
# Maxicode Encoder

**Version 2.1.3**

# Programmer's Manual

Silver Bay Software LLC
100 Adams Street
Dunstable, MA 01827
Phone: (800) 364-2889
Fax: (888) 315-9608
support@silverbaysoftware.com

Document Version 20091005

# Updating from a Previous Version of the Encoder

If you are updating from a Version 2.0.x encoder to the Version 2.1.x encoder, you need to make the corresponding changes in your input structures. The specific changes are:

- The `MODE-CONTROL` element has been added to the COBOL input structure.
- The `modeControl` element has been added to the C input structure.
- The `modeControl` element has been added to the Visual Basic structure.

# Table of Contents


# Maxicode Encoder

**Version 2.1.3**

# Programmer's Manual

## 1 Introduction

### 1.1 Contents of this Manual

This manual is broken into three sections:

- an introduction, ,
- a programmer's reference, and
- a printing guide.

The introduction provides a quick overview of the Maxicode symbol and a general discussion on how to program with the Silver Bay Software LLC Maxicode encoder. The programmer's reference section provides the specific details of the API's for each of the supported programming languages. Finally, the printing section provides guidelines for formatting the output of the encoder in a variety of print environments, including AFP, Metacode, and HP-PCL.

### 1.2 Maxicode Symbology Overview

Maxicode is a medium capacity, two-dimensional barcode symbology especially designed for the high-speed scanning application of package sorting and tracking. UPS introduced Maxicode in 1992 after underlying development dating from the late 1980s. In 1996, AIM USA standardized Maxicode in its "Uniform Symbology Specification – Maxicode."

Maxicode symbols have the following general characteristics:

- A two-dimensional array of hexagons surrounding a central, circular bull's-eye.
- A fixed symbol size, both graphically (always about 1" square) and in total data capacity (up to about 100 characters).
- The data content is broken into two "messages"; the Primary Message which contains the postal code, country code, and class of service, and the Secondary Message which contains other, supplemental information like tracking number, package weight, and shipping address.

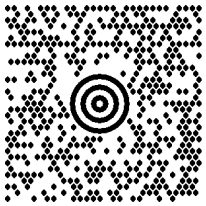


**Figure 1 - Sample Maxicode Symbol**

The fixed symbol size and circular bull's-eye are critical to high-speed scanning. The bull's-eye pattern is used as a "finder," allowing the symbol to be easily located regardless of the symbol's orientation relative to the scanner. The scale of the symbol and its hexagons are also well matched to the scanning resolution available in over-the-belt scanners. A sample Maxicode symbol is shown above in Figure 1.

The separation of the data into primary and secondary message components allows the sorting information stored in the primary message component to be recovered even in situations in which the secondary message is damaged beyond readability.

When a Maxicode symbol contains sorting information, the decoded message is in a format known as a Structured Carrier Message (SCM). The specific format of the SCM is discussed in detail in a later section.

Although primarily intended for use in package sorting environments, Maxicode can also be used as a general-purpose symbology. Thus, its use is not limited to the encoding of Structured Carrier Messages – any stream of data may be encoded in a set of Maxicode symbols. The Silver Bay Software Maxicode Encoder provides API's that allow such general purpose use in addition to API's specifically designed for sorting applications.

## 1.3 Encoder Operation

The process of converting textual data to a two-dimensional barcode is called *encoding*. This sophisticated process involves data validation, data compaction, and the insertion of error-correction information. When this document refers to the *encoder*, it is referring to the Silver Bay Software Maxicode library. While the process of encoding the shipping information is quite complex, we have developed a simple-to-use set of functions for generating Maxicode symbols.

The Silver Bay Software LLC Maxicode encoder is designed to be compatible with as many printing environments as possible. Since printing technologies, processes, and systems vary widely from computer system to computer system, the encoder does not directly print the symbol. Instead, the encoder returns a sequence of characters to the calling program. These characters correspond to specially designed *code points* in a custom font provided with the encoder. When the returned characters are rendered (i.e., printed) in this font, the result is a Maxicode symbol. It is the responsibility of the application programmer to generate the appropriate print stream data to invoke the font on the printer and to send the characters returned by the encoder to the printer. This is discussed in more detail in a later section.

### *1.3.1 Data Inputs*

In the most commonly-used API, the input into the encoder function is a simple data record. The table below briefly describes each of the elements of this input record. A more detailed discussion is provided in each of the language-specific sections of this manual.

| **Record Element** | Size | Description | Required? |
|---|---|---|---|
| Postal Code | 9 | Ship To Postal Code | Yes |
| Country Code | 3 | Ship To ISO Country Code | Yes |
| Class of Service | 3 | Class of Service | Yes |
| Tracking Number | 10 | UPS Tracking Number | Yes |
| Shipper Number | 6 | UPS Shipper Number | Yes |
| Julian Day of Pickup | 3 | Julian day of the year the package was picked-up | Yes |
| Shipment ID Number | 30 | Customer Assigned reference number | No |
| Package Number | 3 | The X in Package X of N | Yes |
| Number in Shipment | 3 | The N in Package X of N | Yes |
| Weight | 3 | Package weight, rounded up to the next pound | Yes |
| Address Validation | 1 | Address validation flag (Y or N) | Yes |
| Address | 35 | Ship To Address | No |
| City | 20 | Ship To City | Yes |
| State | 2 | Ship To State | Yes |
| Mode Control | 1 | Encoding mode control | Yes |

As mentioned earlier, there is space in a Maxicode symbol for about 100 characters of information. The fields that UPS designates as required (all of them except Shipment ID Number and Address) will use most of these characters. When using the encoder, if you attempt to supply optional data (e.g. Shipment ID Number and/or Address), you may very well exceed the capacity of a Maxicode symbol. Thus, providing any of the optional data elements presents a real problem: the symbol is not capable of storing that much data. If the Maxicode encoder is passed more data than it can encode it will return an error, and no symbol will be generated.

For this reason, it is highly recommended that you leave the optional fields blank. Most of these fields have been added for future use when two Maxicode symbols will be used to store all of the information. A separate set of API functions has been developed to handle multiple Maxicode symbols; however, they are not discussed in this document.

### *1.3.2 Encoder Output*

Due to the wide variety of printer connection methods and character translation devices in use in today's print centers, characters output by a program may not be the same characters which eventually arrive at the printer. A common example is a Xerox printer in an IBM mainframe environment. Natively, a Xerox printer is an ASCII device while IBM mainframes are EBCDIC. Somewhere along the way, the data sent from an application program to the printer will be converted from EBCDIC to ASCII. This can occur at the application program, via a converter

box, or even at the printer itself. For example, if an EBCDIC machines prints a 'zero' character, hexadecimal value F0, this will be converted to the ASCII equivalent, hexadecimal value 30, before it is printed on the Xerox printer.

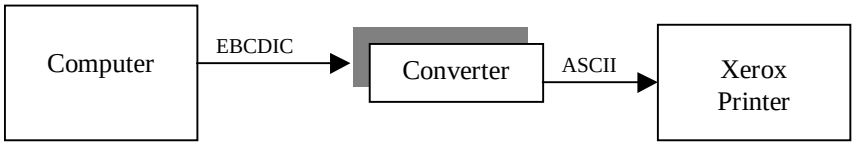



**Figure 2 - Schematic of Printing System**

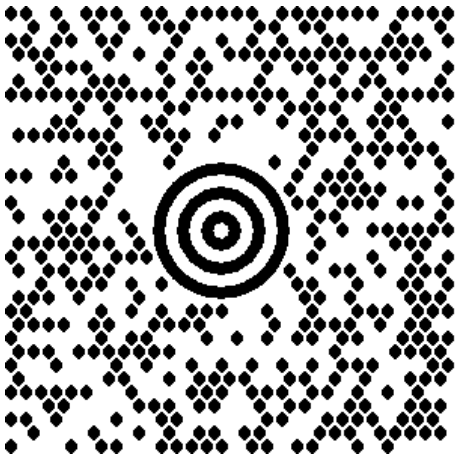
The custom Maxicode font provided with the encoder has its *code points* (or characters) located as specific hexadecimal values. This font consists of characters representing hexagons and spaces, as well as a character that prints the bull's-eye at the center of the symbol. When your application program sends the encoder's output characters to the printer, they may or may not be translated on the way to the printer. The important thing is that when they arrive at the printer they must match the font's code points. Rather than attempt to code for all possible translation scenarios in the encoder, we have instead provided a function which allows the calling program to specify which output characters to use. A table has been provided at the end of this guide that lists the most common printer configurations and the corresponding values that must be supplied to this initialization function. The exact use of this function is discussed in the language specific sections of the encoder.

The actual output of the encoder is a two-dimensional array of characters. This array is 17 lines long with each line containing 30 characters. The following is an example of what this output may look like if printed with just a normal font (i.e., not using the custom font):

```
230303230331222231233222212222
331220321012103131311331133320
103223110030320012023121302010
222232322312222323131221322222
111230121331012200212131333030
013212320222000001013300013232
222222310000000000121202201012
120120230000000000021100222032
110020103500000000003023331130
303110011100000000101030021120
002012320020000000020110003020
222132110031322003111001301210
313310020202020202020100330130
130310213121202130311302121100
233332300112003211113112130220
111031202110230033010231310310
202002202020022020000020200200
```

**Figure 3 - Sample Encoder Output, Printed with Standard Font**

However, when these same characters are printed using the custom Maxicode font, rather than 0's, 1's, 2's, 3's, and 5's (there's only one 5 characters; it will be the bullseye), you will instead get a Maxicode symbol:



***Note: Not drawn to scale***

**Figure 4 - Sample Encoder Output, Printed with Custom Font**

## 1.4 Steps to Using the Encoder

Based on all of the above, the generalized procedure for using the encoder is as follows:

1. Based on the type of printer being used, and the method via which it is connected to the computer, make the custom Maxicode font available to the printer.
2. If necessary, call the encoder initialization function with the correct character values. The values you use are based on your specific printing configuration. In most cases, this step is optional.
3. For each Maxicode symbol to be printed:
    a. Place the data to be encoded into one of the structure or record formats supported by the API functions.
    b. Call the appropriate encoder API.
    c. Check the return code to ensure that the encode operation succeeded.
    d. Send the appropriate command to invoke the Maxicode font on your printer.
    e. Send the characters returned by the encoder to the printer.
    f. Return the printer to the "normal" font.

A number of sample programs have been provided with the distribution media as well, demonstrating the use of the Maxicode encoder.

## 1.5 API's Provided

The Maxicode encoder provides several API's for generating Maxicode symbols. Each of these API's is described briefly in the table below.

| API | Description |
|---|---|
| UPS Structure | This is the easiest and most commonly used API. It is intended for environments where only standard UPS Maxicode symbols are needed. All of the required and optional information is passed to the encoder in a single record structure. The encoder takes care of all formatting and data validation.<br>NOTE: This API does not support Structured Message Append. |
| SCM String | This API accepts a properly formatted Structured Carrier Message string. The format of a SCM string is discussed in detail in a later section.<br>NOTE: This API does not support Structured Message Append. |
| SCM Structure | This API accepts Structured Carrier Message information split into primary message fields and a properly formatted Secondary Message string. The format of a Secondary Message string is discussed in detail in a later section.<br>This API supports Structured Message Append. |
| SEC Buffer | This API is used to encode general-purpose (non-SCM) information using Standard Error Correction. The information to be encoded is passed to the encoder as an array of bytes.<br>This API supports Structured Message Append. |
| EEC Buffer | This API is used to encode general-purpose (non-SCM) information using Extended Error Correction. The information to be encoded is passed to the encoder as an array of bytes.<br>This API supports Structured Message Append. |

## 1.6 Character Set Issues

The Maxicode symbology stores its internal information using the ASCII character set. While the ASCII character set is common on microcomputers and minicomputers, the EBCDIC character set is more common on IBM mid-range and mainframe systems. On EBCDIC systems, input data must be converted from EBCDIC to ASCII before it is encoded into the symbol.

The encoder library provides facilities to handle EBCDIC-to-ASCII conversions automatically, or to allow the user application to perform the conversion before passing the data to the encoder. This is done by having two versions of most of the API functions-–one which assumes that the data has already been converted to ASCII, and one that assumes it is still in the computer's "native" (EBCDIC or ASCII) character set. On EBCDIC machines, the "native" character set functions automatically perform an EBCDIC-to-ASCII conversion on all textual data as part of the encoding process. On ASCII machines, the "native" functions assume the data is already in ASCII and perform no conversion.

# 2 Maxicode Symbology Technical Details

This section of the document is intended for users who want or need more information on the internal details of a Maxicode symbol. The encoder provides a set of functions for UPS-specific applications that handle all these internal details for you. If you are using these functions, you do not need to read this section. If, on the other hand, you need more control over the symbols generated, and thus need to use some of the other interface functions, you will need the information which follows.

Note that the definitive reference for technical details of the Maxicode symbology is the *AIM USA Technical Specification – Uniform Symbology Specification – Maxicode*. This specification is available through

AIM USA
634 Alpha Drive
Pittsburgh, PA 15238 USA
Phone: (412) 963-8588
E-mail: tech @ aimusa.org
Web: http://www.aimusa.org

## 2.1 Physical Structure

In a single Maxicode symbol, a total of 886 hexagons are arranged around the bull's-eye finder in a hexagonal grid 33 rows high. These hexagons are referred to as "modules." Odd-numbered rows of the symbol contain 30 modules, while even-numbered rows contain 29 modules, offset by a half module to form a hexagonal array. Of these 886 hexagons, the 2 in the upper-right corner are always black, 18 located around the bull's-eye are used to help determine the orientation of the symbol, and the remaining 864 encode data at one bit per hexagon. Black modules encode a binary "1" and white modules (those left unprinted) encode a binary "0". The 864 data modules are grouped into 144 6-bit symbol characters, also referred to as "codewords." Some of these codewords are used to represent encoded data, while others are error correction codewords. The error correction codewords are included in the symbol to compensate for the fact that barcodes are frequently damaged during handling. Through a mathematical technique known as Reed-Solomon Error Correction, the barcode scanner uses this extra information to reconstruct missing or damaged portions of the Maxicode symbol so that the encoded data can be recovered. The precise number of data codewords versus error correction codewords in a symbol depends on the symbol's "mode." This will be covered shortly.

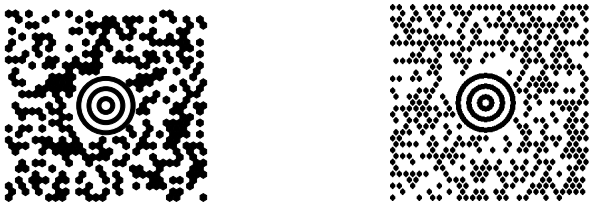
Characters are encoded into the barcode using a sequence of codeword values. A Maxicode symbol can encode any 8-bit value into the body of the symbol. This includes all the ASCII characters, as well as non-printable binary values. Each 6-bit codeword in the symbol encodes either a specific character, or a control character that changes how subsequent codewords are interpreted. This allows the encoding of 8-bit values into a stream of 6-bit codewords.

Because of the fixed size of a Maxicode symbol, there is a fixed upper limit to the number of characters that can be encoded in a single symbol. This value ranges from 60-144 characters, depending on the symbol mode, and the precise character sequence. Character sequences

consisting of similar characters (e.g. all upper case alphas and digits) require fewer control codewords, and thus allow more characters to be encoded in a symbol.

## 2.2 Historical Overview

Figure 5 shows two representative Maxicode symbols, one on the left as it looked when initially introduced to the public and one on the right as it looks today. During the process of standardization through the AIM USA Technical Symbology Committee (TSC), a number of modifications were made to the preferred Maxicode implementation. These changes were made to improve the readability of the symbology, and to improve its tolerance for damage to the printed image. The changes introduced in the AIM USS are backward compatible, allowing readers to work with either symbol type, and to auto-discriminate between them.



**(circa 1992)** **(circa 1996)**

**Figure 5 - Representative Maxicode symbols**

Important changes in the newer USS-Maxicode specification include:

- Shrinking the size of the dark data "modules" so they no longer touch their neighbors.
- Changes in the layout of the secondary message and the way its error correction is performed.
- Expanded encoding capabilities to allow any 8-bit value to be encoded as part of the message.
- A multi-symbol "Structured Message Append" capability, allowing a message that will not fit in a single symbol to be continued in one or more additional symbols.

Although many Maxicode systems and printers in use today follow the original specification, new printing equipment and installations should adhere to the more robust and flexible symbol design of USS-Maxicode. As such, the Silver Bay Software Maxicode encoder only supports the generation of the newer format. The encoder supports all the features of the new specification. In particular, Structured Message Append support is available, although this requires some extra steps on the part of the programmer to achieve.

## 2.3 Internal Encoding Details

This section of the document provides a high-level overview of the internal structure of a Maxicode symbol. Only the "new" format symbols are described herein. For information on "old" format symbols, AIM USA also publishes a document entitled "Guideline on Mode 0 for Maxicode."

As mentioned earlier, a Maxicode symbol contains two message components: a primary message component and a secondary message component. These two components are stored in different

parts of the symbol, and each has error correction information added to it, allowing recovery of the data even when the physical symbol is of poor quality or is damaged.

A Maxicode symbol's primary message component encodes 60 bits of information, and serves two special uses. First, it contains four bits that directly indicate the "mode" in which all the rest of the symbol is encoded.

The modes that are currently defined are indicated in the following table:

| Mode | Use |
|---|---|
| 0 | Obsolete. Indicates an "old type" symbol containing a Structured Carrier Message. This mode has been superceded by Modes 2 and 3. |
| 1 | Obsolete. Indicates an "old type" symbol containing general-purpose data. This mode has been superceded by Mode 4. |
| 2 | Indicates the symbol contains a Structured Carrier Message with a numeric postal code. |
| 3 | Indicates the symbol contains a Structured Carrier Message with an alphanumeric postal code. |
| 4 | Indicates the symbol contains general-purpose data, protected by the standard error correction |
| 5 | Indicates the symbol contains general-purpose data, protected by enhanced error correction. |
| 6 | Reserved for symbols used to program internal parameters of Maxicode readers. |

The use of a consistent mode numbering system across both "old" and "new" symbols allows readers to determine what type of symbol is being scanned. The encoder supports Modes 2, 3, 4 and 5. Modes 0 and 1 are not supported because they are obsolete, while Mode 6 is only used by manufacturers of barcode scanners.

For modes representing Structured Carrier Messages (modes 0, 2 and 3), the remaining 56 bits of the primary message contains all the information needed for package sorting:

- The destination country code,
- The "ship-to" postal code (zip code), and
- The class of service.

This information normally requires up to 15 characters to represent, but is encoded using an efficient binary compaction technique (different from the codeword system used to encode the secondary message) to allow it to fit in 56 bits. Thus, in most cases, high-speed sorting systems only need to decode the primary message in order to properly route a package. This represents a considerable efficiency improvement in such applications. In the Structured Carrier Message modes, the secondary message component contains additional information about the package such as the tracking number, package weight, etc. that, while important, is not required for package routing.

