DLS SDK

Reference Manual for Windows

Table of Contents

Chapter 1
Getting Started

Introduction

The DLS SDK provides a binary compatible library to support existing Software Development Kit (SDK) applications that were developed using the DLS SDK. The 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#
- VB and VB.NET
- VBScript
- JavaScript
- Microsoft Access
- any other application that support COM

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.
- Built-in support for photo ID applications.
- Built-in support for reading, re-sizing, and printing JPG, TIF, PNG, 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.
- Intelligent Mail barcodes for faster delivery of mail (replacing the existing POSTNET barcode).

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

- Printing shipping labels from your corporate order-entry application.
- Printing barcode inventory labels from your accounting application.
- Printing patient file folder labels and barcode medical labels from your medical office automation application.
- Printing address labels and seminar badges for your meeting management program.

Before you continue

You should have an understanding of COM (Component Object Model) before attempting to use this SDK. This reference manual is not intended to teach you everything there is to know about COM.

What's in the SDK

The SDK includes many samples showing how to print labels using the DLS SDK**.**

Only samples and documentation are installed with the SDK.

The DLS SDK is an "implementation" of the interfaces described in the SDK. This library is installed automatically when the DYMO Label v.8.2 or later application is installed.

Samples provided in the SDK are written in C++, C#, VB.NET, and Microsoft Access, 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.

What's New in the DLS SDK

Besides supporting the existing COM interfaces from the DLS 7 SDK, the new DLS SDK includes the following new features:

- Implemented using the new DYMO Label v.8 codebase. No need to have DLS 7 installed to run DLS 7 SDK-based applications.
- Binary compatible with existing SDK applications.
- Supports opening and printing of DLS 7 and DYMO Label v.8 label file formats (".lwl" and ".label" files).
- Adds support for LabelWriter 450 series and 4XL printers.
- Supports Intelligent Mail barcode in Address Objects (replacing POSTNET barcode).
- Supports customized web proxy settings (see IDymoAddin6 interface).
- Supports querying printer's online and offline status (see IDymoAddin6 interface).
- Supports setting image via URL (see IDymoLabels3 interface).
- Supports print job control in the low level COM interface (see ILabelEngine4 interface).

What is ***NOT*** supported by the DLS SDK

Due to implementation restrictions and other factors, the new DLS SDK does not support the following SDK interfaces:

- DDE Interface
- DLL Interface
- D1 Tape Library API
- Show, Hide and Quit API calls.
- Barcode Customization (see Appendix B)

If your application is written using any of these interfaces, you are required to continue to use the DLS 7 application to support your application. To enable printing to the latest LabelWriter 450 series and 4XL printers, you will need to install an update to the DLS 7.8 installation (the update installer can be found on the DYMO Support Web site).

Installing the SDK

You install the SDK and the DLS SDK in two steps:

1. Install the DYMO Label v.8 software. This installs the necessary libraries needed to develop SDK applications.
2. Install the SDK. This installs only SDK samples (source code included) and documentation.

Refer to the *Quick Start Card* that came with your LabelWriter printer for instructions on installing DYMO Label software. If you downloaded the SDK from the DYMO Web site, run the downloaded file to install the SDK.

Refer to the ReadMe.txt document in the SDK Samples folder for an overview of the samples included in the SDK.

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 Page.
- You can also send email to sdkhelp@dymo.com to get support by email, usually in less than 24 hours.

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

- The DYMO Label 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
DLS SDK Background

Overview

The DLS SDK is an implementation of the most commonly used DLS SDK API (the High-Level and Low-Level COM Interfaces) using the new DYMO Label v.8 codebase. The new implementation is backward compatible with existing SDK applications and continues to allow new application developers to easily adapt various label printing solutions using different programming languages.

High-Level COM Interface

The High-Level COM interface provides the easiest programming interface for the most common label printing applications. The interface allows an application to open an existing or customized label file, change text, barcode or address data, and print the label.

Low-Level COM Interface

Unlike the high-level interfaces, the low-level interface provides developers with full control over the size, attributes, and appearance of labels. Everything that can be done through the high-level DYMO Label interfaces can be done through the low-level COM interface as well, but requires 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 COM interface.

