Ship Manager API

Label Extensions Developer's Guide

Table of Contents

FedEx Ship Manager API

Shipping Label Extensions

Overview

FedEx Ship Manager API is a software-based product that allows customers to create their own shipping applications using a variety of development environments, such as Microsoft Visual Basic, C/C++, PowerBuilder, VBScript, JavaScript and other possible environments.

Since there are a variety of development environments, FedEx Ship Manager API also provides the mechanisms for a customer to create shipping labels of various types to accommodate the different printing scenarios.

However, FedEx Ship Manager API itself does not provide the mechanism to directly create and send the labels to a customer's printer; previously that has been the function of the customer's browser. It became necessary to develop an extension to FedEx Ship Manager API to provide these functions to the programmer when necessary.

The purpose of this document is to provide documentation and support for creating shipping labels for printing within the customer's client-side application or browser by using the FedEx-provided 'Shipping Label Extensions' tool. The installation and use of this tool will be outlined in this document.

FedEx Label Types

FedEx Ship Manager API supports the use of two types of labels. These types of labels are used for different scenarios. The label types are:

1. A Plain Paper label generated from a ‘PNG’ image that consists of a FedEx generated image and an HTML wrapper that contains the FedEx legal agreement and instructions. This label is usually printed from a browser to a customer's laser printer.
2. An adhesive-backed 4 x 6.75 inch label that is printed to a supported thermal printer. This label is usually used for high-volume printing situations.

Labels can be categorized by three attributes: label type, printer type and format.

- The label type (only 1 = standard is currently allowed) is specified with field 1368 in the Ship transaction.
- The printer type (1 = plain paper (such as laser), 2 = Eltron Orion, Zebra LP2443, LP2844, LP2348 Plus, or 3 = Eltron Eclipse) is specified with field 1369.
- The format type (3 = 4x6¾ thermal with doc-tab, 4 = 4x6¾ thermal without doc-tab and 5 = plain paper PNG) is specified with field 1370.

While the content of each of these labels is the same, their usages are different. Each provides a bar-coded image for the purpose of scanning the package during handling. The uses of each label depend on the type of working environment to be used.

Each label requires specific support files or requirements that are described in the following sections of this document.

FedEx Supported Thermal Printers

The following thermal printers are currently supported via FedEx Ship Manager API:

Eltron Orion
Eltron Eclipse
Zebra LP2443
Zebra LP2844
Zebra LP2348 Plus

These printers are all compatible with the ASCII EPL2 programming language (page mode). If your printer supports this programming language it may work as well. Check your printer users guide for details.

Thermal printers are supported both as a direct write to the printer connected to a system serial port, or as a native Windows installed printer for LPT, Serial, or USB connections. Refer to the methods description part of this document for details.

Signature Proof of Delivery Letter (SPOD)

FedEx Ship Manager API can be used to request a Signature Proof of Delivery letter. This package can be used to provide a letter image for printing from a browser or saving to a file.

Shipping Label Extensions Package

FedEx Ship Manager API Shipping Label Extensions consists of the following:

- The FedEx Ship Manager API Label Extensions DLL - LBLEXTS.DLL
- The FedEx Ship Manager API Label Extensions Library - LBLEXTS.LIB
- The FedEx Ship Manager API Label Extensions Type Library – LBLEXTS.TLB
- The FedEx Ship Manager API Label Extensions error codes, called LBLEXTS.H
- Support files for various label types. These files are shown below:

1) For Browser-based labels:
   domlabel.html - an HTML template for US Domestic labels in .gif format*
   usdompng.html - an HTML template for US Domestic labels in .png format
   intlabel.html - an HTML template for International labels in .gif format.*
   intlpng.html - an HTML template for International labels in .png format.
   candomlabel.html - an HTML template for intra Canada shipments in .gif format.*
   candompng.html - an HTML template for intra Canada shipments in .png format.
   canintlabel.html - an HTML template for Canadian export shipments in .gif format.*
   canintlpng.html - an HTML template for Canadian export shipments in .png format.
   returninstructions.html - an HTML template for Return Manager instructions.
   * Deprecated

2) For Signature Proof of Delivery Letters
   spodpng.html – an HTML template for proof of delivery letters

All of these files are included in FedEx Ship Manager API. The HTML files are in the 'Labels' folder. The LBLEXTS.DLL is in the \WIN32\BIN folder, LBLEXTS.LIB and LBLEXTS.TLB are in the \WIN32\LIB folder and LBLEXTS.H is in the \WIN32\INCLUDE folder. These folders can be found where you installed FedEx Ship Manager API. The default location is C:\Program Files\FedEx\FedEx Ship Manager API. The JLblexts.class is included in the JLblexts.jar file. See “Using with Java” and the method descriptions later in this document for further information.