In the general-purpose data modes (modes 4 and 5), the primary message is still physically separated from the secondary message in the symbol. For data content purposes, however, they are simply considered two portions of a single data stream. In this case, 54 of the 56 bits of the primary message are treated as nine 6-bit codewords, and the remaining 2 bits are unused. The nine primary message codewords are combined with the secondary message codewords, and the entire set is decoded as a unit.

Because of the importance of the primary message component to package sorting, the 60-bit primary message is protected by 60 bits of error correction information. This allows recovery of the primary message information even if up to 25% of the hexagons representing the primary message are damaged. For Modes 2, 3 and 4, the secondary message consists of 504 data bits (84 codewords) and 240 bits of error correction. For Mode 5, the amount of error correction in the secondary message is increased to 336 bits, and the data portion reduced to 408 bits (68 codewords). This allows greater protection against damage, but at the cost of lower data content per symbol.

Maxicode also includes a capability referred to as "Structured Message Append." Using this feature, a message that is too long to fit in a single symbol can be continued across one or more additional symbols. Special coding at the beginning of the secondary message indicates whether this is an "isolated" symbol or whether it is part of a Structured Message Append set. The encoder supports the generation of symbols using this feature; however this requires the use of the more advanced interface functions.

## 2.4 Maxicode Encoding Modes

The Maxicode symbology provides support for both numeric postal codes, such as are used in the United States, and mixed alphanumeric postal codes, such as are used in Canada. Internally, the symbology uses two different "modes" to represent these two different types of postal codes. "Mode 2" is capable of encoding postal codes up to 9 digits in length, but is only capable of encoding all-numeric postal codes. "Mode 3" is capable of encoding alphanumeric postal codes, but will only encode up to 6 characters.

In versions of the encoder prior to Version 2.1.x, the encoder always internally determined the mode to use. In order to adapt to changes in how different countries represent their postal codes, and in how UPS would like to see different postal codes encoded, beginning in Version 2.1.1, the encoder now provides a means via which the programmer can exercise control over the encoding mode used.

The encoder now offers four options: the original AIM algorithm, a revised UPS algorithm, Mode 2 only, and Mode 3 only. These are described below:

AIM Algorithm: This is the algorithm that is described in the AIM specification for the Maxicode symbology. It uses Mode 2 for any all-numeric postal code, regardless of the length, and uses Mode 3 for any postal code that contains one or more alphabetic characters. Prior to version 2.1.x, this was the algorithm implemented internally by the encoder. Thus, this mode should be selected by users who are upgrading from an earlier version of the encoder, unless the previous encoder was not producing the desired results.

UPS Algorithm: This is a revised version of the auto-detection algorithm that UPS seems to prefer. In this algorithm, only 5-digit or 9-digit postal codes are encoded using Mode 2. Mode 3 is used for all postal codes that contain one or more alphabetic characters, as well as all-numeric postal codes that are not 5 or 9 digits in length.

| | |
|---|---|
| Mode 2 only: | This forces the encoder to try to use Mode 2 to encode the postal code. If the postal code contains alphabetic characters and "Mode 2 only" is selected, the encoder will return an error. |
| Mode 3 only: | This forces the encoder to use Mode 3 to encode the postal code. |

The encoder always accepts postal codes up to nine characters in length. If the encoder is instructed to use Mode 3, or chooses Mode 3 as a result of one of the auto-detecting algorithms, the postal code will be automatically truncated to a maximum of 6 characters. As a result, "Mode 3 only" should never be used to encode symbols for United States destinations, because this will cause the 9-digit postal code to be truncated to 6 digits.

## 2.5 Maxicode in United Parcel Service (UPS) Applications

UPS has specific requirements of their customers for the content of Maxicode symbols. This section of the document summarizes these requirements, as well as how the information is encoded into the primary and secondary message portions of a Maxicode symbol. For more information on message content in UPS applications, consult *Guide to Bar Coding with UPS*, a publication describing UPS's bar coding standards and requirements. This document is available through your local UPS Account Executive.

### *2.5.1 Structured Carrier Message Format*

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

The ANSI standard message format consists of a message header, message body, and message terminator.

The message header consists of two parts:

- The four-character sequence `[)>`$^{R_S}$ where $^{R_S}$ represents the ASCII "Record Separator" character. These characters identify the message as belonging to the ANSI standard.
- The five-character sequence `01`$^{G_S}$`96` where $^{G_S}$ represents the ASCII "Group Separator" character. This is the standard Transportation Data Format Header.

The message body consists of a list of data fields (discussed below) in a particular order. The $^{G_S}$ character is used to separate individual fields.

The message terminator consists of the two-character sequence $^{R_S}{}^{E_{O_T}}$ where $^{R_S}$ indicates the ASCII "Record Separator" character and $^{E_{O_T}}$ indicates the ASCII "End Of Transmission" character.

The hexadecimal values for all three special characters are as follows. Both the EBCDIC and ASCII values have been provided:

| Character | EBCDIC Value | ASCII Value |
|---|---|---|
| $^{R}_{S}$ | 0x1E | 0x1E |
| $^{G}_{S}$ | 0x1D | 0x1D |
| $^{E}O_{T}$ | 0x37 | 0x04 |

For UPS applications, the following data elements are mandatory:

| Item | Size and Type |
|---|---|
| Ship-To Postal Code | 5 or 9 digits in the USA, up to 6 alphanumeric characters in other countries. |
| Ship-To Country Code | 3 digits (840 for USA) |
| Class of Service | 3 digits |
| Tracking Number | 10-character alphanumeric |
| UPS Standard Carrier Alpha Code | "UPSN" |

The following data elements are optional:

| Item | Size and Type |
|---|---|
| UPS Shipper Number | 6-character alphanumeric |
| Julian Day of Pickup | 3 digits |
| Shipment ID Number | 1-30 character alphanumeric |
| Package In Shipment (package N of X total packages) | 1-4 digits "/" 1-4 digits |
| Weight in pounds | 1-5 digits |
| Address Validation | "Y" or "N" |
| Ship-To Address | 1-35 alphanumeric |
| Ship-To City | 1-35 alphanumeric |
| Ship-To State | 2-character alpha |

**NOTE:** If all the information listed in these tables is provided for an individual package, it is quite possible that the data will not fit into a single symbol. After encoding the message header, mandatory information and the message terminator, a Maxicode symbol only has space for a maximum of 53 characters of optional information. This includes the $^{G}_{S}$ characters separating the fields. As a result, a single symbol can hold only a maximum of 43 characters of optional information.

A typical minimum data string, including only the mandatory elements, would be

[)>$^{R}_{S}$01$^{G}_{S}$97123456789$^{G}_{S}$840$^{G}_{S}$001$^{G}_{S}$1Z12345678$^{G}_{S}$UPSN$^{R}_{S}$$^{E}O_{T}$

while a full example might be

[)>$^{R}_{S}$01$^{G}_{S}$96123456789$^{G}_{S}$840$^{G}_{S}$001$^{G}_{S}$1Z12345678$^{G}_{S}$UPSN$^{G}_{S}$06X610$^{G}_{S}$159$^{G}_{S}$
1234567$^{G}_{S}$1/2$^{G}_{S}$3.1$^{G}_{S}$Y$^{G}_{S}$634 MAIN ST$^{G}_{S}$YORK$^{G}_{S}$PA$^{R}_{S}$$^{E}O_{T}$

In these messages, the elements are:

| Item | Meaning |
|---|---|
| [)>$^{R}_{S}$01$^{G}_{S}$96 | Standard prefix |
| 123456789 | Nine-digit Ship-To postal code |
| 840 | Three-digit Ship-To country code |
| 001 | Three-digit Class of Service |
| 1Z12345678 | Ten-character Tracking Number |
| UPSN | Four-Character Standard Carrier Alpha Code for UPS |
| 06X610 | UPS Shipper Number |
| 159 | Julian Day of Pickup |
| 1234567 | Shipment ID Number |
| 1/2 | Package In Shipment (package 1 of 2 total packages) |
| 3.1 | Weight (3.1 pounds) |
| Y | Address Validation |
| 634 MAIN ST | Ship-To Address |
| YORK | Ship-To City |
| PA | Ship-To State |

Note that if a field is omitted, the $^{G}_{S}$ character terminating the field must be preserved. Thus, in the second example, if the Package In Shipment and Weight fields were omitted, the latter portion of the message would be changed from

…1234567$^{G}_{S}$1/2$^{G}_{S}$3$^{G}_{S}$Y$^{G}_{S}$634 MAIN ST$^{G}_{S}$…

to

…1234567$^{G}_{S}$$^{G}_{S}$$^{G}_{S}$Y$^{G}_{S}$634 MAIN ST$^{G}_{S}$…

where the adjacent $^{G}_{S}$ characters indicate the position of blank fields.

Again, note that the lengths defined earlier represent the maximum supported length of each individual field. If all fields are filled to their maximum length, the resulting data stream will not fit within a single symbol. The encoder library will return an error if the data passed to it will not fit in a single symbol. Certain API functions of the encoder support the Structured Message Append feature of Maxicode, which allows a long message to be split across two or more symbols. This is covered later in the document.

### *2.5.2 Primary and Secondary Message Formats*

The Maxicode encoder provides UPS-specific functions that eliminate the need to understand the internal formatting of a Maxicode barcode for UPS use. These functions simply accept the individual UPS mandatory and optional parameters and perform all the required formatting internally. In certain specialized applications, however, these functions may not have all the capability required. For example, in order to generate a set of Structured Message Append symbols to represent a long message, the more powerful SCM functions, rather than the UPS-specific functions, must be used. Use of these functions requires an understanding of how a

message is broken down into primary and secondary message components, since the SCM functions require the programmer to "manually" format the secondary message component.

UPS Maxicode barcodes are Structured Carrier Messages (SCM), and as such, use either Mode 2 or Mode 3 symbols. Mode 2 applies when the Ship-To Postal Code is all numeric, as it is in the USA. Mode 3 is reserved for international applications in which the Ship-To Postal Code is alphanumeric (e.g. Canada). In either type of symbol, the Ship-To Postal Code, the Country Code and the Service Class are encoded in the primary message component of the barcode, while the remaining elements are encoded in the secondary message component. This requires that the SCM message header (`[)>`$^{R}_{S}$`01`$^{G}_{S}$`96`) and the remaining elements that appear after the primary message, be grouped together into the secondary message.

The specific rules for formatting the secondary message from a Structured Carrier Message are:

1. The first nine data characters `[)>`$^{R}_{S}$`01`$^{G}_{S}$`96` are extracted to be encoded in the secondary message.
2. The next three data elements, representing respectively the postal code, country code, and service class and their separators ($^{G}_{S}$) are dropped from the data source (since these fields are encoded in the primary message)
3. The remaining string of data is then encoded in the secondary message after the header "`[)>`$^{R}_{S}$`01`$^{G}_{S}$`96`".

If we return to the message example given earlier:

`[)>`$^{R}_{S}$`01`$^{G}_{S}$`96`**`123456789`**$^{G}_{S}$**`840`**$^{G}_{S}$**`001`**$^{G}_{S}$`1Z12345678`$^{G}_{S}$`UPSN`$^{R}_{S}$$^{E}$`0`$_{T}$

The Ship-To Postal Code (`123456789`), the Country Code (`840`), and the Class of Service (`001`), shown underlined, are encoded in the primary message. These fields, along with their terminating $^{G}_{S}$ characters, are removed from the body of the message, and the remaining data

`[)>`$^{R}_{S}$`01`$^{G}_{S}$`961Z12345678`$^{G}_{S}$`UPSN`$^{R}_{S}$$^{E}$`0`$_{T}$

is encoded as the secondary message. In the second example given earlier,

`[)>`$^{R}_{S}$`01`$^{G}_{S}$`96`**`123456789`**$^{G}_{S}$**`840`**$^{G}_{S}$**`001`**$^{G}_{S}$`1Z12345678`$^{G}_{S}$`UPSN`$^{G}_{S}$`06X610`$^{G}_{S}$`159`$^{G}_{S}$`1234567`$^{G}_{S}$$^{G}_{S}$`3.1`$^{G}_{S}$`Y`$^{G}_{S}$`634 MAIN ST`$^{G}_{S}$`YORK`$^{G}_{S}$`PA`$^{R}_{S}$$^{E}$`0`$_{T}$

the encoded secondary message is

`[)>`$^{R}_{S}$`01`$^{G}_{S}$`961Z12345678`$^{G}_{S}$`UPSN`$^{G}_{S}$`06X610`$^{G}_{S}$`159`$^{G}_{S}$`1234567`$^{G}_{S}$$^{G}_{S}$`3.1`$^{G}_{S}$`Y`$^{G}_{S}$`634 MAIN ST`$^{G}_{S}$`YORK`$^{G}_{S}$`PA`$^{R}_{S}$$^{E}$`0`$_{T}$

Thus, an alternate expression of the encoding rule for the secondary message is to write out the entire formatted Structured Carrier Message, then delete the postal code, country code, service class and the corresponding three $^{G}_{S}$ characters. The string that remains is placed in the secondary message portion of the symbol.

### *2.5.3 Compressed Maxicode Format*

In late 2001, UPS introduced a new "compressed" form of Maxicode. The goal of this new form was to allow more information to be encoded within a single symbol. Prior to the introduction of compression, a full address, including street address, department, etc. would typically result in more data than a single Maxicode symbol could handle. By using some proprietary compression techniques, UPS was able to squeeze this type of information into the symbol as well.

Because the compression algorithms are proprietary to UPS, the Silver Bay Software encoder cannot directly generate a compressed Maxicode symbol. It is possible, however, to use the UPS-provided DLL to perform the data compression operation, and then pass the compressed data into the encoder to perform the remaining operations necessary to reduce the data to a symbol. At present, this operation is only supported via the Microsoft Windows version of the encoder.

# 3 Using the Encoder for UPS Applications

The Maxicode encoder provides two sets of API's. The simpler API is designed specifically for use in UPS shipping applications, and is described in this chapter. A more sophisticated API is also available for more complex application. This API is discussed in the following chapter.

## 3.1 COBOL Language API

### *3.1.1 Initializing the Encoder: MAXINIT*

In some rare cases, it may be necessary to change the default characters the Maxicode encoder generates. The most notable case is when printing from an EBCDIC system to a Xerox printer using Metacode. By default, on an EBCDIC system (e.g., S/370 or AS/400), the Maxicode encoder will generate EBCDIC characters. However, a Xerox printer is an ASCII device. Therefore, when the application program is sending Metacode to the Xerox printer, it must first convert its output to ASCII. While the application programmer could allow the encoder to generate EBCDIC characters and then subsequently convert them to ASCII, this function allows the application programmer to override the default characters generated by the encoder. Thus, this would allow the encoder to directly generate the characters in ASCII.

**Call syntax:**

```
CALL "MAXINIT" USING MAXICODE-FONT-INFO-REC.
```

**Parameters:**

```
01  MAXICODE-FONT-INFO-REC.
    05  NO-HEX-CHAR          PIC X.
    05  LO-HEX-CHAR          PIC X.
    05  HI-HEX-CHAR          PIC X.
    05  TWO-HEX-CHAR         PIC X.
    05  PAD-CHAR             PIC X.
    05  BULL-CHAR            PIC X.
```

**Notes:**

- This function does not need to be called before every symbol encoded. It only needs to be called once as part of program initialization.
- If this function is not called, the encoder will still operate, using default values (the characters '0' through '5'). In most cases, these default values will be correct.
- Note the use of double quotes in the CALL statement around the procedure name. Some platforms may require single quotes.
- If you are using the encoder as an AS/400 ILE service program, remember to include LINKAGE TYPE IS PROCEDURE in the call statement.

**Example:**

```
 WORKING STORAGE SECTION.
 …

*  Values for Xerox printer, programs sends METACODE;
*  thus we need encoder to generate ASCII values and
*  not EBCDIC.

 01  MAXICODE-FONT-INFO-REC.
     05  NO-HEX-CHAR          PIC X VALUE X'30'.
     05  LO-HEX-CHAR          PIC X VALUE X'31'.
     05  HI-HEX-CHAR          PIC X VALUE X'32'.
     05  TWO-HEX-CHAR         PIC X VALUE X'33'.
     05  PAD-CHAR             PIC X VALUE X'34'.
     05  BULL-CHAR            PIC X VALUE X'35'.

 PROCEDURE DIVISION.

 0000-INITIALIZATION.
 . . .
     CALL "MAXINIT" USING MAXICODE-FONT-INFO-REC.
```

### *3.1.2 Calling the Encoder: MAXUPSN*

The actual message encoded in a Maxicode symbol conforms to a format known as a Structured Carrier Message. This format is very similar to an EDI record, using special headers, trailers, field separators, and variable length fields. Rather than require you to code to this arcane format, the Maxicode encoder library provides a much higher level API.

The COBOL API procedure for encoding a UPS symbol is MAXUPSN. The procedure takes two parameters; an input record and an output record. The structure of these records must match the definitions shown in the **Parameters** section below.