Program Architecture

The DLS SDK architecture is shown in the figure below:

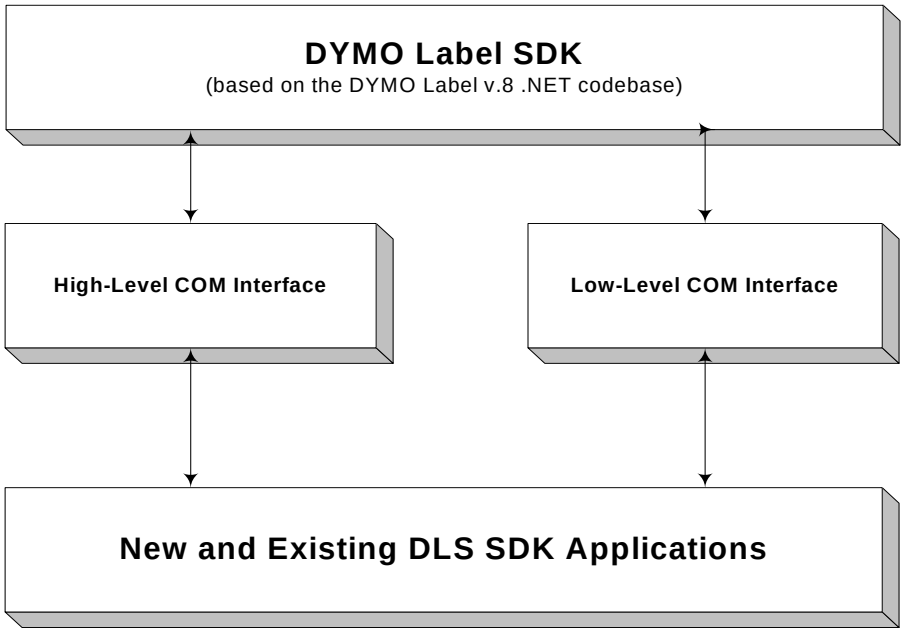

Figure 1

As shown in Figure 1, both the High-Level and the Low-Level COM interface control is provided by the DLS SDK implementation. This is different from DLS 7 where the Label Engine DLL provided the implementation for the Low-Level COM interface and the Dymolbl.exe (the DYMO Label application) provided the implementation for the High-Level COM interface.

Chapter 3
DLS SDK
Interface Descriptions

High-Level COM Interface

Purpose and Capabilities

The High-Level COM Interface provides an easy-to use interface for SDK applications to perform the most frequently used label printing functions. The goal of the interface is to allow SDK applications to print labels using the DYMO LabelWriter printers with minimal code. While the interface is simple, it is full-featured and provides solutions for a wide range of label printing SDK applications.

Interface Description

Program ID Information

The COM interface exports two main COM interfaces: **IDymoAddIn** and **IDymoLabels.** The COM servers that implement these interfaces can be created using the following class names:

Dymo.DymoAddIn
Dymo.DymoLabels

The DLS SDK also provides updated versions of the **IDymoAddIn** and **IDymoLabels** interfaces (see IDymoAddin6 and IDymoLabels3 interfaces detailed descriptions).

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.

*Functional Difference Alert:

The function will try to open the specified label file name with the .label extension first, even if the parameter specifies a different file extension.

For example, if your application calls:

IDymoAddIn.Open(“mylabel.lwl”);

The function will try to look for the file in the following order:

mylabel.label

mylabel.lwl

mylabel.lwt

* The reason for this behavior has to do with how the implementation handles both ".lwl" and ".label" file formats. The implementation converts ".lwl" format into ".label" format internally before performing any actions on a label file. What this means is that when a label that was opened as ".lwl" will be saved as a ".label" file if the Save() or SaveAs() method is called.

So if an SDK application opens a ".lwl" label file, modifies it, then saves the file. The file would be saved as a ".label" file. When the application returns to open the same ".lwl" file expecting to see the modifications, the Open() method would need to open the ".label" file for the modification to be seen.

`Save()`

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

*Functional Difference Alert:

The function will save the currently opened label in the .label file extension, even if the parameter specifies a different file extension.

For example, if your application calls:

IDymoAddIn.Save();

The function will save the file with the same name as the currently opened file but with a .label file extension. The label file is in the new DYMO Label v.8 .label file format.