The sections below cover how to use the FedEx Ship Label Extensions within a 'C' based programming environment. These rules also apply to other 32 bit programming languages such as POWERBUILDER, and 32 bit languages that can call Windows-based DLL's. FedEx Ship Label Extensions also includes a COM interface for use with VB, VBScript, JavaScript, and can be used with C++ programs, refer to the *Ship Label Extension Function Calls and COM Interfaces* section of this document for a description of the functions and interfaces. There are samples of using the COM interface in VB, C++ and Active Server Pages (using both VBScript and JavaScript) at http://www.fedex.com/globaldeveloper/shipapi. There are also Java and C sample programs available at this URL. Note: you must have the appropriate development and runtime environments to run these samples.

Placement of Ship Label Extension files

Regardless of language used, the placement of the files provided with Ship Label Extensions is up to you. Please remember the following when placing your files:

1. Place all the HTML files from the Ship Label Extensions Package in a directory where your application can access them.
2. If you are not using the COM interface place LBLEXTS.DLL in your applications directory or the system directory.
3. If you are using the COM interface place LBLEXTS.DLL in a directory where your application can use it and register it with regsvr32.exe.
4. Create a directory in which to store the label files.

Linking Your Program to Ship Label Extensions

You must be able to link your program to the Ship Label Extensions DLL. Most Windows-based languages do this by declaring a function prototype of the functions required. For 'C' and 'C++' programming, use the file LBLEXTS.H to add to your project to declare the functions. If using 'C' or 'C++', add the file LBLEXTS.LIB to your project file to link your code to the library. Most other languages do not require the file LBLEXTS.LIB to link to the Ship Label Extensions DLL. To use the COM interface with C++ import the LBLEXTS.TLB file using the **#import** preprocessor statement. From MS Visual Basic 6.0, open your project and select **Project** then **References**. Find the LBLEXTS Type Library in the list and select the checkbox.

Determining Error Status non-COM

All functions in the Ship Label Extensions DLL return a 32 bit Windows-style Integer as the status code. Programmers using the functions should test the return value for a valid error code. Valid error codes are always negative numbers, with 0 reserved for successful response. Refer to the file LBLEXTS.H for a complete list of error codes.

Determining Error Status using COM

All interfaces set an HRESULT, which can be checked to determine success or failure. For examples of how to check for errors in your programming language see the samples which can be downloaded from http://www.fedex.com/globaldeveloper/shipapi/.

Selecting the Label Format

To properly use Ship Label Extensions, you must set fields 1368, 1369, 1370 in your send transaction buffer before calling FedExAPITransaction( ). The value of these fields should be

- Field 1368 – Always set to “1” for a standard label.
- Field 1369 – “1” for Plain Paper (Printing from Browser), “2” for Eltron Orion, Zebra LP2443, LP2844, LP2348 Plus, or “3” for Eltron Eclipse.
- Field 1370 – “3” for 4x6.75 inch thermal with doc-tab, “4” for 4x6.75 inch without doc-tab or “5” for .png format.

Ship Label Extension Function Calls and COM Interfaces

The following sections deal with the actual function calls that perform the label creation. There are 6 new functions and 4 deprecated functions.

These functions are normally called immediately after the FedEx Ship Manager API function call FedExAPITransaction( ). The reason is that the Ship Label Extensions require access to the request and response buffer contents of the last shipment to perform their functions. A typical sequence is as follows:

1. Program calls FedExAPITransaction( ) to perform a shipping transaction
2. Program then calls one of the Ship Label Extensions to print or save a label, such as:

   DisplayLabelInBrowser( ) – to print a PNG style label

To print a Signature Proof of Delivery letter, first send the transaction and then call DisplaySPODInBrowser ().

Using the COM Interface

Refer to the sample code available at the FedEx Ship Manager API website http://www.fedex.com/globaldeveloper/shipapi/ for usage of the COM interface for your programming language. These samples illustrate how to create an instance of the interfaces and how to convert the HRESULT to an API defined error code.

Using with Java

To use the label extensions from your Java program you must place the Jlblexts.jar file in the classpath or current application directory, then create an instance of the JLblexts object. For example: JLblexts JLabel = new JLblexts();

The Jlblexts.jar file uses JNI to access the C library.

For WIN32: The lblexts.dll file must be located in a directory defined in the system PATH variable.

For Solaris/Linux: JNI libraries must be in the path defined by the Java property "java.library.path" to be loaded with the System.loadLibrary("libname") method. To extend the java.library.path, define the environment variable DYLD_LIBRARY_PATH with the additional directories, and the FedEx Ship Manager API shared libraries (.so files) should be copied into that directory.