**Call Statement:**

```
CALL "MAXUPSN" USING MAXICODE-INPUT-REC
                     MAXICODE-OUTPUT-REC.
```

**Parameters:**

```
01 MAXICODE-INPUT-REC.
   05 POSTAL-CODE           PIC X(9).
   05 COUNTRY-CODE          PIC 9(3).
   05 SERVICE-CLASS         PIC 9(3).
   05 TRACKING-NUMBER       PIC x(10).
   05 SHIPPER-NUMBER        PIC X(6).
   05 JULIAN-DAY-OF-PICKUP  PIC 9(3).
   05 SHIPMENT-ID           PIC X(30) VALUE SPACES.
   05 PACKAGE-NUMBER        PIC 9(4).
   05 PACKAGE-COUNT         PIC 9(4).
   05 PACKAGE-WEIGHT        PIC 9(5).
   05 ADDRESS-VALIDATION    PIC X(1).
   05 SHIP-TO-ADDRESS       PIC X(35) VALUE SPACES.
   05 SHIP-TO-CITY          PIC X(20).
   05 FILLER                PIC X(15) VALUE SPACES.
   05 SHIP-TO-STATE         PIC X(2).
   05 MODE-CONTROL          PIC X(1)

01 MAXICODE-OUTPUT-REC.
   05 OUTPUT-LINES          PIC X(30) OCCURS 17.
   05 RESULT-CODE           PIC 9(3).
```

All fields in the input record (`MAXICODE-UPS-INPUT-REC`) must contain either a valid value or be initialized to spaces (for character fields) or zeros (for numeric fields). The following table discusses each data element in detail:

| Element | Max Len | Contents |
|---|---|---|
| `POSTAL-CODE` | 9 | **Required:** the destination zip code/postal code. This field must contain one of the following:<br>A 9 digit numeric, US Zip Code<br>A 1-6 digit alphanumeric, International Postal Code (letters must be upper case), padded with spaces (to 9 characters).<br>If COUNTRY-CODE is 840 (USA), this field must contain a 9-digit US Zip code. If the last 4 digits of the zip code are not known, use zeros. |
| `COUNTRY-CODE` | 3 | **Required:** the destination country code (USA = 840). A valid country code between 000 and 999 must be specified. |
| `SERVICE-CODE` | 3 | **Required:** the UPS class of service for the package. Valid service codes include:<br>NEXT DAY AIR 001<br>2ND DAY AIR 002<br>3 DAY SELECT 012<br>GROUNDTRAC 003<br>DELIVERYTRAC II 005<br>DELIVERYTRAC 008<br>Consult your UPS documentation for more information and a complete list of service codes. |
| `TRACKING-NUMBER` | 10 | **Required:** the UPS Tracking Number (composed of the two-character data identifier, seven-digit reference number, and one digit check digit). |
| `SHIPPER-NUMBER` | 6 | **Required:** your UPS-assigned Shipper Number. |
| `JULIAN-DAY-OF-PICKUP` | 3 | **Required:** the numeric day of the year in which the package was picked up (Jan 1 = 001, Jan 31 = 031, Feb 1 = 032, Feb 28 = 057, etc.).<br>If not known, this field must be filled with zeros. |
| `SHIPMENT-ID` | 30 | **Optional:** the shipper-assigned identification number for the shipment. If the value is less than 30 characters long, pad with spaces.<br>If not specified, this field must be filled with spaces.<br>It is highly recommended that this field **always** be left blank; otherwise, the message may be too long to encode. |
| `PACKAGE-NUMBER` | 4 | **Required:** the "N" in "Package N of X"<br>This field is 4 digits long for backward compatibility with older versions of the encoder. The valid range is 0 through 999. |
| `PACKAGE-COUNT` | 4 | **Required:** the "X" in "Package N of X"<br>This field is 4 digits long for backward compatibility with older versions of the encoder. The valid range is 0 through 999. |

| Element | Max Len | Contents |
|---|---|---|
| PACKAGE-WEIGHT | 5 | **Required:** the weight of the package in pounds. Weight should be rounded up to the next pound.<br>If not known, this field must be filled with zeros.<br>This field is 5 digits long for backward compatibility with older versions of the encoder. The valid range is 0 through 999. |
| ADDRESS-VALIDATION | 1 | **Required:** indicates that the address information provided comes from a certified data source. Valid values are 'Y' or 'N' |
| SHIP-TO-ADDRESS | 35 | **Optional:** the destination street address. If the value is less than 35 characters long, pad with spaces.<br>If not specified, this field must be filled with spaces.<br>It is highly recommended that this field **always** be left blank; otherwise, the message may be too long to encode. |
| SHIP-TO-CITY | 20 | **Required:** the destination city. If the value is less than 20 characters long, pad with spaces. |
| FILLER | 15 | **Required:** reserved: value **must** be spaces. |
| SHIP-TO-STATE | 2 | **Required:** the destination state abbreviation. |
| MODE-CONTROL | 1 | **Required:** Encoder mode control<br>*This field is new in the Version 2.1.x encoder.* It has the following defined values:<br>'0' Use the AIM-standard algorithm to automatically determine the encoding mode.<br>'1' Use the alternate UPS algorithm to automatically determine the encoding mode.<br>'2' Force the encoder to use Mode 2 to encode the symbol.<br>'3' Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

The output record (MAXICODE-OUTPUT-REC) has two fields; the output character table and a result code:

| Element | Max Len | Contents |
|---|---|---|
| OUTPUT-LINES | 30 x 17 | If the encoder can successfully encode the input data, this table will contain the font characters which, when printed using the provided font, will create the Maxicode symbol.<br>NOTE: on some platforms, it may be simpler to represent this data element as 17 individual elements (like in an AS/400 DDS). |
| RESULT-CODE | 3 | The encoder returns its status in this field. If the encoder was successful, 000 is returned; otherwise an error code is returned (see next table). |

The following values are defined for the `RESULT-CODE` field:

| Value | Meaning |
|---|---|
| 000 | Success |
| 001 | An invalid parameter was passed. This typically indicates that a NULL pointer was passed to the encoder. This error does not normally occur with COBOL programs. |
| 002 | The data passed will not fit in a single Maxicode symbol; too many of the optional fields have been populated. |
| 003 | POSTAL-CODE is invalid. Possible causes include:<br>• POSTAL-CODE contains characters other than digits and upper-case alphabetic characters.<br>• COUNTRY-CODE is 840 and POSTAL-CODE is not a 5-digit or 9-digit Zip Code.<br>• MODE-CONTROL is set to '2', and the postal code contains letters. |
| 004 | COUNTRY-CODE is invalid. This occurs if COUNTRY-CODE is not numeric. |
| 005 | SERVICE-CLASS is invalid. This occurs if SERVICE-CLASS is not numeric. |
| 006 | TRACKING-NUMBER is not 10 characters. |
| 007 | ADDRESS-VALIDATION is invalid. Valid values are 'Y', 'N', or space. |
| 008 | SHIP-TO-STATE is invalid. The field must be two characters in length or all blanks. |
| 009 | The PACKAGE-NUMBER and PACKAGE-COUNT fields are incorrect or inconsistent. Possible causes include:<br>• One of the fields is zero and the other is not<br>• One or both of the fields are not numeric<br>• PACKAGE-NUMBER is greater than PACKAGE-COUNT |
| 011 | PACKAGE-WEIGHT is invalid. Possible causes include:<br>• The field is not numeric<br>• The field has not been formatted correctly (must be in the range of 0 through 999) |

In addition to these values, a number of "internal" error codes are defined. All these errors have a value of 100 or greater. Should you encounter an internal error code, please note the code and contact Silver Bay Software technical support.

**Notes:**

- A Maxicode symbol can only hold about 100 characters; if you populate any of the "optional" data elements of the input record, the encoder may not be able to generate a symbol and will return a result code of 002.
- The degree to which the encoder API validates postal codes, country codes and service classes is limited to ensuring that the data passed can be encoded into a Maxicode label (i.e. is the correct length, and is of the correct type – digits or alphanumeric). The encoder has no way to ensure that the actual data is valid.
- Be sure to always initialize the entire input record; failure to do so could result in "garbage" characters being encoded, or even encoder errors (002, message too long to encode for example).

- Note the use of double quotes in the CALL statement around the procedure name. Some platforms may require single quotes.
- If you are using the encoder as an AS/400 ILE service program, remember to include LINKAGE TYPE IS PROCEDURE in the call statement (refer to the sample ILE COBOL program).

**Example:**

```
WORKING STORAGE SECTION.
 …

 01 MAXICODE-INPUT-REC.
    05 POSTAL-CODE           PIC X(9).
    05 COUNTRY-CODE          PIC 9(3).
    05 SERVICE-CLASS         PIC 9(3).
    05 TRACKING-NUMBER       PIC X(10).
    05 SHIPPER-NUMBER        PIC X(6).
    05 JULIAN-DAY-OF-PICKUP  PIC 9(3).
    05 SHIPMENT-ID           PIC X(30) VALUE SPACES.
    05 PACKAGE-NUMBER        PIC 9(4).
    05 PACKAGE-COUNT         PIC 9(4).
    05 PACKAGE-WEIGHT        PIC 9(5).
    05 ADDRESS-VALIDATION    PIC X(1).
    05 SHIP-TO-ADDRESS       PIC X(35) VALUE SPACES.
    05 SHIP-TO-CITY          PIC X(20).
    05 FILLER                PIC X(15) VALUE SPACES.
    05 SHIP-TO-STATE         PIC X(2).
    05 MODE-CONTROL          PIC X(1) VALUE '0'

 01 MAXICODE-OUTPUT-REC.
    05 OUTPUT-LINES          PIC X(30) OCCURS 17.
    05 RESULT-CODE           PIC 9(3).
 …

 PROCEDURE DIVISION.

 1100-MAXICODE-ENCODER.
 …

*   Make sure all fields are blank or zero
    INITIALIZE MAXICODE-INPUT-REC.
```

```
*   Populate required fields
    MOVE "841706672"         TO POSTAL-CODE.
    MOVE 840                 TO COUNTRY-CODE.
    MOVE 003                 TO SERVICE-CLASS.
    MOVE "1Z12345675"        TO TRACKING-NUMBER.
    MOVE "12345E"            TO SHIPPER-NUMBER.
    MOVE 89                  TO JULIAN-DAY-OF-PICKUP.
    MOVE 1                   TO PACKAGE-NUMBER.
    MOVE 1                   TO PACKAGE-COUNT.
    MOVE 10                  TO PACKAGE-WEIGHT.
    MOVE "Y"                 TO ADDRESS-VALIDATION.
    MOVE "SALT LAKE CITY"    TO SHIP-TO-CITY.
    MOVE "UT"                TO SHIP-TO-STATE.

    CALL "MAXUPSN" USING MAXICODE-INPUT-REC
                         MAXICODE-OUTPUT-REC.
```

## 3.2 C Language API

### *3.2.1 Initializing the Encoder: MaxInitC*

In some rare cases, it may be necessary to change the default characters the Maxicode encoder generates. The most notable case is when printing from an EBCDIC system to a Xerox printer using Metacode. By default, on an EBCDIC system (e.g., S/370 or AS/400), the Maxicode encoder will generate EBCDIC characters. However, a Xerox printer is an ASCII device. Therefore, when the application program is sending Metacode to the Xerox printer, it must first convert its output to ASCII. While the application programmer could allow the encoder to generate EBCDIC characters and then subsequently convert them to ASCII, this function allows the application programmer to override the default characters generated by the encoder. Thus, this would allow the encoder to directly generate the characters in ASCII.

**Prototype:**

```
#include "maxiapi.h"

void MaxInitC(MAXIINITCPTR pCharSet);
```

**Arguments:**

`pCharSet – pointer to an sMaxiFontInit structure (defined in maxiapi.h)`

**Notes:**

- The include file `maxiapi.h` contains all of the type definitions and prototypes for the encoder API functions.
- This function does not need to be called before every symbol encoded. It only needs to be called once as part of program initialization.
- If this function is not called, the encoder will still operate, using default values (the characters '0' through '5'). In most cases, these default values will be correct.

**Example:**

```
void main(void)
{
    /*  Values for Xerox printer, we're sending metacode  */
    MAXINIT charSet = { 0x30, 0x31, 0x32, 0x33, 0x34, 0x35 };

    MaxInitC(&charSet);

    …
}
```

### *3.2.2 Calling the Encoder: MaxUpsNC*

The actual message encoded in a Maxicode symbol conforms to a format known as a Structured Carrier Message. This format is very similar to an EDI record, using special headers, trailers, field separators, and variable length fields. Rather than require you to code to this arcane format, the Maxicode encoder library provides a much higher level API.

The C API function for encoding a symbol is MaxUpsNC. The function takes two arguments; a pointer to an input structure and a pointer to an output structure. The structure of these records must match the definitions shown in the **Arguments** section.

**Prototype:**

```
#include "maxiapi.h"

void MaxUpsNC(MAXIUPSINFOCPTR pInput,
              MAXIFONTOUTPTR  pOutput);
```

**Arguments:**

```
pInput – pointer to an sMaxicodeUpsInfo structure (defined in maxiapi.h)
pOutput – pointer to an sMaxicodeFontOutput structure (defined in maxiapi.h)
```

All fields in the input structure (`pInput`) must contain either a valid value or be initialized to an empty string (for character fields) or zero (for numeric fields). An `sMaxicodeUpsInfo` structure is defined as follows:

```
struct sMaxicodeUpsInfo
{
    unsigned long   countryCode;
    unsigned long   serviceClass;
    unsigned long   julianDayOfPickup;
    unsigned long   packageNumber;
    unsigned long   packageCount;
    unsigned long   packageWeight;
    char            postalCode[12];
    char            trackingNumber[12];
    char            shipperNumber[8];
    char            shipmentID[32];
    char            shipToAddress[36];
    char            shipToCity[36];
    char            shipToState[4];
    char            addressValidation;
    char            modeControl;
    char            pad[2];
};
```

Note that all of the character field widths have been adjusted so each element lands on a double word (4 byte) boundary. This was done to ensure maximum portability; the extra bytes are ignored by the encoder. Refer to the table below for the valid data length of each character field:

| Element | Max Len | Contents |
|---|---|---|
| `postalCode` | 9 | **Required:** the destination zip code/postal code represented as a NUL terminated string. This field must contain one of the following:<br>A 9 digit numeric, US Zip Code<br>A 1-6 digit alphanumeric, International Postal Code (letters must be upper case).<br>If COUNTRY-CODE is 840 (USA), this field must contain a 9 digit US Zip code. If the last 4 digits of the zip code are not known, use zeros. |
| `countryCode` | N/A | **Required:** the destination country code (USA = 840). A valid country code between 0 and 999 must be specified. |
| `serviceCode` | N/A | **Required:** the UPS class of service for the package. Valid service codes include:<br>NEXT DAY AIR 1<br>2ND DAY AIR 2<br>3 DAY SELECT 12<br>GROUNDTRAC 3<br>DELIVERYTRAC II 5<br>DELIVERYTRAC 8<br>Consult your UPS documentation for more information. |
| `trackingNumber` | 10 | **Required:** the UPS Tracking Number (composed of the two-character data identifier, seven-digit reference number, and one digit check digit). |
| `shipperNumber` | 6 | **Required:** your UPS-assigned Shipper Number represented as a NUL terminated string. |
| `julianDayOfPickup` | N/A | **Required:** the numeric day of the year in which the package was picked up (Jan 1 = 1, Jan 31 = 31, Feb 1 = 32, Feb 28 = 57, etc.). Value cannot exceed 366. |
| `shipmentId` | 30 | **Optional:** the shipper-assigned identification number for the shipment represented as a NUL terminated string.<br>If not specified, this field must contain an empty string (the NUL character).<br>It is highly recommended that this field **always** be left blank; otherwise, the message may be too long to encode. |
| `packageNumber` | N/A | **Required:** the "N" in "Package N of X." Value cannot exceed 999. |
| `packageCount` | N/A | **Required:** the "X" in "Package N of X." Value cannot exceed 999. |
| `packageWeight` | N/A | **Required:** the weight of the package in 10ths of a pound (e.g., 2lbs = 20). Value cannot exceed 9999 (999 pounds). UPS requires that the package weight be rounded up to next pound.<br>If not known, this field must be zero. |

| Element | Max Len | Contents |
|---|---|---|
| addressValidation | 1 | **Required:** indicates that the address information provided comes from a certified data source. Valid values are 'Y' or 'N' |
| shipToAddress | 35 | **Optional:** the destination street address represented as a NUL terminated string.<br>If not specified, this field must contain an empty string (the NUL character). |
| shipToCity | 35 | **Required:** the destination city represented as a NUL terminated string.<br>**NOTE:** even though this field allows up to 35 characters, you must truncate the city name to 20 characters maximum; otherwise your Maxicode symbol may not encode.<br>It is highly recommended that this field **always** be left blank; otherwise, the message may be too long to encode. |
| shipToState | 2 | **Required:** the destination state abbreviation represented as a NUL terminated string.. |
| modeControl | 1 | **Required:** Encoder mode control<br>*This field is new in the Version 2.1.x encoder.* It has the following defined values:<br>'0' Use the AIM-standard algorithm to automatically determine the encoding mode.<br>'1' Use the alternate UPS algorithm to automatically determine the encoding mode.<br>'2' Force the encoder to use Mode 2 to encode the symbol.<br>'3' Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

The output structure, `sMaxicodeFontOutput`, has two fields:

```
#define MAXICODE_ROW_PAIRS  17
#define MAXICODE_COLS       30

struct sMaxicodeFontOutput
{
    char        output[MAXICODE_ROW_PAIRS][MAXICODE_COLS];
    MAXIERROR   resultCode;
};
```

| Element | Max Len | Contents |
|---|---|---|
| output | 17 x 30 | If the encoder can successfully encode the input data, this table will contain the font characters which, when printed using the provided font, will create the Maxicode symbol. |
| resultCode | N/A | The encoder returns its status in this field. If the encoder was successful, MERR_OK is returned; otherwise an error code is returned (see next table). |

`resultCode` is an enumerated type, `MAXIERROR`, defined in `maxidefs.h` (this file is automatically included by `maxiapi.h`). The table below lists the error values defined for `MAXIERROR`:

| Value | Meaning |
|---|---|
| `MERR_OK` | Success |
| `MERR_INV_PARAM` | An invalid parameter was passed. This typically indicates that a NULL pointer was passed to the encoder. It also indicates an invalid julianDayOfPickup. |
| `MERR_TOO_LONG` | The data passed will not fit in a single Maxicode symbol; too many of the optional fields have been populated. |
| `MERR_INV_POSTALCODE` | postalCode is invalid. Possible causes include:<br>• postalCode contains characters other than digits and upper-case alphabetic characters.<br>• countryCode is 840 and postalCode is not a 9 digit Zip Code. |
| `MERR_INV_COUNTRYCODE` | countryCode is invalid. This occurs if countryCode is greater than 999. |
| `MERR_INV_SERVICECLASS` | serviceClass is invalid. This occurs if serviceClass is greater than 999. |
| `MERR_INV_TRACKINGNUM` | trackingNumber is not 10 characters. |
| `MERR_INV_ADDRVALIDATION` | addressValidation is invalid. Valid values are 'Y', 'N', space, or NUL. |
| `MERR_INV_ADDRSTATE` | shipToState is invalid. The field must be two characters in length or a NUL string. |
| `MERR_INV_PACKAGENUM` | The packageNumber and packageCount fields are incorrect or inconsistent. Possible causes include:<br>• One of the fields is zero and the other is not<br>• One or both field values exceed 999.<br>• packageNumber is greater than packageCount |
| `MERR_INV_WEIGHT` | packageWeight is invalid. This occurs if packageWeight exceeds 9999 (999 lbs.). |

In addition to these values, a number of "internal" error codes are defined. All these errors have a value of 100 or greater. Should you encounter an internal error code, please note the code and contact Silver Bay Software technical support.

**Notes:**

- A Maxicode symbol can only hold about 100 characters; if you populate any of the "optional" data elements of the input record, the encoder may not be able to generate a symbol and will return a result code of 2 (`MERR_TOO_LONG`).
- For backward compatibility with older versions of the Maxicode encoder, the `shipToCity` field is 35 characters long. However, you ***mus***t truncate the city name to 20 characters maximum to comply with UPS' latest specifications. Furthermore, it is highly recommended that this field always be left blank; otherwise, the message may be too long to encode.

- Also for backward compatibility, the `packageWeight` field is expressed as an integer in tenths of a pound. However, UPS now requires that the package weight be rounded up to the next pound. You must still format the input to the encoder in tenths of a pound (i.e., multiply the actual weight in pounds by 10).
- The degree to which the encoder API validates postal codes, country codes and service classes is limited to ensuring that the data passed can be encoded into a Maxicode label (i.e. is the correct length, and is of the correct type – digits or alphanumerics). The encoder has no way to ensure that the actual data is valid.
- Be sure to always initialize the entire input record; failure to do so could result in "garbage" characters being encoded, or even encoder errors (`MERR_TOO_LONG`, message too long to encode).