`SaveAs(const FileName: WideString)`

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

*Functional Difference Alert:

The function will save the specified label file name in the .label file extension, even if the parameter specifies a different file extension.

For example, if your application calls:

IDymoAddIn.SaveAs(“newlabel.lwl”);

The function will save the file as “newlabel.label”. The label file is in the new DYMO Label v.8 .label file format.

`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).*

*Functional Difference Alert:

The bShowDialog parameter is ignored. No print progress dialog is shown.

`Hide()`

The function has been deprecated to no-op. If your existing application depends on this deprecated feature, then you are required to use the DLS 7 application. The DLS SDK cannot support your existing SDK application.

`Quit()`

The function has been deprecated to no-op. If your existing application depends on this deprecated feature, then you are required to use the DLS 7 application. The DLS SDK cannot support your existing SDK application.

`Show()`

The function has been deprecated to no-op. If your existing application depends on this deprecated feature, then you are required to use the DLS 7 application. The DLS SDK cannot support your existing SDK application.

`GetDymoPrinters()`

Returns a list of DYMO printer names. The vertical-bar '¦' character separates each printer name in the list.
Example:
DYMO LabelWriter 450 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)`

The function has been deprecated to no-op. If your existing application depends on this deprecated feature, then you are required to use the DLS 7 application. The DLS SDK cannot support your existing SDK application..

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:<sample>\Label Files\Address (30252, 30320).LWL¦C:<sample>\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.

*Functional Difference Alert:

The function used to return the same list of files in the "Label Files" dropdown of the DLS 7 application. It's changed to:

The first time any SDK application is run, the MRU list is initialized with the "Recent Layouts" list from the DYMO Label v.8 application. Once the MRU list is initialized, it is maintained separately from the DYMO Label v.8 application. All SDK applications share the same MRU list within the same user account.

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.

*Functional Difference Alert:

The bShowDialog parameter is ignored. No print progress dialog is shown.

`GetCurrentPaperTray()`

Returns the current active paper tray.

Further Information: When the currently selected printer is a LabelWriter Twin Turbo, DYMO Label software 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, DYMO Label software 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()`

Wrapping IDymoAddIn4.PrintLabel() and IDymoAddIn4.PrintLabelEx() calls within the StartPrintJob() and EndPrintJob() calls will cause labels to be printed as pages of the same print job. The benefit is seen with reduced the print job

overhead and increased label printing speed when printing to LabelWriter 400 and 450 series printers.

Example:

```
// this printing loop creates a 10 page print job
StartPrintJob();
for (i = 0; i < 10; i++)
{
      // update some fields on the label
      ...

      LabelEngine.PrintLabel(…); // print one label
}
void EndPrintJob();
```

this code above will print labels much faster than the code below:

```
// this printing loop creates 10 different one page print jobs
for (i = 0; i < 10; i++)
{
      // update some fields on the label
      ...

      LabelEngine.PrintLabel(…); // print one label
}
```

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 DYMO Label software, refer to the DYMO Label software 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 DYMO Label software.

IDymoAddIn6 Properties and Methods

The **IDymoAddIn6** interface inherits directly from the **IDymoAddIn5** interface and provides the following additional program control functions. The new functions were added to support proxy settings for URL related functions, such as the OpenURL() and SetImageURL() .

```
void SetupProxySettings(string protocol, string serverName, long
                    Port, string proxyBypass, string userName,
                    string password)
```

Allows customized proxy settings (different from IE's default proxy settings). All URL related function calls in the SDK will adhere to these proxy settings.

```
void ClearProxySettings()
```

Clears all proxy settings and revert back to using IE's default proxy settings.

```
bool proxyBypass
```

Setting the property to **true** will cause all URL related SDK functions to bypass any proxy settings, including IE's default proxy settings.

Setting the property to **false** (the default value) means all URL related SDK functions will use either IE's default proxy setting or the user specified proxy settings.

```
bool IsPrinterOnline(string PrinterName)
```

Returns true if the specified printer is online, false if the printer is offline. This applies to locally connected printers only. Shared printers always return true.

```
void SetGraphicsAndBarcodePrintMode(bool isModeOn)
```

When the mode is on (default value), labels containing barcode(s) will print at high quality mode but the print speed is reduced. Unsetting this mode will cause all labels to print at the fasted print speed.