You can override the java.library.path in the Java 2 SDK release 1.2 from the command line. For example, the following command starts up a program Foo which needs to load a native library in the c:\mylibs directory:

```
java -Djava.library.path=c:\mylibs Foo
```

Displaying PNG image label in a browser

This type label is used when there is a browser available to display and print the label. The scenario is that the user's application will call the DisplayLabelInBrowser() function which will launch the default browser (if it's not running) or use the browser if it is running to display the label. The user will then click the browser's 'print' button and print their copy of the label.

Function Prototype

```
LabelStatus = DisplayLabelInBrowser ( char *RequestBuffer,
                                      char *ReplyBuffer,
                                      char *TemplatesPath,
                                      char *SavePath);
```

OS Compatibility

WIN32

Com Interface

```
HRESULT DisplayLabelInBrowser([in] BSTR RequestBuffer,
                              [in] BSTR ReplyBuffer,
                              [in] BSTR TemplatesPath,
                                   [in] BSTR SavePath);
```

Java Method

```
LabelStatus JDisplayLabelInBrowser( byte [] RequestBuffer,
                                    byte [] ReplyBuffer,
                                    byte [] TemplatesPath,
                                    byte [] SavePath );
```

Where:

LabelStatus = an Integer status code
RequestBuffer = the shipment request
ReplyBuffer = the shipment response
TemplatesPath = the path where the label extension .html files are stored (MyApp\Templates)
SavePath = the path where the label files are to be stored (MyApp\Labels)

Saving the PNG label image to a file

To extract the label image file and generate the HTML file and image without displaying the label in a browser call the function SaveLabelToFile().

Function Prototype

```
LabelStatus = SaveLabelToFile ( char *RequestBuffer,
                                char *ReplyBuffer,
                                char *TemplatesPath,
                                char *SavePath,
                                char *SavedFileName );
```

OS Compatibility

WIN32, SOLARIS, LINUX

Com Interface

```
HRESULT SaveLabelToFile([in] BSTR RequestBuffer,
                        [in] BSTR ReplyBuffer,
                        [in] BSTR TemplatesPath,
                        [in] BSTR SavePath,
                        [out, retval] BSTR* SavedFilename);
```

Java Method

```
LabelStatus JSaveLabelToFile( byte [] RequestBuffer,
                              byte [] ReplyBuffer,
                              byte [] TemplatesPath,
                              byte [] SavePath,
                              byte [] SavedFileName );
```

Where:

LabelStatus = an Integer status code

RequestBuffer = the shipment request

ReplyBuffer = the shipment response

TemplatesPath = the path where the label extension .html files are stored (MyApp\Templates)

SavePath = the path where the label files to be stored (MyApp\Labels)

SavedFileName = the name of the HTML file generated to display the label in a browser. The image file will have the same name with an extension of .png. In the case of a COD shipment the HTML file will have a link to the COD label.

Printing a Thermal label

This type label is used when there is a FSM API supported thermal printer attached to the user's system. These type labels are used when speed of printing is desired and using self-adhesive labels is a requirement.

Function Prototypes

To print directly to the thermal printer via a serial port:

```
LabelStatus = PrintThermalLabel(char *RequestBuffer,
                                char *ReplyBuffer,
                                short CommPort,
                                long  BaudRate,
                                char *SavePath);
```

To print to the thermal printer via the Windows Print Manager:

```
LabelStatus = PrintThermalWithNamedPrinter(char *PrinterName,
                                char *RequestBuffer,
                                char *ReplyBuffer,
                                char *SavePath);
```

OS Compatibility

WIN32

Com Interface

```
HRESULT PrintThermalLabel([in] BSTR RequestBuffer,
                          [in] BSTR ReplyBuffer,
                          [in] short ComPort,
                          [in] long BaudRate,
                          [in] BSTR SavePath);

HRESULT PrintThermalWithNamedPrinter([in] BSTR PrinterName,
                          [in] BSTR RequestBuffer,
                          [in] BSTR ReplyBuffer,
                          [in] BSTR SavePath);
```

Java Methods

```
LabelStatus JPrintThermalLabel( byte [] RequestBuffer,
                                byte [] ReplyBuffer,
                                byte [] CommPort,
                                byte [] CommSpeed,
                                byte [] SavePath );

LabelStatus JPrintThermalWithNamedPrinter(byte [] PrinterName,
                                          byte [] RequestBuffer,
                                          byte [] ReplyBuffer,
                                          byte [] SavePath );
```

Where:

LabelStatus = an Integer status code
PrinterName = Name of printer in the Windows Printers folder
RequestBuffer = the shipment request
ReplyBuffer = the shipment response
CommPort = either 1, 2, 3, or 4 to indicate which serial port the printer is on
BaudRate = either 1200, 4800, 9600, or 19200 to indicate the speed to of the attached printer.
NOTE: most Eltron printers are set for 9600 only.
SavePath = the path to use when writing the label data to disk.

Saving the thermal label commands to a file

To extract the thermal label commands to a file without sending them to a printer, call the function SaveThermalLabelToFile().