**Example:**

```
MAXIUPSINFO maxiInput;
MAXIFONTOUT maxiOutput;

/*  Set entire structure to nulls (zeros)  */
memset(&maxiInput, 0, sizeof(maxiInput));

/*  Populate required fields  */
strcpy(maxiInput.postalCode, "841706672");
maxiInput.countryCode = 840;
maxiInput.serviceClass = 3
strcpy(maxiInput.trackingNumber, "1Z12345675");
strcpy(maxiInput.shipperNumber, "12345E");
maxiInput.julianDayOfPickup = 89;
maxiInput.packageNumber = 1;
maxiInput.packageCount = 1;
maxiInput.packageWeight = 100;  /* In tenth's of a pound  */
maxiInput.addressValidation = 'Y';
strcpy(maxiInput.shipToCity, "SALT LAKE CITY");
strcpy(maxiInput.shipToState, "UT");
maxiInput.modeControl = '0';

/*  Call the encoder  */
MaxUpsNC(&maxiInput, &maxiOutput);

if ( maxiOutput.resultCode == MERR_OK )
{
    /* Call function to print symbol  */
    PrintSymbol("CUPS", &maxiOutput);
}
else
{
    printf("encoding error %d\n", (int)maxiOutput.resultCode);
}
```

## 3.3 Visual Basic API

The encoder provides a version of the UPS encoding routine that is adapted for use by Visual Basic programs, or by applications that use embedded Visual Basic techniques, such as Microsoft Word or Microsoft Excel.

In order to use the encoder with Visual Basic, you must define User Defined Types to match the input and output data structures expected by the encoder, and declare the API function itself. This may be done as indicated below.

Input to the encoder is performed using a User Defined Type formatted like this:

```
Private Type MaxicodeInput
       countryCode As Long
       serviceClass As Long
       julianDayOfPickup As Long
       packageNumber As Long
       packageCount As Long
       packageWeight As Long
       postalCode As String
       trackingNumber As String
       shipperNumber As String
       shipmentID As String
       shipToAddress As String
       shipToCity As String
       shipToState As String
       addressValidation As Boolean
       modeControl As String
End Type
```

The various elements are described in the following table:

| Element | Max Len | Contents |
|---|---|---|
| `postalCode` | 9 | **Required:** the destination zip code/postal code represented as a string. This field must contain one of the following:<br>• A 9 digit numeric, US Zip Code<br>• A 1-6 digit alphanumeric, International Postal Code (letters must be upper case).<br>If countryCode is 840 (USA), this field must contain a 9 digit US Zip code. If the last 4 digits of the zip code are not known, use zeros. |
| `countryCode` | N/A | **Required:** the destination country code (USA = 840). A valid country code between 0 and 999 must be specified. |
| `serviceClass` | N/A | **Required:** the UPS class of service for the package. Valid service codes include:<br>NEXT DAY AIR 1<br>2ND DAY AIR 2<br>3 DAY SELECT 12<br>GROUNDTRAC 3<br>DELIVERYTRAC II 5<br>DELIVERYTRAC 8<br>Consult your UPS documentation for more information. |
| `trackingNumber` | 10 | **Required:** the UPS Tracking Number (composed of the two-character data identifier, seven-digit reference number, and one digit check digit), represented as a string. |
| `shipperNumber` | 6 | **Required:** your UPS-assigned Shipper Number represented as a string. |

| Element | Max Len | Contents |
|---|---|---|
| julianDayOfPickup | N/A | **Required:** the numeric day of the year in which the package was picked up (Jan 1 = 1, Jan 31 = 31, Feb 1 = 32, Feb 28 = 57, etc.). Value cannot exceed 366. |
| shipmentId | 30 | **Optional:** the shipper-assigned identification number for the shipment represented as a string.<br>If not specified, this field must contain an empty string (“”).<br>It is highly recommended that this field **always** be left blank; otherwise, the message may be too long to encode. |
| packageNumber | N/A | **Required:** the “N” in “Package N of X.” Value cannot exceed 999. |
| packageCount | N/A | **Required:** the “X” in “Package N of X.” Value cannot exceed 999. |
| packageWeight | N/A | **Required:** the weight of the package in 10ths of a pound (e.g., 2lbs = 20). Value cannot exceed 9999 (999 pounds). UPS requires that the package weight be rounded up to next pound.<br>If not known, this field must be zero. |
| addressValidation | 1 | **Required:** indicates that the address information provided comes from a certified data source. Valid values are True or False |
| shipToAddress | 35 | **Optional:** the destination street address represented as a string.<br>If not specified, this field must contain an empty string. |
| shipToCity | 35 | **Required:** the destination city represented as a string.<br>**NOTE:** even though this field allows up to 35 characters, you must truncate the city name to 20 characters maximum; otherwise your Maxicode symbol may not encode.<br>It is highly recommended that this field **always** be left blank; otherwise, the message may be too long to encode. |
| shipToState | 2 | **Required:** the destination state abbreviation represented as a string. |
| modeControl | 1 | **Required:** Encoder mode control<br>*This field is new in the Version 2.1.x encoder*. It has the following defined values:<br>“0” Use the AIM-standard algorithm to automatically determine the encoding mode.<br>“1” Use the alternate UPS algorithm to automatically determine the encoding mode.<br>“2” Force the encoder to use Mode 2 to encode the symbol.<br>“3” Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

The output area contains two fields:

```
Private Type MaxicodeOutput
       resultCode As Integer
       outputLine(16) As String
End Type
```

| Element | Max Len | Contents |
|---|---|---|
| `outputLine` | 17x30 | If the encoder can successfully encode the input data, this table will contain the font characters which, when printed using the provided font, will create the Maxicode symbol. |
| `resultCode` | N/A | The encoder returns its status in this field. If the encoder was successful, zero is returned; otherwise an error code is returned (see next table). |

The following values are defined for the `resultCode` field:

| Value | Meaning |
|---|---|
| 0 | Success |
| 1 | An invalid parameter was passed. This typically indicates that a NULL pointer was passed to the encoder. This error does not normally occur with Visual Basic programs. |
| 2 | The data passed will not fit in a single Maxicode symbol; too many of the optional fields have been populated. |
| 3 | postalCode s invalid. Possible causes include:<br>• postalCode contains characters other than digits and upper-case alphabetic characters.<br>• COUNTRY-CODE is 840 and postalCode is not a 5-digit or 9-digit Zip Code. |
| 4 | countryCode is invalid. This occurs if countryCode is not numeric. |
| 5 | serviceClass is invalid. This occurs if serviceClass is not numeric. |
| 6 | trackingNumber is not 10 characters. |
| 7 | addressValidation is invalid. Valid values are "True" and "False" |
| 8 | shipToState is invalid. The field must be two characters in length or all blanks. |
| 9 | The packageNumber and packageCount fields are incorrect or inconsistent. Possible causes include:<br>• One of the fields is zero and the other is not<br>• One or both of the fields are not numeric<br>• packageNumber is greater than packageCount |
| 11 | packageWeight is invalid. Possible causes include:<br>• The field is not numeric<br>• The field has not been formatted correctly (must be in the range of 0 through 999) |

In addition to these values, a number of "internal" error codes are defined. All these errors have a value of 100 or greater. Should you encounter an internal error code, please note the code and contact Silver Bay Software technical support.

Finally, it is necessary to declare the API routine to Visual Basic:

```
Private Declare Sub MaxUpsVB Lib "maxicode2101.dll"
       (ByRef mxInput As MaxicodeInput, ByRef mxOutput As MaxicodeOutput)
```

**Notes:**

- A Maxicode symbol can only hold about 100 characters; if you populate any of the "optional" data elements of the input record, the encoder may not be able to generate a symbol and will return a result code of 2 (data too long).

- The `shipToCity` field ***mus***t be truncated to 20 characters maximum to comply with UPS' latest specifications. Furthermore, it is highly recommended that this field always be left blank; otherwise, the message may be too long to encode.
- Also for backward compatibility, the `packageWeight` field is expressed as an integer in tenths of a pound. However, UPS now requires that the package weight be rounded up to the next pound. You must still format the input to the encoder in tenths of a pound (i.e., multiply the actual weight in pounds by 10).
- The degree to which the encoder API validates postal codes, country codes and service classes is limited to ensuring that the data passed can be encoded into a Maxicode label (i.e. is the correct length, and is of the correct type – digits or alphanumerics). The encoder has no way to ensure that the actual data is valid.
- Be sure to always initialize the entire input record; failure to do so could result in "garbage" characters being encoded, or even encoder errors (`MERR_TOO_LONG`, message too long to encode).

**Example:**

```
Dim mxInput As MaxicodeInput
Dim mxOutput As MaxicodeOutput

mxInput.countryCode = 840
mxInput.serviceClass = 1
mxInput.trackingNumber = "1Z12345670"
mxInput.shipperNumber = "123456"
mxInput.julianDayOfPickup = 12
mxInput.packageCount = 1
mxInput.packageNumber = 1
mxInput.packageWeight = 10
mxInput.postalCode = "339120000"
mxInput.shipToCity = "Fort Myers"
mxInput.shipToState = "FL"
mxInput.addressValidation = False
mxInput.modeControl = "0"

Call MaxUpsVB(mxInput, mxOutput)

If mxOutput.resultCode = 0 Then
    For i = 0 To 16
        'Print mxOutput.outputLine(i) via the appropriate method
    Next i
Else
    'an error has occurred
End If
```

# 4 Generic Maxicode API's

Although the UPS shipping functions described in the previous chapter will suffice for most applications, the encoder provides a more "generic" set of functions for generating Maxicode symbols and manipulating their contents. These API's are described in this chapter.

## 4.1 COBOL Language API

### *4.1.1 Initialization*

Before any of the encoding API functions are called, the encoder must be initialized. This process indicates to the encoder which binary values it should use to output the encoded symbol. This is important because different printers and different host-to-printer interconnect methods result in different character translations between host and printer. The Maxicode encoder compensates for such variations by outputting different values, by using different font files, or both. Be sure you read the chapter entitled **Printing the Maxicode Symbol** for information related to how to print the symbol characters properly in your environment.

Call examples:

```
CALL "MAXINIT" USING MAXICODE-FONT-INFO-REC.
```

Arguments:

| Position | Record Format | Use |
|---|---|---|
| 1 | MAXICODE-FONT-INFO-REC | Font characters to use |

Notes:

- Be sure you read the chapter entitled for information related to how to print the symbol characters properly in your environment.
- This function does not need to be called before every symbol encoded. It only needs to be called once as part of program initialization.
- If this function is not called, the default values indicated below are used. These default values may not be appropriate for your environment.

The record used by this function has the following format:

```
01  MAXICODE-FONT-INFO-REC.
    05  NO-HEX-CHAR          PIC X.
    05  LO-HEX-CHAR          PIC X.
    05  HI-HEX-CHAR          PIC X.
    05  TWO-HEX-CHAR         PIX X.
    05  PAD-CHAR             PIX X.
    05  BULL-CHAR            PIX X.
```

The elements of this record are as follows:

| Element | Contents |
|---|---|
| NO-HEX-CHAR | The "full width blank" character. |
| LO-HEX-CHAR | The character containing the second row hexagon. |
| HI-HEX-CHAR | The character containing the first row hexagon. |
| TWO-HEX-CHAR | The character containing both hexagons. |
| PAD-CHAR | The "narrow blank" character. |
| BULL-CHAR | The character containing the bull's-eye character. |

Default values output by the encoder (in hexadecimal) are:

| Element | ASCII | EBCDIC |
|---|---|---|
| NO-HEX-CHAR | 0x30 | 0xF0 |
| LO-HEX-CHAR | 0x31 | 0xF1 |
| HI-HEX-CHAR | 0x32 | 0xF2 |
| TWO-HEX-CHAR | 0x33 | 0xF3 |
| PAD-CHAR | 0x34 | 0xF4 |
| BULL-CHAR | 0x35 | 0xF5 |

### *4.1.2 Result Codes*

The Maxicode encoder API functions indicate success or failure by returning a numeric result code in the output record. The following return codes are defined:

| Value | Meaning |
|---|---|
| 000 | Success |
| 001 | An invalid parameter was passed. This typically indicates a NULL pointer, to a function. |
| 002 | The data passed would not fit in a single Maxicode symbol. |
| 003 | The postal code was invalid. Possible causes include:<br>• The postal code is longer than 9 digits, or 6 alphanumeric characters<br>• The postal code contains characters other than digits and upper-case alphabetic characters |
| 004 | The country code was invalid. Possible causes include:<br>• The country code was not numeric or it was greater than 999. |
| 005 | The service class was invalid. Possible causes include:<br>• The service class was not numeric or it was greater than 999. |
| 006 | The tracking number field of the UPS structure was invalid (wrong length) |
| 007 | The address validation field of the UPS structure was invalid. Valid values are 'Y', 'N', space or NUL |
| 008 | The SHIP-TO-STATE field of the UPS structure was invalid. It must be either empty (all blanks), or two characters in length. |
| 009 | The package number and package count fields are incorrect or inconsistent. Possible causes include:<br>• One of the fields is zero and the other is not<br>• The package number is greater than the package count |

| Value | Meaning |
|---|---|
| 010 | The symbol number or symbol count fields are incorrect or inconsistent. Possible causes include:<br>• One of the fields is less than 1 or greater than 8.<br>• The symbol number is larger than the symbol count |
| 011 | The weight is invalid. Typically caused by a weight greater than 999.9 pounds. |

In addition to these values, a number of "internal" error codes are defined. All these have values 100 or greater. Should you encounter an internal error code, please note the error code and contact Silver Bay Software technical support.

Note that the degree to which the encoder API validates postal codes, country codes and service classes is limited to ensuring that the data passed can be encoded into a Maxicode label. The encoder does not ensure that the actual data is valid.

## *4.1.3 Record Formats*

### 4.1.3.1 Output Record

All the Maxicode encoder COBOL API functions return their output into a record of the following form:

```
01  MAXICODE-OUTPUT-REC.
    05  OUTPUT-LINES         PIC X(30) OCCURS 17.
    05  RESULT-CODE          PIC 9(3).
```

The elements of this record are as follows:

| Element | Contents |
|---|---|
| OUTPUT-LINES | This is the output data. This area consists of 17 rows of 30 characters. When rendered in the special Maxicode font provided with the encoder, these characters produce the image of the Maxicode symbol. |
| RESULT-CODE | This code indicates the success or failure of the operation. See **Result Codes** on page 33 for a description of the values. |

### 4.1.3.2 UPS-Specific Input Record Format

The following record format is used as input to the UPS-specific encoder routines:

```
01  MAXICODE-UPS-INFO-REC.
    05  POSTAL-CODE          PIC X(9).
    05  COUNTRY-CODE         PIC 9(3).
    05  SERVICE-CLASS        PIC 9(3).
    05  TRACKING-NUMBER      PIC X(10).
    05  SHIPPER-NUMBER       PIC X(6).
    05  JULIAN-DAY-OF-PICKUP PIC 9(3).
    05  SHIPMENT-ID          PIC X(30).
    05  PACKAGE-NUMBER       PIC 9(4).
    05  PACKAGE-COUNT        PIC 9(4).
    05  PACKAGE-WEIGHT       PIC 999.9.
    05  ADDRESS-VALIDATION   PIC X(1).
    05  SHIP-TO-ADDRESS      PIC X(35).
    05  SHIP-TO-CITY         PIC X(35).
    05  SHIP-TO-STATE        PIC X(2).
    05  MODE-CONTROL         PIC X(1).
```

The mandatory elements of this record are as follows:

| Element | Contents |
|---|---|
| COUNTRY-CODE | The country code for the destination country.<br>(USA's country code is 840) |
| SERVICE-CLASS | The "class of service" code assigned by UPS. |
| POSTAL-CODE | The destination postal code. For all-numeric postal codes, this may be up to nine digits.<br>If the country code is set to 840 (USA), this field must be exactly 5 or 9 digits in length.<br>For postal codes containing alphabetic characters (which must be in upper case), the limit is six characters.<br>Space pad the field for postal codes less than 9 digits. |
| TRACKING-NUMBER | The 10-character UPS tracking number for the item. |
| MODE-CONTROL | Encoder mode control<br>This field is new in the Version 2.1.x encoder. It has the following defined values:<br>'0' Use the AIM-standard algorithm to automatically determine the encoding mode.<br>'1' Use the alternate UPS algorithm to automatically determine the encoding mode.<br>'2' Force the encoder to use Mode 2 to encode the symbol.<br>'3' Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

The optional elements of the record are shown below. To omit a numeric item, set it to zero. To omit a string item, set it to all spaces.

| Element | Max Len | Contents |
|---|---|---|
| SHIPPER-NUMBER | 6 | The UPS-assigned number for the shipper |
| JULIAN-DAY-OF-PICKUP | 3 | The numeric day of the year in which the package was picked up. (Jan 1 = 1, Jan 31 = 31, Feb 1 = 32, Feb 28 = 57, etc) |
| SHIPMENT-ID | 30 | The shipper-assigned identification number for the shipment |
| PACKAGE-NUMBER | 4 | The "N" in "Package N of X" |
| PACKAGE-COUNT | 4 | The "X" in "Package N of X" |
| PACKAGE-WEIGHT | | The weight of the package in pounds and tenths of pounds |
| ADDRESS-VALIDATION | 1 | "Y", "N" (or space to omit) |
| SHIP-TO-ADDRESS | 35 | The destination street address |
| SHIP-TO-CITY | 35 | The destination city |
| SHIP-TO-STATE | 2 | The destination state abbreviation |

**NOTE:** If all the information listed in these tables is provided for an individual package, it is quite possible that the data will not fit into a single symbol. After encoding the message header, mandatory information and the message terminator, a Maxicode symbol only has space for a maximum of 53 characters of optional information. This includes the $^{G}_{S}$ characters separating

the fields. As a result, a single symbol can hold only a maximum of 43 characters of optional information.

Input parameter validation is limited to ensuring that the data passed in will fit within a Maxicode barcode. The data tests are thus typically limited to ensuring that individual fields are not too big to encode.

