints to the specified device context. This function can be used when trying to mix your output with ours. For instance, if you need to print a label with a MaxiCode barcode, you can use another DLL to print the MaxiCode barcode, and then call this function to print everything else. Because this function is limited to outputting to an existing print job, it uses a device context as its only parameter.

hDC	The device context for the output.
Returns:	True on success, False if file could not be written.

```
BOOL WINAPI OutputXY(int X, int Y, HDC hDC);
Function OutputXY(X,Y : Integer; DC : HDC) : BOOL; stdcall;
```

This is the same as Output(), but provides an offset (in TWIPS) used to move the label origin on the page. Using this function, an application can use the DLL to print on a laser label sheet, by calling it once for every label on the page, passing the X and Y coordinates of the upper left corner of the label being printed.

X	Offset from left edge of paper to starting label position in TWIPS.
Y	Offset from top edge of paper to starting label position in TWIPS.
hDC	The device context for the output.
Returns:	True on success, False if file could not be written.

```
void WINAPI DrawLabel(HDC hDC);
Procedure DrawLabel(DC: HDC); stdcall;
```

Draws the current label design on the passed device context using the current Zoom setting.

hDC	The device context for the output.

```
void WINAPI SetZoom(int Percent);
Procedure SetZoom(Percent : Integer); stdcall;
```

Sets the zooming factor for DrawLabel() commands. Zoom is a scaling factor (in percent) to apply when drawing. If <> 100, the function changes the scale of the drawing to the specified size. The Device Context is pushed and popped around the call to preserve the current DC settings.

Percent	The percentage of actual size to render the label when DrawLabel is next called. This value is ignored if it is less than 20 or greater than 400.

```
void WINAPI SetShadow(BOOL State);
Procedure SetShadow(State : BOOL); stdcall;
```

Turns the display of the shadow on and off. After calling this function, the DLL modifies the Label bitmap, so you must call GetLabelInfo to retrieve the new size of the bitmap.

State	If TRUE, then label is drawn with shadow. If FALSE, then no shadow is drawn.

```
void WINAPI GetLabelInfo(PLABELINFO LabelInfo);
Procedure GetLabelInfo(LabelInfo : PLABELINFO); stdcall;
```

Fills a TLabelInfo structure with information about the current label.

LblInfo	Points to the TLabelInfo structure to be filled.

If LabelInfo is an invalid pointer, this call is ignored.

```
int WINAPI GetLabelNames(char * Buf, int nBytes);
Function GetLabelNames(Buf : PChar; nBytes : Integer) : Integer;
                    stdcall;
```

Fills a buffer with the names of all Label paper names. The buffer is returned as a series of 64 byte blocks of null terminated label names, with the last block being followed by a second null.

Buf	Points to the memory block to be filled with the label names.
nSize	Specifies the size of the buffer (in bytes).
Returns:	If the buffer is too small, the function returns the number of bytes required. Otherwise, it fills the buffer with the names of the labels and returns a 1. Returns 0 if Buf is invalid.

```
void WINAPI NewLabel(LPCSTR Name);
Procedure NewLabel(Name : PChar); stdcall;
```

Clears any existing label design from memory and creates a new label based on the supplied Name.

Name	Points to a null-terminated string specifying a valid label name, as returned in the GetLabelNames call.

```
BOOL WINAPI IsModified(void);
Function IsModified : BOOL; stdcall;
```

Used to determine if a design has been modified without being saved to disk.

Returns:	TRUE if the current label design has been modified but not written to disk, FALSE otherwise.

```
BOOL WINAPI SetModified(BOOL State);
Function SetModified(State : BOOL) : BOOL; stdcall;
```

This function is used to set or clear the current label's "modified" flag.

State	If TRUE, then the label is marked as having been changed. If FALSE, then the label is marked as being unchanged from what was read from disk.
Returns:	Previous state of the modified flag.

```
HANDLE WINAPI AddObject(LPCSTR ObjType, LPCSTR ObjName, RECT
                    Size, int Rotation, void * Attrib);
Function AddObject(ObjType, ObjName : PChar; Size: TRect;
                    Rotation : Integer; Attrib : Pointer) :
                    THandle; stdcall;
```

Adds a new object to the label design.