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.

IDymoLabels3 Properties and Methods

The **IDymoLabels3** interface inherits from the IDymoLabels2 interface and provides the following additional function:

`SetImageURL(string ObjectName, string imageURL, string imageTypeStr)`

Allows specifying URL as the image source for an image object on the label. The "imageTypeStr" parameter was used to indicate the type of the image file in the URL, but it is no longer needed in the new implementation.

Low-Level COM Interface

Purpose and Capabilities

The Low-Level COM interface is a collection of objects that provides application developers the ability to create full-featured label printing applications. Although considerably more complex than the High-Level COM interface, it gives SDK applications full control to open, create, modify, save, and print labels.

ILabelEngine COM Interface

Introduction

As is shown in Figure 1, the Low-Level COM interface is implemented by the same DLS SDK that implements the High-Level COM interface. The added benefit (not seen in the previous

implementation) is that use of High-Level and Low-Level COM objects (i.e. calling methods, setting and reading of property values) can be combined but still produce consistent results. For example, if you open a label file using the High-Level COM interface and modified the label using the Low-Level COM interface, then print the label using the High-Level COM interface, you can expect the printed label to contain the modifications.

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 following table. All of these interfaces can be obtained from 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:

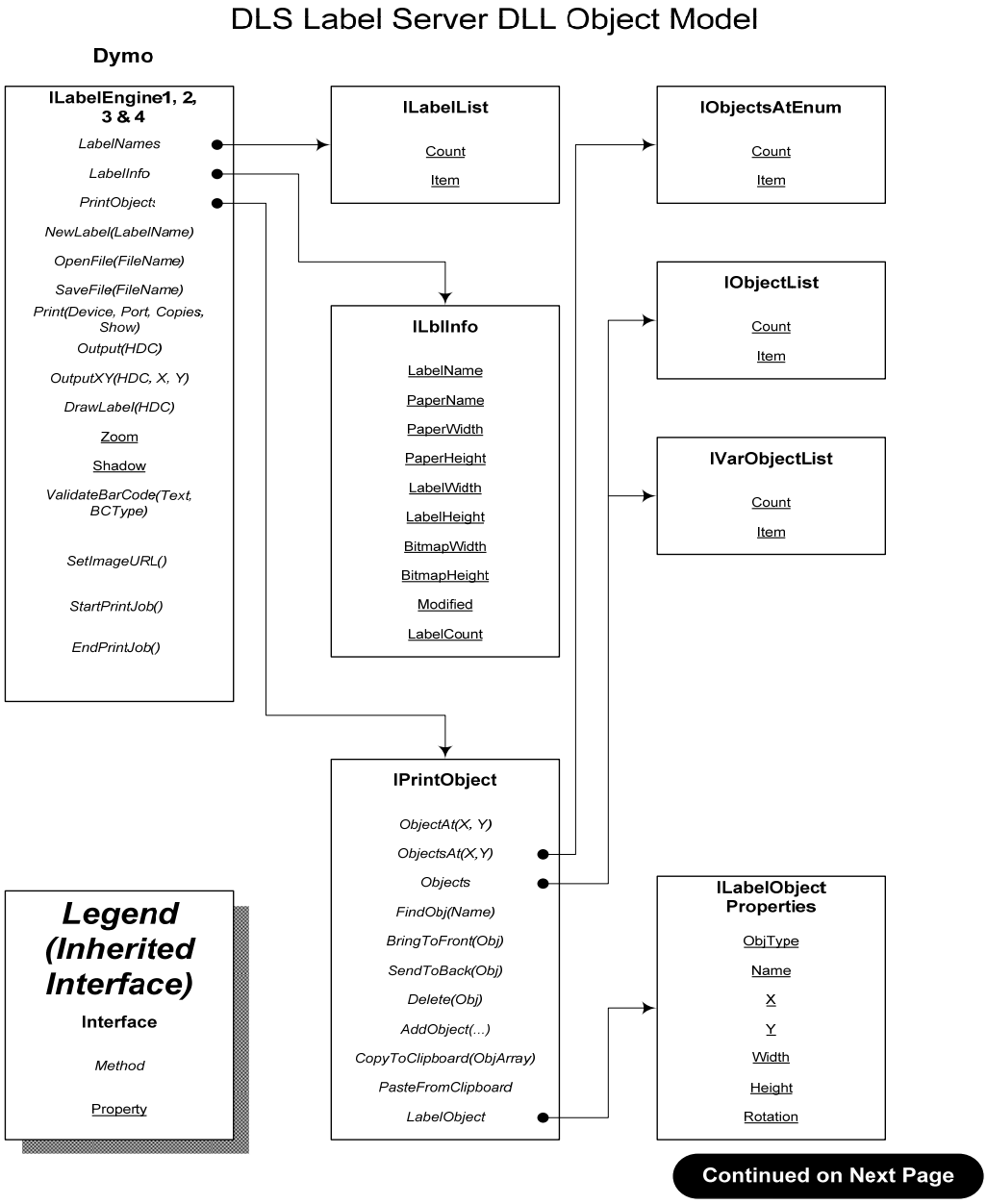

Figure 2

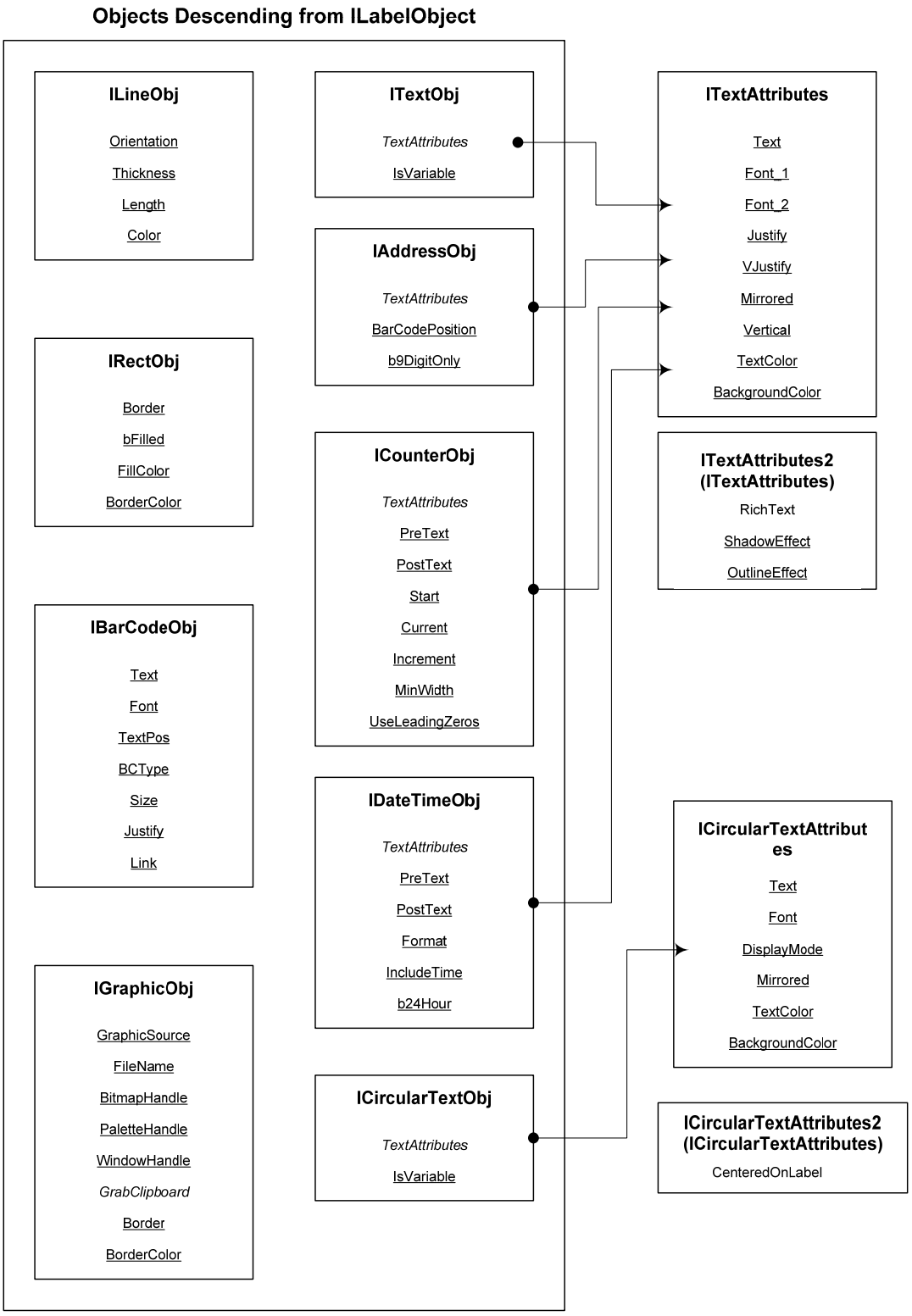
Objects Descending from ILabelObject
ILineObj
Orientation
Thickness
Length
Color
IRectObj
Border
bFilled
FillColor
BorderColor
IBarCodeObj
Text
Font
TextPos
BCType
Size
Justify
Link
IGraphicObj
GraphicSource
FileName
BitmapHandle
PaletteHandle
WindowHandle
GrabClipboard
Border
BorderColor
ITextObj
TextAttributes
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 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 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 19.)