#### 4.1.3.3 SCM-Specific Input Record Format

The following record format is used as input to the SCM encoder routines:

```
01  MAXICODE-SCM-REC.
    05  SYMBOL-NUM           PIC 9(1).
    05  SYMBOL-COUNT         PIC 9(1).
    05  POSTAL-CODE-LEN      PIC 9(1).
    05  POSTAL-CODE          PIC X(9).
    05  COUNTRY-CODE         PIC 9(3).
    05  SERVICE-CLASS        PIC 9(3).
    05  SECONDARY-MSG-LEN    PIC 9(3).
    05  SECONDARY-MSG        PIC X(126).
    05  MODE-CONTROL         PIC X(1).
```

The elements of this record are as follows:

| Element | Contents |
|---|---|
| SYMBOL-NUM | The "N" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| SYMBOL-COUNT | The "X" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| POSTAL-CODE-LEN | Length of the postal code (1-9 for numeric postal codes, 1-6 for alphanumeric postal codes) |
| POSTAL-CODE | The destination postal code. For all-numeric postal codes, this may be up to nine digits.<br>If the country code is set to 840 (USA), this field must be exactly 5 or 9 digits in length.<br>For postal codes containing alphabetic characters (which must be in upper case), the limit is six characters.<br>Space pad the field for postal codes less than 9 digits. |
| COUNTRY-CODE | The country code for the destination country.<br>(USA's country code is 840) |
| SERVICE-CLASS | The "class of service" code |
| SECONDARY-MSG-LEN | The length of the secondary message field. |
| SECONDARY-MSG | The secondary message field itself. |

| Element | Contents |
|---|---|
| MODE-CONTROL | Encoder mode control<br>This field is new in the Version 2.1.x encoder. It has the following defined values:<br>'0' Use the AIM-standard algorithm to automatically determine the encoding mode.<br>'1' Use the alternate UPS algorithm to automatically determine the encoding mode.<br>'2' Force the encoder to use Mode 2 to encode the symbol.<br>'3' Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

#### 4.1.3.4 Pre-Formatted String Input Record Format

The following record format is used as input to the string encoder routines:

```
01  MAXICODE-STRING-REC.
    05  SYMBOL-NUM           PIC 9(1).
    05  SYMBOL-COUNT         PIC 9(1).
    05  INPUT-LEN            PIC 9(3).
    05  INPUT-STRING         PIC X(144).
```

The elements of this record are as follows:

| Element | Contents |
|---|---|
| SYMBOL-NUM | The "N" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| SYMBOL-COUNT | The "X" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| INPUT-LEN | Length of the pre-formatted input string. |
| INPUT-STRING | The pre-formatted Structured Carrier Message string. |

### *4.1.4 Encoding Structured Carrier Message Symbols*

The Maxicode encoder provides support for encoding Structured Carrier Messages in a variety of manners:

- Via UPS-specific functions.
  This is the simplest way to encode a Maxicode symbol for UPS use. All internal formatting is handled automatically by the function.
- Via String functions.
  If your application is already constructing Structured Carrier Message strings, these functions will accept this as input. These functions are not UPS-specific, but will generate UPS-compatible output if the string input is formatted properly.
- Via generic Structured Carrier Message (SCM) functions.
  These functions are the most general purpose and powerful. They permit the generation of Structured Message Append symbols, but require the secondary message to be properly formatted as discussed in **Primary and Secondary Message Formats** on page 13. As with the string functions, these functions are not UPS-specific, but will generate UPS-compatible output if the input is formatted properly.

In addition, the Maxicode encoder provides two versions of each API function. One version assumes that the input data is in the native character set of the local computer system (ASCII or EBCDIC). These functions automatically perform ANSI EBCDIC-to-ASCII conversion on their input data if the native character set is EBCDIC. The other assumes that its input is in ASCII, regardless of whether the local computer system is based on ASCII or EBCDIC, and thus performs no character translations, even on EBCDIC machines. Normally, the native functions are the simplest to use. The ASCII functions are provided to handle the case where binary information is encoded that would be improperly handled by an internal EBCDIC-to-ASCII conversion, or where an EBCDIC-to-ASCII conversion other than the ANSI standard one is required.

The API functions available are as follows:

| Function | Input form | Character Set |
|---|---|---|
| MAXUPSA | UPS Structure | ASCII |
| MAXUPSN | UPS Structure | Native |
| MAXSCMA | SCM Structure | ASCII |
| MAXSCMN | SCM Structure | Native |
| MAXSTRA | SCM String | ASCII |
| MAXSTRN | SCM String | Native |

#### 4.1.4.1 UPS Functions

The UPS-specific functions are listed below. These functions take their input in a record that contains all the mandatory and optional fields for a UPS-specific Maxicode symbol. These routines then automatically perform the low-level message formatting required. These routines provide the easiest-to-use method of integrating UPS Maxicode support into an application.

Call examples:

```
CALL "MAXUPCN" USING MAXICODE-UPS-INFO-REC, MAXICODE-OUTPUT-REC.

CALL "MAXUPCA" USING MAXICODE-UPS-INFO-REC, MAXICODE-OUTPUT-REC.
```

Arguments:

| Position | Record Format | Use |
|---|---|---|
| 1 | MAXICODE-UPS-INFO-REC | Input data |
| 2 | MAXICODE-OUTPUT-REC | Output data |

Notes:

- No support is provided for Structured Append using the UPS-specific functions. If the presented data does not fit into a single symbol, the result code 002 will be returned. To break information up across two or more symbols, the SCM functions must be used.
- MAXUPCN assumes the contents of the input record are in the native character set, and performs EBCDIC to ASCII conversions automatically as required.

Example (The following code segment is from the sample program CBLUPS)

```
*************************************************************
* This section formats input data, creates a call to sub-     *
* routines and creates an output file which includes a maxicode
* symbol and a result code from each call                     *
*************************************************************
 2000-PROCESS-DATA.

     PERFORM 3000-POPULATE-UPS-REC.
     PERFORM 4000-ENCODE-SYMBOL.
     IF MAXI-OK
         PERFORM 5000-PRINT-SYMBOL
     ELSE
         DISPLAY 'Encoder failed.  Error code = '
                 RESULT-CODE.

*************************************************************
* Populate the fields in the UPS record.                     *
*************************************************************
 3000-POPULATE-UPS-REC.

     MOVE '339010000'         TO POSTAL-CODE.
     MOVE 840                 TO COUNTRY-CODE.
     MOVE 001                 TO SERVICE-CLASS.
     MOVE '1Z34567890'        TO TRACKING-NUMBER.
     MOVE '102562'            TO SHIPPER-NUMBER.
     MOVE 034                 TO JULIAN-DAY-OF-PICKUP.
     MOVE SPACES              TO SHIPMENT-ID.
     MOVE 1                   TO PACKAGE-NUMBER.
     MOVE 1                   TO PACKAGE-COUNT.
     MOVE 20.0                TO PACKAGE-WEIGHT.
     MOVE 'Y'                 TO ADDRESS-VALIDATION.
     MOVE '2201 SECOND ST'    TO SHIP-TO-ADDRESS.
     MOVE 'FT MYERS'          TO SHIP-TO-CITY.
     MOVE 'FL'                TO SHIP-TO-STATE.
     MOVE '0'                 TO MODE-CONTROL.

*************************************************************
* Call the Maxicode encoder.                                 *
*************************************************************
 4000-ENCODE-SYMBOL.

     CALL 'MAXUPSN' USING MAXICODE-UPS-INFO-REC
                          MAXICODE-OUTPUT-REC.
```

#### 4.1.4.2 String Functions

The string functions take as their input records containing a pre-formatted Structured Carrier Message string, along with the length of that string. The routines encode the contents of this string into the Maxicode symbol. Use these functions if your application is already formatting Structured Carrier Messages that you want to represent in a Maxicode symbol.

Call examples:

```
CALL "MAXSTRN" USING MAXICODE-STRING-REC, MAXICODE-OUTPUT-REC.

CALL "MAXSTRA" USING MAXICODE-STRING-REC, MAXICODE-OUTPUT-REC.
```

Arguments:

| Position | Record Format | Use |
|---|---|---|
| 1 | MAXICODE-STRING-REC | Input data |
| 2 | MAXICODE-OUTPUT-REC | Output data |

Notes:

- No support is provided for Structured Append using the String functions. If the presented data does not fit into a single symbol, the result code 002 will be returned. To break information up across two or more symbols, the SCM functions must be used.
- MAXSTRN assumes the contents of the input record are in the native character set, and performs EBCDIC to ASCII conversions automatically as required.
- An example of a pre-formatted Structured Carrier Message string is shown below:

  [)>$^{R}_{S}$01$^{G}_{S}$96123456789$^{G}_{S}$840$^{G}_{S}$001$^{G}_{S}$1Z12345678$^{G}_{S}$UPSN$^{R}_{S}$$^{E}$0$_{T}$

Example (The following code segment is from the sample program CBLSTR)

```
*************************************************************
* This section formats input data, creates a call to sub-     *
* routines and creates an output file which includes a maxicode
* symbol and a result code from each call                     *
*************************************************************
 2000-PROCESS-DATA.

     PERFORM 3000-POPULATE-STRING-REC
     PERFORM 4000-ENCODE-SYMBOL
     IF MAXI-OK
         PERFORM 5000-PRINT-SYMBOL
     ELSE
         DISPLAY 'Encoder failed.  Error code = '
                 RESULT-CODE.

*************************************************************
* Populate the fields in the STRING record.                   *
*************************************************************
 3000-POPULATE-STRING-REC.

     MOVE 1                   TO SYMBOL-NUM
     MOVE 1                   TO SYMBOL-COUNT

*************************************************
*    We need to build the SCM string as per the   *
```

```
*    specification.  Start with the standard 9    *
*    byte header.                                 *
************************************************
     MOVE SCM-HEADER          TO TMP-DATA
     MOVE 9                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD

*    Append postal code
     MOVE '339010000'         TO TMP-DATA
     MOVE 9                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Ship-To country code
     MOVE '840'               TO TMP-DATA
     MOVE 3                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Class of Service
     MOVE '001'               TO TMP-DATA
     MOVE 3                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Tracking number
     MOVE '1Z34567890'        TO TMP-DATA
     MOVE 10                  TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append SCAC code
     MOVE 'UPSN'              TO TMP-DATA
     MOVE 4                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append UPS Shipper Number
     MOVE '102562'            TO TMP-DATA
     MOVE 6                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Julian Day of Pickup
     MOVE '034'               TO TMP-DATA
     MOVE 3                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Shipment ID Number (none, so just GS)
     PERFORM 2200-APPEND-GS

*    Append Package N of X
     MOVE '1/1'               TO TMP-DATA
     MOVE 3                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Weight (this is 20 lbs)
     MOVE '20'                TO TMP-DATA
     MOVE 2                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS
```

```
*    Append Address Validation
     MOVE 'Y'                 TO TMP-DATA
     MOVE 1                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Ship-To Address
     MOVE '2201 SECOND ST'    TO TMP-DATA
     MOVE 14                  TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Ship-To City
     MOVE 'FT MYERS'          TO TMP-DATA
     MOVE 8                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Ship-To State
     MOVE 'FL'                TO TMP-DATA
     MOVE 2                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD

*    Append the SCM trailer (RS + EOT)
     MOVE SCM-TRAILER         TO TMP-DATA
     MOVE 2                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD

     MOVE TMP-STRING          TO INPUT-STRING
     MOVE TMP-STRING-LEN      TO INPUT-LEN.

*************************************************************
* Append TMP-FIELD to the end of TMP-STRING                 *
*************************************************************
 2100-MOVE-FIELD.

     MOVE TMP-DATA TO
          TMP-STRING(TMP-STRING-LEN + 1 : TMP-DATA-LEN)
     ADD TMP-DATA-LEN TO TMP-STRING-LEN.

*************************************************************
* Append a group seperator to the secondary message         *
*************************************************************
 2200-APPEND-GS.

     MOVE NATIVE-GS TO TMP-DATA
     MOVE 1         TO TMP-DATA-LEN

     PERFORM 2100-MOVE-FIELD.

*************************************************************
* Call the Maxicode encoder.                                *
*************************************************************
 4000-ENCODE-SYMBOL.

     CALL 'MAXSTRN' USING MAXICODE-STRING-REC
                          MAXICODE-OUTPUT-REC.
```

#### 4.1.4.3 Generic Structured Carrier Message (SCM) Functions

The SCM functions take as their input a record containing non-UPS-specific Structured Carrier Message data. The contents of this record reflect the low-level formatting of a Structured Carrier Message in a Maxicode, and require that the user perform the proper formatting of the Secondary Message portion of the message. (See the section titled **Primary and Secondary Message Formats**). These routines also provide support for Structured Append. Use these functions if you are preparing a Structured Carrier Message for a carrier other than UPS and it is not convenient to use the String functions, or in situations in which it is necessary to split a message across two or more symbols.

Call examples:

```
CALL "MAXSCMN" USING MAXICODE-SCM-REC, MAXICODE-OUTPUT-REC.

CALL "MAXSCMA" USING MAXICODE-SCM-REC, MAXICODE-OUTPUT-REC.
```

Arguments:

| Position | Record Format | Use |
|---|---|---|
| 1 | MAXICODE-SCM-REC | Input data |
| 2 | MAXICODE-OUTPUT-REC | Output data |

Notes:

- MAXSCMN assumes the contents of the input structure are in the native character set, and performs EBCDIC to ASCII conversions automatically as required.

Example (The following code segment is from the sample program CBLSCM)

```
************************************************************
* This section formats input data, creates a call to sub-     *
* routines and creates an output file which includes a maxicode
* symbol and a result code from each call                     *
************************************************************
 2000-PROCESS-DATA.

     PERFORM 3000-POPULATE-SCM-REC
     PERFORM 4000-ENCODE-SYMBOL
     IF MAXI-OK
         PERFORM 5000-PRINT-SYMBOL
     ELSE
         DISPLAY 'Encoder failed.  Error code = '
                 RESULT-CODE.

************************************************************
* Populate the fields in the SCM record.                      *
************************************************************
 3000-POPULATE-SCM-REC.

     MOVE 1                  TO SYMBOL-NUM
     MOVE 1                  TO SYMBOL-COUNT
     MOVE '339010000'        TO POSTAL-CODE
     MOVE 9                  TO POSTAL-CODE-LEN
     MOVE 840                TO COUNTRY-CODE
     MOVE 001                TO SERVICE-CLASS
     MOVE '0'                TO MODE-CONTROL

************************************************
```

```
*    We need to build the secondary message as    *
*    per the specification.  Start with the       *
*    standard 9 byte header.                      *
************************************************
     MOVE SCM-HEADER          TO TMP-DATA
     MOVE 9                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD

*    Append Tracking number
     MOVE '1Z34567890'        TO TMP-DATA
     MOVE 10                  TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append SCAC code
     MOVE 'UPSN'              TO TMP-DATA
     MOVE 4                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append UPS Shipper Number
     MOVE '102562'            TO TMP-DATA
     MOVE 6                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Julian Day of Pickup
     MOVE '034'               TO TMP-DATA
     MOVE 3                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Shipment ID Number (none, so just GS)
     PERFORM 2200-APPEND-GS

*    Append Package N of X
     MOVE '1/1'               TO TMP-DATA
     MOVE 3                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Weight (this is 20 lbs)
     MOVE '20'                TO TMP-DATA
     MOVE 2                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Address Validation
     MOVE 'Y'                 TO TMP-DATA
     MOVE 1                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Ship-To Address
     MOVE '2201 SECOND ST'    TO TMP-DATA
     MOVE 14                  TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
     PERFORM 2200-APPEND-GS

*    Append Ship-To City
     MOVE 'FT MYERS'          TO TMP-DATA
     MOVE 8                   TO TMP-DATA-LEN
     PERFORM 2100-MOVE-FIELD
```

```
           PERFORM 2200-APPEND-GS

      *    Append Ship-To State
           MOVE 'FL'                TO TMP-DATA
           MOVE 2                   TO TMP-DATA-LEN
           PERFORM 2100-MOVE-FIELD

      *    Append the SCM trailer (RS + EOT)
           MOVE SCM-TRAILER         TO TMP-DATA
           MOVE 2                   TO TMP-DATA-LEN
           PERFORM 2100-MOVE-FIELD

           MOVE TMP-SCND-MSG        TO SECONDARY-MSG
           MOVE TMP-SCND-MSG-LEN    TO SECONDARY-MSG-LEN.

      ************************************************************
      * Append TMP-FIELD to the end of TMP-SCND-MSG              *
      ************************************************************
       2100-MOVE-FIELD.

           MOVE TMP-DATA TO
                TMP-SCND-MSG(TMP-SCND-MSG-LEN + 1 : TMP-DATA-LEN)
           ADD TMP-DATA-LEN TO TMP-SCND-MSG-LEN.

      ************************************************************
      * Append a group seperator to the secondary message        *
      ************************************************************
       2200-APPEND-GS.

           MOVE NATIVE-GS TO TMP-DATA
           MOVE 1         TO TMP-DATA-LEN

           PERFORM 2100-MOVE-FIELD.

      ************************************************************
      * Call the Maxicode encoder.                               *
      ************************************************************
       4000-ENCODE-SYMBOL.

           CALL 'MAXSCMN' USING MAXICODE-SCM-REC
                                MAXICODE-OUTPUT-REC.
```

### *4.1.5 Encoding Generic Message (Non-Structured Carrier Message) Data*

Although Maxicode was designed primarily for representing Structured Carrier Message data sets, it can be used to encode generic unstructured data as well, using modes 4 and 5, as discussed in **Internal Encoding Details** on page 8. The Maxicode encoder provides four functions to allow the creation of Maxicode symbols that do not follow the Structured Carrier Message format. Two levels of error correction are supported – Standard Error Correction, corresponding to Mode 4, and Extended Error Correction, corresponding to Mode 5. Extended Error Correction provides better protection against symbol damage, at the expense of allowing fewer characters to be encoded into a symbol. As with the Structured Carrier Message functions, ASCII and Native version are provided for each function.

The API functions available are as follows:

| Function | Error Correction | Character Set |
|---|---|---|
| MAXSECA | Standard | ASCII |

| | | |
|---|---|---|
| MAXSECN | Standard | Native |
| MAXEECA | Extended | ASCII |
| MAXEECN | Extended | Native |

Call examples:

```
CALL "MAXSECA" USING MAXICODE-STRING-REC, MAXICODE-OUTPUT-REC.

CALL "MAXSECN" USING MAXICODE-STRING-REC, MAXICODE-OUTPUT-REC.

CALL "MAXEECA" USING MAXICODE-STRING-REC, MAXICODE-OUTPUT-REC.

CALL "MAXEECN" USING MAXICODE-STRING-REC, MAXICODE-OUTPUT-REC.
```

Arguments:

| Position | Record Format | Use |
|---|---|---|
| 1 | MAXICODE-STRING-REC | Input data |
| 2 | MAXICODE-OUTPUT-REC | Output data |

Notes:

- MAXSECN and MAXEECN assume the contents of the input data are in the native character set, and perform EBCDIC to ASCII conversions automatically as required.

Example (The following code segment is from the sample program CBLEEC):

```
*************************************************************
* This section formats input data, creates a call to sub-     *
* routines and creates an output file which includes a maxicode
* symbol and a result code from each call                     *
*************************************************************
 2000-PROCESS-DATA.

     PERFORM 3000-POPULATE-EEC-REC
     PERFORM 4000-ENCODE-SYMBOL
     IF MAXI-OK
         PERFORM 5000-PRINT-SYMBOL
     ELSE
         DISPLAY 'Encoder failed.  Error code = '
                 RESULT-CODE.

*************************************************************
* Populate the fields in the SCM record.                      *
*************************************************************
 3000-POPULATE-EEC-REC.

     MOVE 1                   TO SYMBOL-NUM
     MOVE 1                   TO SYMBOL-COUNT

     MOVE "This is a message encoded using mode 5 (EEC)." TO
          INPUT-STRING
     MOVE 45                  TO INPUT-LEN.

*************************************************************
* Call the Maxicode encoder.                                  *
*************************************************************
 4000-ENCODE-SYMBOL.
```

```
        CALL 'MAXEECN' USING MAXICODE-STRING-REC
                             MAXICODE-OUTPUT-REC.
```

## 4.2 C Language API

### *4.2.1 Initialization*

Before any of the encoding API functions are called, the encoder must be initialized. This process indicates to the encoder which binary values it should use to output the encoded symbol. This is important because different printers and different host-to-printer interconnect methods result in different character translations between host and printer. The Maxicode encoder compensates for such variations by outputting different values, by using different font files, or both. Be sure you read the chapter entitled **Printing the Maxicode Symbol** for information related to how to print the symbol characters properly in your environment.