ObjType	Points to a null-terminated string that gives the type of object to be created. Valid values include: TEXT ADDRESS GRAPHIC RECTANGLE LINE BARCODE COUNTER DATE-TIME CIRCULAR-TEXT
ObjName	Points to a null-terminated string that provides the name to give the object (for example, 'Return Address' or 'Part Number'). The object's name is only provided as a way of differentiating between objects of the same type.
Size	Specifies a Rectangle that provides the left edge (Rect.Left), top (Rect.Top), width (Rect.Right), and height (Rect.Bottom) of the object to be created. All units are in TWIPS and relative to the upper left corner of the label. Rect.Left and Rect.Top have a minimum value of 0. Width and height (Rect.Right and Rect.Bottom) have a minimum of 1 TWIP.
Rotate	Specifies the type of rotation to apply to the object (0, 90, 180, or 270 degrees).

Attributes	Points to an object specific attribute structure (for example, TAddressAttributes, TLineAttributes, TBarCodeAttributes, and so on).
Returns:	A handle (TObjectID) to the new object.

```
int WINAPI EnumLabelObjects(char *Buf, int nBytes);
Function EnumLabelObjects(Buf : PChar; nBytes : Integer) :
                    Integer; stdcall;
```

Fills a buffer with TObjectInfo structures, one for each object on the current label. This is used to get a list of all objects on a label.

Buf	Points to the memory block to be filled in with ObjInfo structures.
nSize	Specifies the size of the buffer (in bytes).
Returns:	If the buffer is too small, the function returns the number of bytes required. Otherwise, it fills the buffer with an array of ObjInfo structures and returns a 1. If the buffer pointer is invalid, it returns 0.

This function should first be called with Buf = nil and nSize = 0. This returns the minimum size of the buffer to be allocated.

```
int WINAPI EnumVariableObjects(char *Buf, int nBytes);
Function EnumVariableObjects (Buf : PChar; nBytes : Integer) :
                    Integer; stdcall;
```

Similar to EnumLabelObjects(), but ignores non-variable objects. Fills a buffer with TObjectInfo structures, one for each variable object on the current label. This is used to get a list of all variable objects on a label. A variable object is defined to be:

- Any Address object
- Text objects with bVariable = TRUE
- Barcode objects with link = nil

Buf	Points to the memory block to be filled in with ObjInfo structures.
nSize	Specifies the size of the buffer (in bytes).
Returns:	If the buffer is too small, the function returns the number of bytes required. Otherwise, it fills the buffer with an array of ObjInfo structures and returns a 1. If the buffer pointer is invalid, it returns 0.

Tthis function should first be called with Buf = nil, and nSize = 0. This returns the minimum size of the buffer to be allocated.

```
TObjectID WINAPI WhichObject(int X, int Y);
Function WhichObject(X,Y : Integer) : TObjectID; stdcall;
```

Reports which object is the top-most object at a given location on the label.

X, Y	Specifies the X and Y coordinates (in TWIPS) for the point being checked.
Returns:	The ID of the top-most object at the given point or NULL, if no object surrounds the point.

```
int WINAPI ObjectsAt(int X, int Y, char * Buf, int nBytes);
Function ObjectsAt(X,Y : Integer; Buf : PChar; nBytes : Integer)
                   : Integer; stdcall;
```

Fills an array of TObjectInfo structures for all objects on the current label that contain a given point.

X, Y	Specifies the X and Y coordinates (in TWIPS) for the point being checked.
Buf	Points to the memory block to be filled in with the TObjectInfo structures.
nBytes	Specifies the size of the buffer (in bytes).
Returns:	If the buffer is too small, the function returns the number of bytes required. Otherwise, it fills the buffer with a series of TObjectInfo structures and returns a 1. Returns 0 if the buffer pointer is invalid, or –1 if no objects exist at the provided X, Y coordinates.

This function should first be called with Buf = nil, and nSize = 0. This returns the minimum size of the buffer to be allocated.

```
int WINAPI ValidateBarCode(char *Text, int BCType)
Function ValidateBarCode(Text : PChar; BCType : Integer) :
                   Integer; stdcall;
```

Used to determine if a text string can be represented in a particular barcode symbology.

Text	The text to be barcoded.
BCType	The symbology to be used. (See description of BarcodeAttributes for allowed values).
Returns:	Returns 0 if the text can be represented by using the given symbology. Possible return codes include:

0	No error

1	Invalid character in text.
2	Text too long to for symbology.
3	Unsupported barcode type.
4	Barcode must contain digits only.
5	Barcode requires an even number of characters.
6	Barcode can not encode characters < ASCII Space.
7	No lower case characters allowed.
8	Barcode can not encode characters > 0x80.
9	First/Last characters not valid for symbology.
10	Text must be 5, 9, or 11 characters long.
11	Library barcodes must be 12 or 13 characters.

```
BOOL WINAPI CopyToClipboard (POBJECTID Buf);
Function CopyToClipboard(Buf : PObjectID) : Boolean; stdcall;
```

Copies a list of objects to the clipboard using the registered clipboard format “LabelObject”.