Function Prototype

```
LabelStatus = SaveThermalLabelToFile(char *RequestBuffer,
                                     char *ReplyBuffer,
                                     char *SavePath
                                     char *SavedFileName);
```

OS Compatibility

WIN32, SOLARIS, LINUX

Com Interface

```
HRESULT SaveThermalLabelToFile([in] BSTR RequestBuffer,
                               [in] BSTR ReplyBuffer,
                               [in] BSTR SavePath,
                               [out, retval] BSTR* SavedFilename);
```

Java Method

```
LabelStatus JSaveThermalLabelToFile( byte [] RequestBuffer,
                                     byte [] ReplyBuffer,
                                     byte [] SavePath,
                                     byte [] SavedFileName );
```

Where:

LabelStatus = an Integer status code
RequestBuffer = the shipment request
ReplyBuffer = the shipment response
SavePath = the path to use when writing the label data to disk.
SavedFileName = the name of the file containing the label generation commands.

Displaying a SPOD letter in a browser

This function is used when there is a browser available to display and print the Signature Proof of Delivery (SPOD) letter. The scenario is that the user's application will call the DisplaySPODInBrowser() function which will launch the default browser (if it's not running) or use the browser if it is running to display the letter. The user will then click the browser's print button and print their copy of the letter.

Function Prototype

```
LabelStatus = DisplaySPODInBrowser( char *RequestBuffer,
                                    char *ReplyBuffer,
                                    char *TemplatesPath,
                                    char *SavePath);
```

OS Compatibility

WIN32

Com Interface

```
HRESULT DisplaySPODInBrowser([in] BSTR RequestBuffer,
                             [in] BSTR TemplatesPath,
                             [in[ BSTR TemplatesPath,
                             [in] BSTR SavePath);
```

Java Method

```
LabelStatus JDisplaySPODInBrowser( byte [] RequestBuffer,
                                   byte [] ReplyBuffer,
                                   byte [] TemplatesPath,
                                   byte [] SavePath );
```

Where:

LabelStatus = an Integer status code
RequestBuffer = the shipment request
ReplyBuffer = the shipment response
TemplatesPath = the path where the label extension .html files are stored (MyApp\Templates)
SavePath = the path where the label files are to be stored (MyApp\Labels)

Saving the SPOD letter image to a file

To extract the letter image and generate the HTML file and image file without displaying the letter in a browser call the function SaveSPODToFile().

Function Prototype

```
LabelStatus = SaveSPODToFile (char *RequestBuffer,
                              char *ReplyBuffer,
                              char *TemplatesPath,
                              char *SavePath,
                              char *SavedFileName );
```

OS Compatibility

WIN32, SOLARIS, LINUX

Com Interface

```
HRESULT SaveSPODToFile([in] BSTR RequestBuffer,
                       [in] BSTR ReplyBuffer,
                       [in] BSTR TemplatesPath,
                       [in] BSTR SavePath,
                       [out, retval] BSTR* SavedFilename);
```

Java Method

```
LabelStatus JSaveSPODToFile( byte [] RequestBuffer,
                             byte [] ReplyBuffer,
                             byte [] TemplatesPath,
                             byte [] SavePath,
                             byte [] SavedFileName );
```

Where:

LabelStatus = an Integer status code
RequestBuffer = the shipment request
ReplyBuffer = the shipment response
TemplatesPath = the path where the label .html files are stored (MyApp\Templates)
SavePath = the path where the label files are to be stored MyApp\Labels)
SavedFileName = the name of the HTML file generated to display the letter in a browser. The image file will have the same name with an extension of .png

Deprecated functions

These functions are still available in the Ship Label Extensions but have been deprecated.

```
int FedExLabelBrowser(char *RequestBuffer,
                      char *ReplyBuffer);

int FedExSigProofOfDeliveryBrowser(char *RequestBuffer,
                                   char *ReplyBuffer);

int FedExLabelThermal(char *RequestBuffer,
                      char *ReplyBuffer,
                      char *CommPort,
                      char *CommSpeed);

int FedExPrepareLabel(char *RequestBuffer,
                      char *ReplyBuffer,
                      char *OutBoundHTMLFilename,
                      char *CODReturnHTMLFilename);
```

Deprecated Java methods

```
int JFedExLabelBrowser( byte [] RequestBuffer,
                        byte [] ReplyBuffer )

int JFedExLabelThermal( byte [] RequestBuffer,
                        byte [] ReplyBuffer,
                        String CommPort,
                        String CommSpeed )

int JFedExPrepareLabel( byte [] RequestBuffer,
                        byte [] ReplyBuffer,
                        byte [] TrkNumHtml,
                        byte [] CODReturnHtml )

int JFedExSigProofOfDeliveryBrowser( byte [] RequestBuffer,
                                     byte [] ReplyBuffer );
```