Prototype:

```
#include "maxiapi.h"      /* prototypes and structures */

void MaxInitC(MAXIINITCPTR pCharSet);
```

Arguments:

| Name | Use |
|---|---|
| pCharSet | Pointer to a MAXIINIT structure containing the character set information. |

Return Value:

```
None
```

Notes:

- This function does not need to be called before every symbol encoded. It only needs to be called once as part of program initialization.
- If this function is not called, the default values indicated below are used. These default values may not be appropriate for your environment.

The structure used by this function has the following definition:

```
struct sMaxiFontInit
{
    char    noHexChar;
    char    loHexChar;
    char    hiHexChar;
    char    twoHexChar;
    char    padChar;
    char    bullChar;
};
typedef struct sMaxiFontInit MAXIINIT;
typedef struct sMaxiFontInit MAXIPTR *MAXIINITPTR;
typedef const struct sMaxiFontInit MAXIPTR *MAXIINITCPTR;
```

The elements of this structure are as follows:

| Element | Contents |
|---|---|
| noHexChar | The "full width blank" character. |

| | |
|---|---|
| lohexChar | The character containing the second row hexagon. |
| hiHexChar | The character containing the first row hexagon. |
| twoHexChar | The character containing both hexagons. |
| padChar | The "narrow blank" character. |
| bullChar | The character containing the bullseye character. |

Default values output by the encoder (in hexadecimal) are:

| Element | ASCII | EBCDIC |
|---|---|---|
| noHexChar | 0x30 | 0xF0 |
| loHexChar | 0x31 | 0xF1 |
| hiHexChar | 0x32 | 0xF2 |
| twoHexChar | 0x33 | 0xF3 |
| padChar | 0x34 | 0xF4 |
| bullChar | 0x35 | 0xF5 |

### *4.2.2 Result Codes*

The Maxicode encoder API functions indicate success or failure with a return code of type MAXIERROR. The following return codes are defined:

| Symbol | Value | Meaning |
|---|---|---|
| MERR_OK | 0 | Success |
| MERR_INV_PARAM | 1 | An invalid parameter was passed. This typically indicates a NULL pointer, to a function. |
| MERR_TOO_LONG | 2 | The data passed would not fit in a single Maxicode symbol. |
| MERR_INV_POSTALCODE | 3 | The postal code was invalid. Possible causes include:<br>• The postal code is longer than 9 digits, or 6 alphanumeric characters<br>• The postal code contains characters other than digits and upper-case alphabetic characters<br>For country code 840 (USA), the postal code is neither 5 nor 9 characters in length |
| MERR_INV_COUNTRYCODE | 4 | The country code was invalid. Possible causes include:<br>• The country code was longer than three digits (string) or greater than 999. |
| MERR_INV_SERVICECLASS | 5 | The service class was invalid. Possible causes include:<br>• The service class was longer than three digits (string) or greater than 999. |
| MERR_INV_TRACKINGNUM | 6 | The tracking number field of the UPS structure was invalid (wrong length) |
| MERR_ADDR_VALIDATION | 7 | The address validation field of the UPS structure was invalid. Valid values are 'Y', 'N', space or NUL |

| Symbol | Value | Meaning |
|---|---|---|
| MERR_INV_STATE | 8 | The shipToState field of the UPS structure was invalid. It must be either empty or two characters in length. |
| MERR_INV_PACKAGENUM | 9 | The package number and package count fields are incorrect or inconsistent. Possible causes include:<br>• One of the fields is zero and the other is not<br>• The package number is greater than the package count |
| MERR_INV_SYMBOLNUM | 10 | The symbol number or symbol count fields are incorrect or inconsistent. Possible causes include:<br>• One of the fields is less than 1 or greater than 8.<br>• The symbol number is larger than the symbol count |
| MERR_INV_WEIGHT | 11 | The weight is invalid. Typically caused by a weight greater than 999.9 pounds. |

In addition to these values, a number of "internal" error codes are defined. All these have a value of 100 or greater. Should you encounter an internal error code, please note the code and contact Silver Bay Software technical support.

Note that the degree to which the encoder API validates postal codes, country codes and service classes is limited to ensuring that the data passed can be encoded into a Maxicode label (i.e. is the correct length, and is of the correct type – digits or alphanumeric). The encoder has no way to ensure that the actual data is valid.

### *4.2.3 Data Structures*

This section describes the various C data structures used for input and output with the encoder library. Note that some data elements may be larger than the maximum data length (for example, `postalCode` is 12 bytes even though the longest zip code supported is 9 digits). This was done to avoid potential data alignment problems with certain compilers and platforms.

#### 4.2.3.1 Output Structure

All the Maxicode encoder API functions return their output in a structure of the following form:

```
#define MAXICODE_ROW_PAIRS   17
#define MAXICODE_COLS        30

struct sMaxicodeFontOutput
{
    char        output[MAXICODE_ROWS][MAXICODE_COLS];
    MAXIERROR   resultCode;
};
typedef struct sMaxicodeFontOutput MAXIFONTOUT;
typedef struct sMaxicodeFontOutput MAXIPTR *MAXIFONTOUTPTR;
typedef const struct sMaxicodeFontOutput MAXIPTR *MAXIFONTOUTCPTR;
```

The elements of this structure are as follows:

| Element | Contents |
|---|---|
| output | This is the output data. This area consists of 17 rows of 30 characters. The rows are not NUL-terminated. |

| | |
|---|---|
| `resultCode` | This code indicates the success or failure of the operation. See **Result Codes** on page 48 for a description of the values. |

#### 4.2.3.2 UPS-Specific Input Structure

The following structure is used as input to the UPS-specific encoder routines documented on page 54:

```
struct sMaxicodeUpsInfo
{
    unsigned long   countryCode;
    unsigned long   serviceClass;
    unsigned long   julianDayOfPickup;
    unsigned long   packageNumber;
    unsigned long   packageCount;
    unsigned long   packageWeight;
    char            postalCode[12];
    char            trackingNumber[12];
    char            shipperNumber[8];
    char            shipmentID[32];
    char            shipToAddress[36];
    char            shipToCity[36];
    char            shipToState[4];
    char            addressValidation;
    char            modeControl;
    char            pad[2];
};
typedef struct sMaxicodeUpsInfo MAXIUPSINFO;
typedef struct sMaxicodeUpsInfo MAXIPTR *MAXIUPSINFOPTR;
typedef const struct sMaxicodeUpsInfo MAXIPTR *MAXIUPSINFOCPTR;
```

The mandatory elements of this structure are as follows:

| Element | Contents |
|---|---|
| `countryCode` | The country code for the destination country.<br>(USA's country code is 840) |
| `serviceClass` | The "class of service" code assigned by UPS. |
| `postalCode` | The destination postal code represented as a NUL-terminated string. For all-numeric postal codes, this may be up to nine digits.<br>If the country code is set to 840 (USA), this field must be numeric and exactly 5 or 9 digits in length.<br>For postal codes containing alphabetic characters (which must be in upper case), the limit is six characters. |
| `trackingNumber` | The 10-character UPS tracking number for the item. (NUL-terminated string) |
| `modeControl` | Encoder mode control<br>This field is new in the Version 2.1.x encoder. It has the following defined values:<br>'0' Use the AIM-standard algorithm to automatically determine the encoding mode.<br>'1' Use the alternate UPS algorithm to automatically determine the encoding mode.<br>'2' Force the encoder to use Mode 2 to encode the symbol.<br>'3' Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

The optional elements of the structure are shown below. To omit a numeric item, set it to zero. To omit a string item, make the first character of the string a NUL character.

| Element | Max Len | Contents |
|---|---|---|
| julianDayOfPickup | N/A | The numeric day of the year in which the package was picked up. (Jan 1 = 1, Jan 31 = 31, Feb 1 = 32, Feb 28 = 57, etc) |
| packageNumber | N/A | The "N" in "Package N of X" |
| packageCount | N/A | The "X" in "Package N of X" |
| packageWeight | N/A | The weight of the package in tenths of a pound. Thus, a 5-pound package would have the value 50. |
| shipperNumber | 6 | The UPS-assigned number for the shipper (NUL-terminated string) |
| shipmentID | 30 | The shipper-assigned identification number for the shipment (NUL-terminated string, up to 30 characters) |
| shipToAddress | 35 | The destination street address (NUL-terminated string, up to 35 characters) |
| shipToCity | 35 | The destination city (NUL-terminated string, up to 35 characters) |
| shipToState | 2 | The destination state abbreviation (2 characters, NUL-terminated string) |
| addressValidation | 1 | "Y", "N" or NUL to omit. |

The maximum length listed for string parameters in the table above does not include the trailing NUL character used to terminate the string. Note that some of the arrays are slightly longer than the legal limit. For example, the `postalCode` field is limited to a maximum of 9 characters plus a NUL, but the array is declared as 12 characters long. This is done for alignment and portability reasons. Characters after the NUL terminator are ignored.

**NOTE:** If all the information listed in these tables is provided for an individual package, it is quite possible that the data will not fit into a single symbol. After encoding the message header, mandatory information, and the message terminator, a Maxicode symbol only has space for a maximum of 53 characters of optional information. This includes the $^{G}_{S}$ characters separating the fields. As a result, a single symbol can hold only a maximum of 43 characters of optional information.

Input parameter validation is limited to ensuring that the data passed in will fit within a Maxicode barcode. The data tests are thus typically limited to ensuring that individual fields are not too big to encode.

#### 4.2.3.3 SCM-Specific Input Structure

The following structure is used as input to the SCM encoder routines documented on page 56:

```
struct sMaxicodeSCM
{
    int     symbolNum;
    int     symbolCount;
    int     postalCodeLen;
    char    postalCode[MAX_POSTAL_CODE+1];
    int     countryCode;
    int     serviceClass;
    int     secondaryMsgLen;
    char    secondaryMsg[MAX_SECONDARY_MSG+1];
    int     modeControl;
};
typedef struct sMaxicodeSCM MAXISCM;
typedef struct sMaxicodeSCM MAXIPTR *MAXISCMPTR;
typedef const struct sMaxicodeSCM MAXIPTR *MAXISCMCPTR;
```

The elements of this structure are as follows:

| Element | Max Len | Contents |
|---|---|---|
| `symbolNum` | N/A | The "N" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| `symbolCount` | N/A | The "X" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| `postalCodeLen` | N/A | Length of the postal code (1-9 for numeric postal codes, 1-6 for alphanumeric postal codes) |
| `postalCode` | 9 | The destination postal code represented as a NUL-terminated string. For all-numeric postal codes, this may be up to nine digits.<br>If the country code is set to 840 (USA), this field must be exactly 5 or 9 digits in length.<br>For postal codes containing alphabetic characters (which must be in upper case), the limit is six characters. |
| `countryCode` | N/A | The destination country code (USA = 840) |
| `serviceClass` | N/A | The class of service being used. |
| `secondaryMsgLen` | N/A | The length of the secondary message field. |
| `secondaryMsg` | 126 | The secondary message field itself. This may be NUL-terminated, but is not required to be. For UPS applications, this field must be formatted as described in **Primary and Secondary Message Formats** on page 13. |

| Element | Max Len | Contents |
|---|---|---|
| `modeControl` | 1 | Encoder mode control<br>This field is new in the Version 2.1.x encoder. It has the following defined values:<br>'0' Use the AIM-standard algorithm to automatically determine the encoding mode.<br>'1' Use the alternate UPS algorithm to automatically determine the encoding mode.<br>'2' Force the encoder to use Mode 2 to encode the symbol.<br>'3' Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

**NOTE:** The secondary message maximum length of 126 characters applies only in the case in which the entire secondary message is composed of digits, a situation which does not conform to the UPS standards. This length is set to this value in order to allow the Maxicode encoder to handle all possible encoding conditions, including those for non-UPS applications. In UPS applications, the limit is 80 characters or less.

### *4.2.4 Encoding Structured Carrier Message Symbols*

The Maxicode encoder provides support for encoding Structured Carrier Messages in a variety of manners:

- Via UPS-specific functions.
  This is the simplest way to encode a Maxicode symbol for UPS use. All internal formatting is handled automatically by the function.
- Via String functions..
  If your application is already constructing Structured Carrier Message strings, these functions will accept this as input. These functions are not UPS-specific, but will generate UPS-compatible output if the string input is formatted properly.
- Via generic Structured Carrier Message (SCM) functions.
  These functions are the most general purpose and powerful. They permit the generation of Structured Message Append symbols, but require the secondary message to be properly formatted as discussed in **Primary and Secondary Message Formats** on page 13. As with the string functions, these functions are not UPS-specific, but will generate UPS-compatible output if the input is formatted properly.
- Via the "Compressed Maxicode" functions.

In addition, the Maxicode encoder provides two versions of most API functions. One version assumes that the input data is in the native character set of the local computer system (ASCII or EBCDIC). These functions automatically perform ANSI EBCDIC-to-ASCII conversion on their input data if the native character set is EBCDIC. The other assumes that its input is in ASCII, regardless of whether the local computer system is based on ASCII or EBCDIC, and thus performs no character translations, even on EBCDIC machines. Normally, the native functions are the simplest to use. The ASCII functions are provided to handle the case where binary information is encoded that would be improperly handled by an internal EBCDIC-to-ASCII

conversion, or where an EBCDIC-to-ASCII conversion other than the ANSI standard one is required..

The API functions available are as follows:

| Function | Input form | Character Set |
|---|---|---|
| MaxUpsAC | UPS Structure | ASCII |
| MaxUpsNC | UPS Structure | Native |
| MaxScmAC | SCM Structure | ASCII |
| MaxScmNC | SCM Structure | Native |
| MaxStrAC | SCM String | ASCII |
| MaxStrNC | SCM String | Native |
| MaxCmpAC | Compressed Data Structure | ASCII |

#### 4.2.4.1 UPS-Specific Functions

The UPS-specific functions are listed below. These functions take their input in a structure that contains all the mandatory and optional fields for a UPS-specific Maxicode symbol. These routines then automatically perform the low-level message formatting required. These routines provide the easiest-to-use method of integrating UPS Maxicode support into an application.

Prototypes:

```
#include "maxiapi.h"      /* prototypes and structures */

void MaxUpsAC(MAXIUPSINFOCPTR pInput, MAXIFONTOUTPTR pOutput);

void MaxUpsNC(MAXIUPSINFOCPTR pInput, MAXIFONTOUTPTR pOutput);
```

Arguments:

| Name | Use |
|---|---|
| pInput | Pointer to a MAXIUPSINFO structure containing the input information |
| pOutput | Pointer to a MAXIFONTOUT structure to receive the output |

Return Value:

```
None
```

Notes:

- No support is provided for Structured Message Append using the UPS-specific functions. If the presented data does not fit into a single symbol, the result code MERR_TOO_LONG will be returned. To break information up across two or more symbols, the SCM functions must be used.
- MaxUpsNC assumes the contents of the input structure are in the native character set, and performs EBCDIC to ASCII conversions automatically as required.

Example (The following code segment is from the sample program CUPS)

```
MAXIUPSINFO maxiRecord;
MAXIFONTOUT maxiOutput;
```

```
/*  Initialize the UPS record  */
maxiRecord.countryCode = 840;
maxiRecord.serviceClass = 1;
maxiRecord.julianDayOfPickup = 34;
maxiRecord.packageNumber = 1;
maxiRecord.packageCount = 1;
maxiRecord.packageWeight = 20;   /* Note: in 10ths of a pound */
strcpy(maxiRecord.postalCode, "33901");
strcpy(maxiRecord.trackingNumber, "1Z34567890");
strcpy(maxiRecord.shipperNumber, "102562");
strcpy(maxiRecord.shipmentID, "");
strcpy(maxiRecord.shipToAddress, "2201 SECOND ST");
strcpy(maxiRecord.shipToCity, "FT MYERS");
strcpy(maxiRecord.shipToState, "FL");
maxiRecord.addressValidation = 'Y';
maxiRecord.modeControl = '0';

/*  Call the encoder  */
MaxUpsNC(&maxiRecord, &maxiOutput);

if ( maxiOutput.resultCode == MERR_OK )
{
    PrintSymbol("CUPS", &maxiOutput);
}
else
{
    printf("encoding error %d\n", (int)maxiOutput.resultCode);
}
```

#### 4.2.4.2 String Functions

The string functions take as their input pointers to a pre-formatted Structured Carrier Message string, along with the length of that string. The routines encode the contents of this string into the Maxicode symbol. Use these functions if your application is already formatting Structured Carrier Messages that you want to represent in a Maxicode symbol.

Prototype:

```
#include "maxiapi.h"      /* prototypes and structures */

void MaxStrAC( const char MAXIPTR *pInput,
               int inputLen,
               MAXIFONTOUTPTR pOutput);

void MaxStrNC( const char MAXIPTR *pInput,
               int inputLen,
               MAXIFONTOUTPTR pOutput);
```

Arguments:

| Name | Use |
|---|---|
| pInput | Pointer to the pre-formatted Structured Carrier Message string |
| inputLen | Length of the input string. If the input string is properly terminated with an EOT character, you may pass a length of zero, and the routines will automatically determine the length. |
| pOutput | Pointer to a MAXIFONTOUT structure to receive the output |

Return Value:

None

Notes:

- No support is provided for Structured Message Append using the String functions. If the presented data does not fit into a single symbol, the result code MERR_TOO_LONG will be returned. To break information up across two or more symbols, the SCM functions must be used.
- MaxStrNC assumes the contents of the input string are in the native character set, and performs EBCDIC to ASCII conversions automatically as required.
- An example of a pre-formatted Structured Carrier Message string is shown below:

  []>$^{R}_{S}$01$^{G}_{S}$96123456789$^{G}_{S}$840$^{G}_{S}$001$^{G}_{S}$1Z12345678$^{G}_{S}$UPSN$^{R}_{S}$$^{E}$0$_{T}$

Example (The following code segment is from the sample program CSTR. Note that it will compile and run on both EBCDIC and ASCII machines without any modifications)

```
#define NATIVE_RS       0x1e
#define NATIVE_GS       0x1d
#if 'A' == 0x41
#define NATIVE_EOT      0x04  /*  ASCII  */
static char scmHeader[9] = { 0x5B, 0x29, 0x3E, NATIVE_RS,
                             0x30, 0x31, NATIVE_GS,
                             0x39, 0x36 };
#else
#define NATIVE_EOT      0x37  /*  EBCDIC  */
static char scmHeader[9] = { 0x4a, 0x5d, 0x6e, NATIVE_RS,
                             0xf0, 0xf1, NATIVE_GS,
                             0xf9, 0xf6 };
#endif
char scmString[256];
int offset;

/*  Format message header  */
    offset = sprintf(scmString,
                     scmHeader);

/*  Append the primary message fields  */
offset += sprintf(scmString + offset,
                  "%s%c%03d%c%03d%c",
                  "33901", NATIVE_GS,
                  840, NATIVE_GS,
                  1, NATIVE_GS);

/*  Append the remaining mandatory fields  */
offset += sprintf(scmString + offset,
                  "%s%cUPSN%c",
                  "1Z34567890", NATIVE_GS,
                  NATIVE_GS);

/*  shipper #, day of pickup, shipment ID  */
offset += sprintf(scmString + offset,
                  "%s%c%03d%c%s%c",
                  "102562", NATIVE_GS,
                  34, NATIVE_GS,
                  "", NATIVE_GS);

/*  N/X, weight, address validation  */
```

```
offset += sprintf(scmString + offset,
                  "%d/%d%c%d%c%c%c",
                  1, 1, NATIVE_GS,
                  2, NATIVE_GS,
                  'Y', NATIVE_GS);

/*  address, city, state  */
offset += sprintf(scmString + offset,
                  "%s%c%s%c%s%c%c",
                  "2201 SECOND ST", NATIVE_GS,
                  "FT MYERS", NATIVE_GS,
                  "FL", NATIVE_RS, NATIVE_EOT);

/*  Call the encoder  */
MaxStrNC(scmString, (int)strlen(scmString), &maxiOutput);

if ( maxiOutput.resultCode == MERR_OK )
{
    PrintSymbol("CSTR", &maxiOutput);
}
else
{
    printf("encoding error %d\n", (int)maxiOutput.resultCode);
}
```

#### 4.2.4.3 Generic Structured Carrier Message (SCM) Functions

The SCM functions take as their input a MAXISCM structure. The contents of this structure reflect the low-level formatting of a Structured Carrier Message in a Maxicode, and require that the user perform the proper formatting of the Secondary Message portion of the message. (See **Primary and Secondary Message Formats** on page 13). These routines also provide support for Structured Message Append. Use these functions if you are preparing a Structured Carrier Message for a carrier other than UPS and it is not convenient to use the String functions, or in situations in which it is necessary to split a message across two or more symbols.