Buf — A null terminated list of IDs identifying the objects to be copied to the clipboard. The last object ID is followed by a null.

Returns: True if the copy was successful, else False.

```
BOOL WINAPI PasteFromClipboard(void);
Function PasteFromClipboard : Boolean; stdcall;
```

Pastes all objects in the clipboard onto the current label.

Returns: True if the paste was successful. If the clipboard does not have any objects with the registered clipboard format “LabelObject,” returns False.

```
TObjectID WINAPI FindObject(char * Name);
Function FindObject(Name :PChar) : TObjectID; stdcall;
```

Name — Identifies an object to be found.

Returns: The ID of the object with the given name. If no object by the given name is on the label, returns 0.

```
BOOL WINAPI DeleteLabelObject(TObjectID ID);
Function DeleteLabelObject (ID : TObjectID) : BOOL; stdcall;
```

Deletes an object from a label design.

ID: Specifies the ID of the object to be deleted.

Returns: True on success, False if an object with the specified ID could not be found.

```
BOOL WINAPI BringObjectToFront(TObjectID ID);
Function BringObjectToFront(ID : TobjectID) : BOOL; stdcall;
```

Reorganizes the label so that the specified object is the top-most object on the label.

ID: Specifies the ID of the object to be moved.

Returns: True on success, False if an object with the specified ID could not be found.

```
BOOL WINAPI SendObjectToBack(TObjectID ID);
Function SendObjectToBack(ID : TObjectID) : BOOL; stdcall;
```

Reorganizes the label so that the specified object is the bottom-most object on the label.

ID: Specifies the ID of the object to be moved.

Returns: True on success, False if an object with the specified ID could not be found.

```
BOOL WINAPI SetAttributes(TObjectID ID, void * Attrib);
Function SetAttributes(ID : TObjectID; Attrib : Pointer) : BOOL;
                    stdcall;
```

This function is used to set the attributes for the specified object. It is used to change the data (text, barcode data, graphic, and so on), as well as its format (font, justification, text location, date format, and so on).

> *Note: If you are using the DLL data structure definitions, use the SetAttributesEx() function instead. The SetAttributes() function is provided only for backward compatibility reasons (for example, programs written using the old DLL data structure definitions).*

ID: Specifies the ID of the object to be moved.

Attrib: Points to an object-type specific attribute structure from which the object's new settings are taken.

Returns: True on success, False if an object with the specified ID could not be found.

```
int WINAPI GetAttributes(TObjectID ID, void * Attrib);
Function GetAttributes(ID : TObjectID; Attrib : Pointer) :
                    Integer; stdcall;
```

Retrieves the current object attributes for the specified object**:**

> *Note: If you are using the DLL data structure definitions, please use the GetAttributesEx() function instead. The GetAttributes() function is provided only for backward compatibility reasons (i.e. for programs written using the old DLL data structure definitions).*

ID — Specifies the ID of the object to be moved.

Attrib — Points to an object-type specific attribute structure from which the object's new settings are taken.

Returns: If Attrib = NULL, returns the size of the memory block required for the Text in the specified object. Otherwise, returns True on success, False if an object with the specified ID could not be found, or Attrib was invalid.

> *Note: If ID references a Text or Address object, and Attributes is NULL, then the function returns the amount of storage (in bytes) that must be allocated for the Text member pointer in the TTextBlockAttributes portion of the objects attribute structure.*

To retrieve the text from an Address or Text object:

1. Call GetAttributes() passing a null for the Attributes pointer. This returns the amount of memory to allocate to hold the text.
2. Allocate the memory required, and put a pointer to the memory in TextBlockAttributes.Text.
3. Fill in a TTextAttributes (or TAddressAttributes) with a reference to the TTextBlockAttributes structure.
4. Call GetAttributes() with a pointer to the TTextAttributes (or TAddressAttributes) structure to retrieve the information.

```
BOOL WINAPI GetInfo(POBJECTINFO ObjectInfo);
Function GetInfo(ObjectInfo : POBJECTINFO) : BOOL; stdcall;
```

Finds the object with the ObjID passed in the TObjectInfo structure, and fills in the rest of the fields of the TObjectInfo structure with information about the object.

ObjectInfo — Points to a TObjectInfo structure in which the ObjID field has been set to the handle of an object to be queried.