`LabelInfo` Returns an interface to the ILblInfo object. (See Figure 2 on page 19.)

`PrintObject` Returns an interface to the IPrintObject object. (See Figure 2 on page 19.)

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 450 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.

ILabelEngine3 Properties and Methods

The **ILabelEngine3** interface inherits from the ILabelEngine2 interface and provides the following additional functions:

```
OpenStream(Buffer: VARIANT)
```

Reads a label file from a buffer. 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 DYMO Label software.

ILabelEngine4 Properties and Methods

The **ILabelEngine4** interface inherits from the ILabelEngine3 interface and provides the following additional functions:

`StartPrintJob()`

`EndPrintJob()`

Wrapping ILabenEngine4.PrintLabel() and ILabenEngine4.PrintLabelEx() calls within the StartPrintJob() and EndPrintJob() calls will cause labels to be printed as pages of the same print job. The benefit is seen with reduced the print job overhead and increased label printing speed when printing to LabelWriter 400 and 450 series printers.

Example:

```
// this printing loop creates a 10 page print job
StartPrintJob();
for (i = 0; i < 10; i++)
{
      // update some fields on the label
      ...

      LabelEngine.PrintLabel(…); // print one label
}
void EndPrintJob();
```

this code above will print labels much faster than the code below:

```
// this printing loop creates 10 different one page print jobs
for (i = 0; i < 10; i++)
{
      // update some fields on the label
      ...
```

```
        LabelEngine.PrintLabel(…); // print one label
    }
```

ILabelList

The implementation returns the new label type names in DYMO Label v.8. However, calling the function with old DLS 7 label names will continue to work.

For example, when the DLS 7 “Address (30252)” label name is used, the implementation will map it to the equivalent “Address Label” in **DYMO Label v.8**..

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.

The implementation changed to that Font1 and Font2 fields are not kept separately from the text field: if the text contains formatting information, the Font1 and Font2 values represent the font format of line1 and line2 + subsequent lines. Setting the Font1 and Font2 property will change the text so line1 of text is in Font1 format and line2 + subsequent lines are in the Font2 format.

If you set RTF text in the Text field, Font1 and Font2 fields are automatically changed to reflect the font formats in line1 and line2 + subsequent lines.

If you set plain text in the Text field and the Text field was empty, the default font format (i.e. Arial, 16pt) is used for the plain text. If the Text field holds some value, then the font format for the plain text will derive from the previous text field data.

The following font styles are not supported:

Shadow font

Outline font

Properties and Methods:

`Text`	Read/Write. String displayed by Text and Address objects. Ignored by the Date/Time and Counter objects. The DLS SDK implementation will return plain text always. The previous implementation will return RTF formatted text if the object was initialized with RTF text

`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.

Appendix A
Barcode Settings

Modifying the Barcode Behavior

The ability to adjust barcode generation behavior has been deprecated in the DLS SDK implementation. If your existing application depends on this deprecated feature, then you are required to use the DLS 7 application. The DLS SDK cannot support your existing SDK application

Index