Prototypes:

```
#include "maxiapi.h"      /* prototypes and structures */

void MaxScmAC( MAXISCMPTR pSCM, MAXIFONTOUTPTR pOutput);

void MaxScmNC( MAXISCMPTR pSCM, MAXIFONTOUTPTR pOutput);
```

Arguments:

| Name | Use |
|---|---|
| pSCM | Pointer to the MAXISCM structure containing the input |
| pOutput | Pointer to a MAXIFONTOUT structure to receive the output |

Return Value:

```
None
```

Notes:

- MaxScmNC assumes the contents of the input structure are in the native character set, and performs EBCDIC to ASCII conversions automatically as required.

- See **Structured Message Append** on page 78 for more detail on building Structured Message Append symbols.

Example (The following code segment is from the sample program CSCM)

```
#define NATIVE_RS       0x1e
#define NATIVE_GS       0x1d
#if 'A' == 0x41
#define NATIVE_EOT      0x04  /*  ASCII  */
static char scmHeader[9] = { 0x5B, 0x29, 0x3E, NATIVE_RS,
                             0x30, 0x31, NATIVE_GS,
                             0x39, 0x36 };
#else
#define NATIVE_EOT      0x37  /*  EBCDIC  */
static char scmHeader[9] = { 0x4a, 0x5d, 0x6e, NATIVE_RS,
                             0xf0, 0xf1, NATIVE_GS,
                             0xf9, 0xf6 };

MAXISCM     maxiScm;
MAXIFONTOUT maxiOutput;
int offset;

maxiScm.symbolNum = 1;
maxiScm.symbolCount = 1;
memcpy(maxiScm.postalCode, "33901", 5);
maxiScm.postalCodeLen = 5;
maxiScm.countryCode = 840;
maxiScm.serviceClass = 1;
maxiScm.symbolNum = 1;

/*  Format message header  */
offset = sprintf(maxiScm.secondaryMsg,
                 scmHeader);

/*  SCAC and tracking #  */
offset += sprintf(maxiScm.secondaryMsg + offset,
                  "%s%cUPSN%c",
                  "1Z34567890", NATIVE_GS,
                  NATIVE_GS);

/*  shipper #, day of pickup, shipment ID  */
offset += sprintf(maxiScm.secondaryMsg + offset,
                  "%s%c%03d%c%s%c",
                  "102562", NATIVE_GS,
                  34, NATIVE_GS,
                  "", NATIVE_GS);

/*  N/X, weight, address validation  */
offset += sprintf(maxiScm.secondaryMsg + offset,
                  "%d/%d%c%d%c%c%c",
                  1, 1, NATIVE_GS,
                  2, NATIVE_GS,
                  'Y', NATIVE_GS);

/*  address, city, state  */
offset += sprintf(maxiScm.secondaryMsg + offset,
                  "%s%c%s%c%s%c%c",
                  "2201 SECOND ST", NATIVE_GS,
                  "FT MYERS", NATIVE_GS,
                  "FL", NATIVE_RS, NATIVE_EOT);

maxiScm.secondaryMsgLen = offset;
```

```
/*  Call the encoder  */
MaxScmNC(&maxiScm, &maxiOutput);

if ( maxiOutput.resultCode == MERR_OK )
{
    PrintSymbol("CSCM", &maxiOutput);
}
else
{
    printf("encoding error %d\n", (int)maxiOutput.resultCode);
}
```

#### 4.2.4.4 Compressed Data Function

The function for dealing with UPS-compressed data is listed below. This function take its input in a structure that contains all the required for a UPS-specific Maxicode symbol using compressed data. This routine then automatically performs the low-level message formatting required.

Prototypes:

```
#include "maxiapi.h"      /* prototypes and structures */

void MaxCmpAC(MAXICOMPRESSEDINFOCPTR pCompressed, MAXIFONTOUTPTR pOutput);
```

Arguments:

| Name | Use |
|---|---|
| pInput | Pointer to a MAXICOMPRESSEDINFO structure containing the input information |
| pOutput | Pointer to a MAXIFONTOUT structure to receive the output |

Return Value:

```
None
```

Example:

```
unsigned char upsCompressedData[64];
unsigned int upsCompressedDataLen;
MAXICOMPRESSEDINFO maxiRecord;
MAXIFONTOUT maxiOutput;

/*  Use the UPS compression routines to input and compress the package  */
/*  data.  Assume the resulting data ends up in the upsCompressedData[] */
/*  array, and the byte count in the upsCompressedDataLen variable.     */

/*  Initialize the input record  */
maxiRecord.countryCode = 840;
maxiRecord.serviceClass = 1;
strcpy(maxiRecord.postalCode, "339010000");
strcpy(maxiRecord.trackingNumber, "1Z34567890");
strcpy(maxiRecord.shipperNumber, "102562");
memcpy(maxiRecord.compressedData, upsCompressedData, upsCompressedDataLen);
maxiRecord.compressedDataLen = upsCompressedDataLen;

/*  Call the encoder  */
MaxCmpAC (&maxiRecord, &maxiOutput);

if ( maxiOutput.resultCode == MERR_OK )
{
    PrintSymbol(&maxiOutput);
```

```
}
else
{
    printf("encoding error %d\n", (int)maxiOutput.resultCode);
}
```

Notes:

- As of this writing, the UPS compression DLL typically returns exactly 48 bytes of data from the compression operation.
- The postal code, country code and service class must be provided both to the UPS compression routines and to the encoder.

### *4.2.5 Encoding Generic Message (Non-Structured Carrier Message) Data*

Although Maxicode was designed primarily for representing Structured Carrier Message data sets, it can be used to encode generic unstructured data as well, using modes 4 and 5, as discussed in **Internal Encoding Details** on page 8. The Maxicode encoder provides four functions to allow the creation of Maxicode symbols that do not follow the Structured Carrier Message format. Two levels of error correction are supported – Standard Error Correction, corresponding to Mode 4, and Extended Error Correction, corresponding to Mode 5. Extended Error Correction provides better protection against symbol damage, at the expense of allowing fewer characters to be encoded in a symbol. As with the Structured Carrier Message functions, ASCII and Native versions are provided for each function.

The API functions available are as follows:

| Function | Error Correction | Character Set |
|---|---|---|
| MaxSecAC | Standard | ASCII |
| MaxSecNC | Standard | Native |
| MaxEecAC | Extended | ASCII |
| MaxEecNC | Extended | Native |

Prototypes:

```
#include "maxiapi.h"      /* prototypes and structures */

void MaxSecAC(const char MAXIPTR *pInput, int inputLen,
              int symbolNum, int symbolCount,
              MAXIFONTOUTPTR pOutput);

void MaxSecNC(const char MAXIPTR *pInput, int inputLen,
              int symbolNum, int symbolCount,
              MAXIFONTOUTPTR pOutput);

void MaxEecAC(const char MAXIPTR *pInput, int inputLen,
              int symbolNum, int symbolCount,
              MAXIFONTOUTPTR pOutput);

void MaxEecNC(const char MAXIPTR *pInput, int inputLen,
              int symbolNum, int symbolCount,
              MAXIFONTOUTPTR pOutput);
```

Arguments:

| Name | Use |
|---|---|
| pInput | Pointer to the data to be encoded. |
| inputLen | Length of the data to be encoded. |
| symbolNum | The "N" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| symbolCount | The "X" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| pOutput | Pointer to a MAXIFONTOUT structure to receive the output |

Return Value:

None

Notes:

- MaxSecNC and MaxEecNC assume the contents of the input data are in the native character set, and perform EBCDIC to ASCII conversions automatically as required.

Example (The following code segment is from the sample program CEEC):

```
MAXIFONTOUT maxiOutput;
char *eecMsg = "This is a message encoded using mode 5 (EEC).";

/*  Call the encoder  */
MaxEecNC(eecMsg,
         (int)strlen(eecMsg),
         1,
         1,
         &maxiOutput);

if ( maxiOutput.resultCode == MERR_OK )
{
    PrintSymbol("CEEC", &maxiOutput);
}
else
{
    printf("encoding error %d\n", (int)maxiOutput.resultCode);
}
```

## 4.3 Visual Basic Language API

### *4.3.1 Initialization*

Before any of the encoding API functions are called, the encoder may be initialized. This process is typically not necessary in a Windows environment, since no EBCDIC-to-ASCII conversion issues are involved.

Prototype:

```
Private Declare Sub MaxInitC Lib "maxicode2011.dll"
       (ByRef mxInput As MaxiInitInput)
```

Return Value:

```
None
```

Notes:

- This function does not need to be called before every symbol encoded. It only needs to be called once as part of program initialization.
- If this function is not called, the default values indicated below are used. These default values are generally appropriate for a Windows environment.

The structure used by this function has the following definition:

```
Private Type MaxiInitInput
    noHexChar as Byte
    loHexChar as Byte
    hiHexChar as Byte
    twoHexChar as Byte
    padChar as Byte
    bullChar as Byte
End Type
```

The elements of this structure are as follows:

| Element | Contents |
|---|---|
| noHexChar | The "full width blank" character. |
| lohexChar | The character containing the second row hexagon. |
| hiHexChar | The character containing the first row hexagon. |
| twoHexChar | The character containing both hexagons. |
| padChar | The "narrow blank" character. |
| bullChar | The character containing the bullseye character. |

Default values output by the encoder (in hexadecimal) are:

| Element | ASCII |
|---|---|
| noHexChar | 0x30 |
| loHexChar | 0x31 |
| hiHexChar | 0x32 |
| twoHexChar | 0x33 |
| padChar | 0x34 |
| bullChar | 0x35 |

### *4.3.2 Result Codes*

The Maxicode encoder API functions indicate success or failure with a return code of type MAXIERROR. The following return codes are defined:

| Value | Meaning |
|---|---|
| 0 | Success |
| 1 | An invalid parameter was passed. This rarely occurs in a Visual Basic program. |
| 2 | The data passed would not fit in a single Maxicode symbol. |
| 3 | The postal code was invalid. Possible causes include:<br>• The postal code is longer than 9 digits, or 6 alphanumeric characters<br>• The postal code contains characters other than digits and upper-case alphabetic characters<br>For country code 840 (USA), the postal code is neither 5 nor 9 characters in length |
| 4 | The country code was invalid. Possible causes include:<br>• The country code was longer than three digits (string) or greater than 999. |
| 5 | The service class was invalid. Possible causes include:<br>• The service class was longer than three digits (string) or greater than 999. |
| 6 | The tracking number field of the UPS structure was invalid (wrong length) |
| 7 | The address validation field of the UPS structure was invalid. Valid values are True or False |
| 8 | The shipToState field of the UPS structure was invalid. It must be either empty or two characters in length. |
| 9 | The package number and package count fields are incorrect or inconsistent. Possible causes include:<br>• One of the fields is zero and the other is not<br>• The package number is greater than the package count |
| 10 | The symbol number or symbol count fields are incorrect or inconsistent. Possible causes include:<br>One of the fields is less than 1 or greater than 8.<br>The symbol number is larger than the symbol count |
| 11 | The weight is invalid. Typically caused by a weight greater than 999.9 pounds. |

In addition to these values, a number of "internal" error codes are defined. All these have a value of 100 or greater. Should you encounter an internal error code, please note the code and contact Silver Bay Software LLC technical support.

Note that the degree to which the encoder API validates postal codes, country codes and service classes is limited to ensuring that the data passed can be encoded into a Maxicode label (i.e. is the correct length, and is of the correct type – digits or alphanumeric). The encoder has no way to ensure that the actual data is valid.

### *4.3.3 Data Structures*

This section describes the various user defined types used for input and output with the encoder library.

#### 4.3.3.1 Output Structure

All the Maxicode encoder API functions return their output in a structure of the following form:

```
Private Type MaxicodeOutput
       resultCode As Integer
       outputLine(16) As String
End Type
```

The elements of this structure are as follows:

| Element | Contents |
|---|---|
| `output` | This is the output data. This area consists of 17 strings of 30 characters. |
| `resultCode` | This code indicates the success or failure of the operation. See **Result Codes** on page 63 for a description of the values. |

#### 4.3.3.2 UPS-Specific Input Structure

The following structure is used as input to the UPS-specific encoder routines documented on page 68:

```
Private Type MaxicodeInput
       countryCode As Long
       serviceClass As Long
       julianDayOfPickup As Long
       packageNumber As Long
       packageCount As Long
       packageWeight As Long
       postalCode As String
       trackingNumber As String
       shipperNumber As String
       shipmentID As String
       shipToAddress As String
       shipToCity As String
       shipToState As String
       addressValidation As Boolean
       modeControl As String
End Type
```

The mandatory elements of this structure are as follows:

| Element | Contents |
|---|---|
| `countryCode` | The country code for the destination country.<br>(USA's country code is 840) |
| `serviceClass` | The "class of service" code assigned by UPS. |

| | |
|---|---|
| `postalCode` | The destination postal code represented as a string. For all-numeric postal codes, this may be up to nine digits.<br>If the country code is set to 840 (USA), this field must be numeric and exactly 5 or 9 digits in length.<br>For postal codes containing alphabetic characters (which must be in upper case), the limit is six characters. |
| `trackingNumber` | The 10-character UPS tracking number for the item. |

The optional elements of the structure are shown below. To omit a numeric item, set it to zero. To omit a string item, pass an empty string.

| Element | Max Len | Contents |
|---|---|---|
| `julianDayOfPickup` | N/A | The numeric day of the year in which the package was picked up. (Jan 1 = 1, Jan 31 = 31, Feb 1 = 32, Feb 28 = 57, etc) |
| `packageNumber` | N/A | The "N" in "Package N of X" |
| `packageCount` | N/A | The "X" in "Package N of X" |
| `packageWeight` | N/A | The weight of the package in tenths of a pound. Thus, a 5-pound package would have the value 50. |
| `shipperNumber` | 6 | The UPS-assigned number for the shipper (String) |
| `shipmentID` | 30 | The shipper-assigned identification number for the shipment (String, up to 30 characters) |
| `shipToAddress` | 35 | The destination street address (String, up to 35 characters) |
| `shipToCity` | 35 | The destination city (String, up to 35 characters) |
| `shipToState` | 2 | The destination state abbreviation (2 character string) |
| `addressValidation` | N/A | True or False |
| `modeControl` | 1 | Encoder mode control<br>This field is new in the Version 2.1.x encoder. It has the following defined values:<br>"0" Use the AIM-standard algorithm to automatically determine the encoding mode.<br>"1" Use the alternate UPS algorithm to automatically determine the encoding mode.<br>"2" Force the encoder to use Mode 2 to encode the symbol.<br>"3" Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

**NOTE:** If all the information listed in these tables is provided for an individual package, it is quite possible that the data will not fit into a single symbol. After encoding the message header, mandatory information, and the message terminator, a Maxicode symbol only has space for a maximum of 53 characters of optional information. This includes the $^{G}_{S}$ characters separating the fields. As a result, a single symbol can hold only a maximum of 43 characters of optional information.

Input parameter validation is limited to ensuring that the data passed in will fit within a Maxicode barcode. The data tests are thus typically limited to ensuring that individual fields are not too big to encode.

#### 4.3.3.3 SCM-Specific Input Structure

The following structure is used as input to the SCM encoder routines documented on page 68:

```
Private Type MaxicodeScmInput
       symbolNum As Long
       symbolCount As Long
       countryCode As Long
       serviceClass As Long
       postalCode As String
       secondaryMsg As String
       modeControl As String
End Type
```

The elements of this structure are as follows:

| Element | Max Len | Contents |
|---|---|---|
| `symbolNum` | N/A | The "N" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| `symbolCount` | N/A | The "X" in "Symbol N of X". If only one symbol is being generated, this should be set to 1. |
| `postalCode` | 9 | The destination postal code represented as a string. For all-numeric postal codes, this may be up to nine digits.<br>If the country code is set to 840 (USA), this field must be exactly 5 or 9 digits in length.<br>For postal codes containing alphabetic characters (which must be in upper case), the limit is six characters. |
| `countryCode` | N/A | The destination country code (USA = 840) |
| `serviceClass` | N/A | The class of service being used. |
| `secondaryMsg` | 126 | The secondary message field itself. For UPS applications, this field must be formatted as described in **Primary and Secondary Message Formats** on page 13. |
| `modeControl` | 1 | Encoder mode control<br>This field is new in the Version 2.1.x encoder. It has the following defined values:<br>"0" Use the AIM-standard algorithm to automatically determine the encoding mode.<br>"1" Use the alternate UPS algorithm to automatically determine the encoding mode.<br>"2" Force the encoder to use Mode 2 to encode the symbol.<br>"3" Force the encoder to use Mode 3 to encode the symbol.<br>See the section entitled **Maxicode Encoding Modes** on page 10 for more details. |

**NOTE:** The secondary message maximum length of 126 characters applies only in the case in which the entire secondary message is composed of digits, a situation which does not conform to the UPS standards. This length is set to this value in order to allow the Maxicode encoder to

handle all possible encoding conditions, including those for non-UPS applications. In UPS applications, the limit is 80 characters or less.