Returns:	True on success, False if an object with the specified ID could not be found or the pointer was invalid.

```
BOOL WINAPI SetInfo(POBJECTINFO ObjectInfo);
Function SetInfo(ObjectInfo : POBJECTINFO) : BOOL; stdcall;
```

Finds the object with the ID passed in TObjectInfo structure, and sets the object name, size, position, and rotation angle based on the other fields in the structure.

ObjectInfo	Points to a TObjectInfo structure in which the ObjID field has been set to the handle of an object to be modified, and the name, size, and rotation information in the TObjectInfo structure has the new settings.
Returns:	True on success, False if an object with the specified ID could not be found or the pointer was invalid.

If the object name passed in the structure is not the same as the current name of the object, and an object by the same name already exists on the label, the name is not changed. If name is null, then just the size and rotation setting are modified.

```
BOOL WINAPI GetLabelMargins(RECT *pRect);
Function GetLabelMargins(Rect :PRECT) : BOOL; stdcall;
```

Returns the widths of the unprintable margins, in TWIPS, for the currently selected paper type.

```
void WINAPI SetPrinterModeState (BOOL State);
Procedure SetPrinterModeState(State : BOOL); stdcall;
```

Enables (State= TRUE) or disables (State = FALSE) the automatic switching to Barcode and Graphics mode when printing labels with non-monochrome bitmaps and/or barcodes that are set to small or medium size.

```
void WINAPI SaveLastLabel ();
Procedure SaveLastLabel(); stdcall;
```

This saves the currently opened label and all its changes in memory. Call LoadLastLabel to retrieve the label from memory. See LoadLastLabel().

```
void WINAPI LoadLastLabel ();
Procedure LoadLastLabel(); stdcall;
```

This loads the label saved to memory. See SaveLastLabel().

```
void WINAPI SetLabelBackground (int Color);
Procedure SetLabelBackground(Color: Integer); stdcall;
```

Changes the background color of the label.

```
int WINAPI GetLabelOrientation(char *LabelName);
Function GetLabelOrientation(LabelName: PCHAR):Integer; stdcall;
```

Returns 0 if the label's orientation is portrait, 1 if the orientation is landscape.

```
int WINAPI SetPrintDefaultMode(BOOL Mode);
Procedure SetPrintDefaultMode(Mode: BOOL); stdcall;
```

When a label object has blank data, it will show a default string (i.e. "Click here to enter text…") in place of the blank data. This default string is not printed by default (i.e. it's only shown when the label is rendered on the screen, not when the label is printed on a LabelWriter). Call this function to enable (Mode=TRUE) or disable (Mode=FALSE) printing of a label object's default string.

```
BOOL WINAPI GetDefaultStringMode();
Function GetDefaultStringMode(): BOOL; stdcall;
```

Returns TRUE if "default strings" are printed, FALSE if default strings are NOT printed.

```
BOOL WINAPI CanUndo();
Function CanUndo(): BOOL; stdcall;
```

Labels.dll now tracks changes made to a label so applications can include both the Undo and Redo editing feature. This function returns TRUE if there are editing commands in the undo buffer, FALSE if there are none.

```
BOOL WINAPI CanRedo();
Function CanRedo(): BOOL; stdcall;
```

Labels.dll now tracks changes made to a label so applications can include both the Undo and Redo editing feature. This function returns TRUE if there are editing commands in the redo buffer, FALSE if there are none.

```
void WINAPI Undo();
Procedure Undo(); stdcall;
```

Undo the last label editing command.

```
void WINAPI Redo();
Procedure Redo(); stdcall;
```

Redo the last label editing command.

```
void WINAPI StartPrintJob();
Procedure StartPrintJob(); stdcall;
```

To take advantage of the faster print speed of the LabelWriter 400 series printer, wrap all PrintLabel() and PrintLabelEx() function calls with a StartPirntJob() and EndPrintJob function call.

```
void WINAPI EndPrintJob();
Procedure EndPrintJob(); stdcall;
```

To take advantage of the faster print speed of the LabelWriter 400 series printer, wrap all PrintLabel() and PrintLabelEx() function calls with a StartPirntJob() and EndPrintJob function call.

```
BOOL WINAPI ReadURLFile(char *sURL);
function ReadURLFile(sURL: PCHAR): BOOL; stdcall;
```

Open a label file using a URL to specify is the location of the file. The function operates the same way as the ReadLabelFile() function.

```
BOOL WINAPI ReadStreamLabelFile(char *Stream, int Size);
function ReadStreamLabelFile(Stream: PChar; Size: Integer):
                    BOOL; stdcall;
```

Open a label file by reading the file content from a memory buffer (i.e. Stream). The memory buffer's size should be specified with the Size parameter and must be passed to the function or the call will fail. The function operates the same way as the ReadLabelFile() function.

```
int WINAPI WriteStreamLabelFile(char *Stream, int Size);
function WriteStreamLabelFile(Stream: PChar; Size: Integer):
                    Integer; stdcall;
```

Receive the label file content in a memory buffer. To find out the buffer size required to receive the file content, call the function using a NULL for the Stream parameter. The function will return the size of the buffer needed. When passing a real buffer, the size parameter should be the size of the buffer and must be passed to the function or the call will fail. The memory buffer's size must be specified or the function will fail. The function operates the same way as the ReadLabelFile() function.

```
void WINAPI SuppressErrorMessage(BOOL bSuppress);
Procedure SuppressErrorMessage (bSuppress: BOOL); stdcall;
```

Several error messages can be displayed from Labels.dll. These message can halt the operation of an unattended system. To prevent the error messages from displaying, call the function to suppress the error messages.

Appendix A
Licensing and Distribution

Licensing Grants and Restrictions

The DLS application and its associated DLLs are the proprietary property of DYMO Corporation, and are protected by United States copyright laws and international copyright treaties.

You may not:

- Disassemble or reverse-engineer the DLS programs.
- Distribute, rent, sell, sub-license, or otherwise make available to others the software or documentation, or copies thereof, except as expressly permitted in this license.

You may:

- Use any and all sample source code supplied as part of this Software Developers Kit.
- Incorporate and distribute all files that make up the "DYMO Label Software" and "Drivers and Utilities" installation programs as part of your application provided that:
    1. Any computer-readable distribution medium (Diskette, CD-ROM, or other), contains the notice "Contains proprietary, copyrighted programs of DYMO Corporation."
    2. All files are distributed in their entirety.
    3. You notify DYMO Corporation, in writing, of the decision to distribute the DLS program. Notification must be sent to "Legal Department, DYMO Corporation, 44 Commerce Road, Stamford, CT 06902-4561" prior to initial shipment.
    4. You provide the most recent versions of all files as downloaded from the DYMO web site at www.DYMO.com/labelwriter or as received from the DYMO Customer Service department in response to your notification of intent to distribute.

With the exception of that which is explicitly stated herein, these grants and restrictions are not in lieu of any other licenses pertaining to the software supplied by DYMO Corporation.

This agreement is governed by the laws of the State of Connecticut. If any provisions are held void or unenforceable, the remainder shall remain valid and enforceable according to its terms.

Questions?

Should you have questions pertaining to the rights and restrictions herein, please contact our Legal department in writing at "Legal Department, DYMO Corporation, 44 Commerce Road, Stamford, CT 06902-4561" or by email at "sdkhelp@dymo.com" with your questions

Appendix B
Barcode Settings

Modifying the Barcode Behavior

For some specialized applications, it may be necessary to override the default barcode dimensions. Although most applications will not require this, the DLS application does provide a means of controlling the base unit width for barcodes. However, this adjustment feature is NOT available for tape label printing, only paper label printing.

By default, the base unit width is 0.010 inches, but this can be modified using the INI file settings described in this appendix. The base unit width defines the narrowest units used for small barcodes. The unit widths used for medium and large barcodes are 1.5 and 2 times the base unit width, respectively. This ratio cannot be modified.

The [Barcode Settings] INI Section

To override the behavior of the barcodes, create a new section called [Barcode Settings] in the DYMOLBL.INI file. Within this section, you can add any of five different entries:

Entry	Purpose	Expressed in (Dimensions)
Narrow Space	Sets width of narrow spaces	Thousandths of an inch
Wide Space	Sets width of wide spaces	Thousandths of an inch
Narrow Bar	Sets width of narrow bars	Thousandths of an inch
Wide Bar	Sets width of wide bars	Thousandths of an inch
Bar Code Compensation	Compensates for thermal blooming of bars	Bit field (see below)

Bar Code Compensation is used to force the bar width to be one less than the calculated value to account for thermal blooming (expansion in width) of the bar. The compensation is given as a bit field, where Bit 0 corresponds to small barcodes, Bit 1 is for medium barcodes, and Bit 2 is for large barcodes. If a given bit is set, then compensation is ON. By default, compensation is on for small barcodes only.

> *NOTE: Because each printer pixel is approximately 5 Mils wide (1/203), the smallest setting that can be used for bar widths or spaces, and still cause a ba code to print is 5 Mils. The software will round the given dimension to the nearest pixel width.*

A sample barcode settings section might look like:

[Barcode Settings]
Narrow Space = 10
Wide Space = 20
Narrow Bar = 10
Wide Bar = 20
Bar Code Compensation = 1

Index