#### 4.3.3.4 Compressed Data Input Structure

The following structure is used as input to the SCM encoder routines documented on page 56:

```
Private Type MaxicodeScmInput
    countryCode As Long
    serviceClass As Long
    postalCode As String
    trackingNumber As String
    shipperNumber As String
    compressedData As String
    compressedDataLen As Long
End Type
```

The elements of this structure are as follows:

| Element | Max Len | Contents |
|---|---|---|
| `postalCode` | 9 | The destination postal code represented as a string. For all-numeric postal codes, this may be up to nine digits.<br><br>If the country code is set to 840 (USA), this field must be exactly 5 or 9 digits in length.<br><br>For postal codes containing alphabetic characters (which must be in upper case), the limit is six characters. |
| `countryCode` | N/A | The destination country code (USA = 840) |
| `serviceClass` | N/A | The class of service being used. |
| `shipperNumber` | 6 | The UPS-assigned number for the shipper (String) |
| `trackingNumber` | 10 | The 10-character UPS tracking number for the item. |
| `compressedData` | 52 | The compressed data from the UPS proprietary compression DLL. Typically this is 48 characters in length. |
| `compressedDataLen` | N/A | The number of bytes of data in the compressed string. |

### *4.3.4 Encoding Structured Carrier Message Symbols*

The Maxicode encoder provides support for encoding Structured Carrier Messages in a variety of manners:

- Via the UPS-specific function.
  This is the simplest way to encode a Maxicode symbol for UPS use. All internal formatting is handled automatically by the function.
- Via generic Structured Carrier Message (SCM) functions.
  These functions are the most general purpose and powerful. They permit the generation of Structured Message Append symbols, but require the secondary message to be properly formatted as discussed in **Primary and Secondary Message Formats** on page 13. As with the string functions, these functions are not UPS-specific, but will generate UPS-compatible output if the input is formatted properly.
- Via the "Compressed Maxicode" functions.

The API functions available are as follows:

| Function | Input form |
|---|---|
| MaxUpsVB | UPS Structure |
| MaxCmpVB | Compressed Data Structure |

#### 4.3.4.1 UPS-Specific Function

The UPS-specific function is listed below. This function takes its input in a structure that contains all the mandatory and optional fields for a UPS-specific Maxicode symbol. This routine then automatically performs the low-level message formatting required. This routine provides the easiest-to-use method of integrating UPS Maxicode support into an application.

API:

```
Private Declare Sub MaxUpsVB Lib "maxicode2011.dll"
       (ByRef mxInput As MaxicodeInput, ByRef mxOutput As MaxicodeOutput)
```

Example:

```
Dim mxInput As MaxicodeInput
Dim mxOutput As MaxicodeOutput

mxInput.countryCode = 840
mxInput.serviceClass = 1
mxInput.trackingNumber = "1Z12345670"
mxInput.shipperNumber = "123456"
mxInput.julianDayOfPickup = 12
mxInput.packageCount = 1
mxInput.packageNumber = 1
mxInput.packageWeight = 10
mxInput.postalCode = "339120000"
mxInput.shipToCity = "Fort Myers"
mxInput.shipToState = "FL"
mxInput.addressValidation = False
mxInput.modeControl = "0"

Call MaxUpsVB(mxInput, mxOutput)

If mxOutput.resultCode = 0 Then
```

```
        For i = 0 To 16
            'Print mxOutput.outputLine(i) via the appropriate method
        Next i
    Else
        'an error has occurred
    End If
```

#### 4.3.4.2 Generic Structured Carrier Message (SCM) Functions

The SCM function takes as its input a `MaxicodeScmInput` user-defined type. The contents of this structure reflect the low-level formatting of a Structured Carrier Message in a Maxicode, and require that the user perform the proper formatting of the Secondary Message portion of the message. (See **Primary and Secondary Message Formats** on page 13). This routine also provides support for Structured Message Append. Use this function if you are preparing a Structured Carrier Message for a carrier other than UPS, or in situations in which it is necessary to split a message across two or more symbols.

API:

```
Private Declare Sub MaxScmVB Lib "maxicode2011.dll"
      (ByRef mxInput As MaxicodeScmInput, ByRef mxOutput As MaxicodeOutput)
```

Notes:

- See **Structured Message Append** on page 78 for more detail on building Structured Message Append symbols.

#### 4.3.4.3 Compressed-Data Function

The Compressed Data function is listed below. This function takes its input in a structure that contains the output from the UPS proprietary compression DLL, as well as other information that is required to generate the Maxicode symbol. The encoder takes the data provided and performs the low-level formatting required to produce a printable Maxicode symbol.

API:

```
Private Declare Sub MaxCmpVB Lib "maxicode2011.dll"
      (ByRef mxInput As MaxiCompInput, ByRef mxOutput As MaxicodeOutput)
```

Example:

```
Dim mxInput As MaxiCompInput
Dim mxOutput As MaxicodeOutput
Dim upsCompressedData as String
Dim upsCompressedDataLen as Long

'  Use the UPS compression routines to input and compress the package
'  data.  Assume the resulting data ends up in the upsCompressedData
'  string, and the byte count in the upsCompressedDataLen variable.

'  Initialize the input record
mxInput.countryCode = 840
mxInput.serviceClass = 1
mxInput.postalCode =  "339010000"
mxInput.trackingNumber =  "1Z34567890"
mxInput.shipperNumber = "102562"
mxInput.compressedData = upsCompressedData
mxInput.compressedDataLen = upsCompressedDataLen

/*  Call the encoder  */
```

```
Call MaxCmpVB(mxInput, mxOutput);

If maxiOutput.resultCode = 0 Then

    For I = 0 to 16
        'Print mxOutput.outputLine(i) via the appropriate method
    Next I
Else
    ' Handle error
End If
```

**Notes:**

- As of this writing, the UPS compression DLL typically returns exactly 48 bytes of data from the compression operation.
- The postal code, country code and service class must be provided both to the UPS compression routines and to the encoder.

# 5 Printing the Maxicode Symbol

When using the Silver Bay Software Maxicode encoder, a generated Maxicode symbol is printed using a special font. This font consists of characters representing hexagons and spaces, as well as a character that prints the bull's-eye at the center of the symbol. It is the responsibility of the application programmer to generate the appropriate print stream data to invoke the font on the printer and to send the characters returned by the encoder to the printer.

## 5.1 The Maxicode Font

An individual Maxicode symbol is composed of 33 rows of hexagons with the bull's-eye pattern in the center. The first row (and all other "odd" rows) contains 30 hexagons. The second row (and all other "even" rows) contains 29 hexagons. Even rows are offset a half hexagon to the right with respect to odd rows.

The Maxicode encoder generates the symbol using 17 rows of characters, 30 characters per row. Each character represents two hexagons, one from an odd row and one from the even row below it. The characters used are diagrammed below (not to scale):

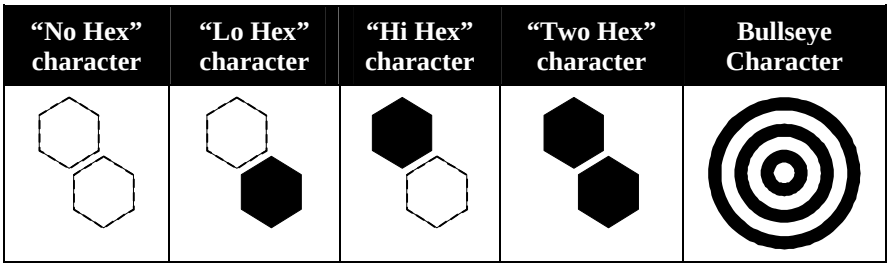



**Figure 6 - Contents of Custom Maxicode Font**

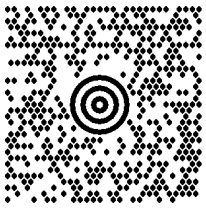
Outlined (unfilled) hexagons in the above diagrams indicate the position of white hexes. These areas are actually completely white in the font – there is no outline drawn. Only six code points (characters) are defined in the Maxicode font – all other code points are unused. The fonts use the following characters as their code points:

| Character | Code Point | ASCII | EBCDIC |
|---|---|---|---|
| No Hex | '0' | 0x30 | 0xF0 |
| Lo Hex | '1' | 0x31 | 0xF1 |
| Hi Hex | '2' | 0x32 | 0xF2 |
| Two Hex | '3' | 0x33 | 0xF3 |
| Pad | '4' | 0x34 | 0xF4 |
| Bullseye | '5' | 0x35 | 0xF5 |

If you were to examine the output of the Maxicode encoder, the actual character output of the encoder looks something like this:

```
230303230331222231233222212222
331220321012103131311331133320
103223110030320012023121302010
222232322312222323131221322222
111230121331012200212131333030
013212320222000001013300013232
222222310000000000121202201012
120120230000000000021100222032
110020103500000000003023331130
303110011100000000101030021120
002012320020000000020110003020
222132110031322003111001301210
313310020202020202020100330130
130310213121202130311302121100
233332300112003211113112130220
111031202110230033010231310310
202002202020022020000020200200
```

Note there are 17 rows, 30 characters in each row. When printed using the custom Maxicode font, the hexagons and bull's-eye of the Maxicode symbol are printed:



## 5.2 Vertical Spacing

The horizontal positioning of the Maxicode characters within a single line is controlled by information within the font. That is, as each character of a line is rendered by the printer, the position of the adjacent character is automatically determined by the printer. However, *the line-to-line spacing must be controlled by the programmer*.

Line spacing can be expressed in a couple of ways. For example, many printers use lines per inch. The Maxicode font needs to be printed at 16.5 lines per inch. However, most printing technologies will not let you specify 16.5 as a valid lines-per-inch setting.

The following sections provide suggestions on how to perform font selection and proper vertical spacing in different printing environments.

## 5.3 Using Hewlett-Packard PCL Fonts

### *5.3.1 Overview*

Before a Maxicode symbol can be printed on an HP-PCL printer, the custom Maxicode font must first be installed. The distribution kit includes two HP-PCL soft fonts; one for portrait orientation and another for landscape. Consult the platform-specific documentation that came with your encoder for the locations and names of the font files.

Soft fonts are stored in the printer's RAM; thus when the printer is turned off or is reset, soft fonts are lost and must be re-installed. We recommend that your application program download the software before each print run. Downloading a soft font involves assigning a Font ID and then sending the font to the printer.

Thus, the steps typically associated with using HP-PCL fonts are as follows:

Step 1: As part of the beginning of your job, include the contents of the appropriate font as part of the binary stream sent to the printer. Typically, you will precede the font with the HP-PCL escape sequence to assign the font an ID of your choosing. For example, if you chose to use a font ID of 12, you would send the sequence:

$^{E}_{C}$*c*nn*D (where $^{E}_{C}$ is the ASCII ESC character, and *nn* is the font ID)

followed by the contents of the font. The font is binary information, so it is important to ensure that no EBCDIC-to-ASCII conversion occurs on this data in the path between the system and the printer.

Step 2: Map the font ID specified for the Maxicode font as the PCL secondary font

$^{E}_{C}$)*nn*X (where $^{E}_{C}$ is the ASCII ESC character, and *nn* is the font ID)

Step 3: When it is time to print the Maxicode symbol, invoke the secondary font using the ASCII $^{S}_{O}$ (Shift Out) character.

Step 4: Print the characters associated with the Maxicode symbol. Each individual line of the symbol must be positioned 0.060 inches below the previous line, thus achieving a spacing of 16.66 lines per inch (since HP-PCL is based on 300dpi units, this is as close to the nominal 16.5 LPI that can be achieved). Each individual line may be positioned using the Horizontal Cursor Positioning and Vertical Cursor Positioning escape sequence:

$^{E}_{C}$*p*xxx*X$^{E}_{C}$*p*yyy*Y (where *xxx* and *yyy* are the horizontal and vertical positions of the individual line expressed in "PCL Units")

Recall that HP-PCL uses 300dpi "PCL Units" when positioning, so the *yyy* value will increase by 18 for each successive line.

Step5: After the symbol has been completely printed, return to the primary font using the ASCII $^{S}_{I}$ (Shift In) character to print the remainder of the page.

There are obviously other ways that can be used to select a particular font at the appropriate point in the print stream – the above is included as one example. Consult the *PCL Printer Language Technical Reference Manual*, available from Hewlett-Packard, for more details on the use of HP-PCL soft fonts.

Note that unless a "reset" escape sequence is sent to the printer during the print stream, it is only necessary to download the font to the printer once at the beginning of the print job.

### *5.3.2 C Language Example*

The following code segment downloads the portrait font as soft font number 100 (any number between 1 and 32767 can be used). This code fragment assumes that `fpPrinter` is the previously opened output stream:

```
char buffer[1024];
FILE *fpPrinter
FILE *fpFont;
int nChars;

...

/*  Select font ID 100  */
fprintf(fpPrinter, "\033*c100D");

/*  Open the MAXICODE font file  */
fpFont = fopen("MAXHP3P", "r");

/*  Copy the file to our output (the printer)  */
for(;;)
{
    nChars = fread(buffer, 1, sizeof(buffer), fpFont);
    if (nChars == 0)
        break;
    fwrite(buffer, 1, nChars, fpPrinter);
}
fclose(fpFont);

/*  Make the font permanent so printer reset leaves it alone  */
fprintf(fpPrinter, "\033*c100d5F");

/*  Set the font as secondary  */
fprintf(fpPrinter, "\033)100X");
```

This sample makes the font "permanent" so that if a printer reset is issued ($^{E}{}_{C}E$) the font will remain in printer memory. We also make the font the secondary font in the printer; this way the Maxicode font can be easily selected by issuing an $^{S}{}_{O}$ character; the primary font can then be restored by issuing an $^{S}{}_{I}$ character. As mentioned earlier, this is only one mechanism for selecting fonts; refer to your HP PCL Reference manual for additional information regarding font selection.

Once the font has been loaded, the steps to printing the Maxicode symbol are as follows:

1. Activate the Maxicode font.
2. Loop through the 17 lines of output characters, setting the print position for each line and then sending the characters.
3. Switch back to the primary font.

The following code segment demonstrates this. Since HP soft fonts are 300 dpi fonts, a line spacing of 18 pixels is used (1 = 300 dots; 300 / 16.5 = 18.18):

```
        #define ASCII_SI       15     /* select primary font */
        #define ASCII_SO       14     /* select secondary font */

        long hPos;
        long vPos;
        int row;
        MAXIFONTOUT maxiOutput;

        ...

        /* do the encode here */

        ...

        /*  Print symbol 3 inches in, 2 inches down (at 300 DPI)  */
        hPos = 900;
        vPos = 600;

        /*  Invoke the secondary (Maxicode font)  */
        fputc(ASCII_SO, fpPrinter);

        for (row = 0;
             row < sizeof(maxiOutput.output)/sizeof(maxiOutput.output[0]);
             row++)
        {
            /*  Set the print position  */
            fprintf(fpPrinter,
                    "\033*p%dX\033*p%dY",
                    hPos,
                    vPos);

            /*  Print a line of characters  */
            fwrite(maxiOutput.output[row],
                   1,
                   sizeof(maxiOutput.output[row]),
                   fpPrinter);

            /*  Increment the vertical line position  */
            vPos += 18;

    }

    /*  Return to the primary font  */
    fputc(ASCII_SI, fpPrinter);
```

The important thing to note from this example is that each line of the encoder's output is positioned before it is sent to the printer. All of the lines are in the same column on the printer (hPos in the example; 900 pixels, or 3 inches). However, the row at which each line is printed (vPos) is 18 pixels down from the previous line. Thus the first line is positioned at pixel 600, the second at 618, the third at 636, and so on.

## 5.4 Using AFP PAGEDEFs

When printing a Maxicode symbol using an AFP printer, the Maxicode font must be installed on the printer or made available as a printing resource to PSF. The exact procedure for accomplishing this is based on your site's configuration. Consult with your system administrator for details on how the font was installed.

Using AFP, font selection and line-to-line spacing is usually controlled from within the Page Definition (PAGEDEF) file. The following is a fragment from a sample PAGEDEF file for a 240 DPI printer. The sample assumes that:

- The Maxicode font has been assigned a logical font name of "MAXIFONT".
- The Maxicode data is being printed on ANSI channel 2.
- The symbol is to be positioned 3.75" (900 pels) from the left edge of the page and 5" (1200 pels) from the top of the page.
- The Maxicode output has been directed to channel 2 by the application program.

```
SETUNITS 1 PELS 1 PELS LINESP 14 PELS;

PRINTLINE REPEAT 17
       POSITION 900 1200
       FONT MAXIFONT
       CHANNEL 2;
```

This same page definition will work for a 300 DPI printer, except the "SETUNITS" line would need to be changed to 18 pels:

```
SETUNITS 1 PELS 1 PELS LINESP 18 PELS;
```

## 5.5 Xerox Printing

When printing a Maxicode symbol using a Xerox printer, the Maxicode font must be installed on the printer. Both the 9700 series and 5-word fonts have been provided on Xerox formatted media. Be sure to install the correct fonts for your printer model. However, installing these fonts on your specific Xerox printer is beyond the scope of this document. The following tables lists the fonts provided (NOTE: there may be additional, unrelated fonts on the disk provided):

| Font Series | Fonts |
|---|---|
| 5-word font family | X5PMAX, X5LMAX |
| 9700 font family | X9PMAX, X9LMAX |

When defining the Maxicode font in your JSL it is not strictly necessary to provided line spacing information (as the font has been setup up with the correct spacing). However, if you are accustomed to using line spacing for your font declarations, the Maxicode fonts use a line spacing of 16.66 lines per inch.

To print the symbol using Metacode commands, each line must be positioned as it is printed. Each of the 17 lines of output must be printed by sending the vertical position, horizontal position, font selection, the 30 characters comprising the line, and the terminate command. A line spacing of 16.66 lines per inch (18 pixels) is used for the Maxicode font (Xerox printer positioning is based on 300 DPI, regardless of the printer's resolution).

## 5.6 Using AS/400 DDS

As mentioned at the beginning of this section, the Maxicode symbol must be printed at 16.5 lines per inch. Unfortunately, this is not a valid value for a DDS. Therefore, in order to print a Maxicode symbol using a DDS, the 17 lines of encoder output must be individual placed on the

page using the POSITION function. NOTE: this requires that the printer file be created with DEVTYPE(*AFPDS).

Consider the following DDS fragment:

```
A         R MAXIOUT                   ENDPAGE
A                                     CDEFNT(X0MAXI3A)
A           MAXI01        30A         POSITION(1.700  1.35)
A           MAXI02        30A         POSITION(1.761  1.35)
A           MAXI03        30A         POSITION(1.821  1.35)
A           MAXI04        30A         POSITION(1.882  1.35)
A           MAXI05        30A         POSITION(1.942  1.35)
A           MAXI06        30A         POSITION(2.003  1.35)
A           MAXI07        30A         POSITION(2.064  1.35)
A           MAXI08        30A         POSITION(2.124  1.35)
A           MAXI09        30A         POSITION(2.185  1.35)
A           MAXI10        30A         POSITION(2.245  1.35)
A           MAXI11        30A         POSITION(2.306  1.35)
A           MAXI12        30A         POSITION(2.367  1.35)
A           MAXI13        30A         POSITION(2.427  1.35)
A           MAXI14        30A         POSITION(2.488  1.35)
A           MAXI15        30A         POSITION(2.549  1.35)
A           MAXI16        30A         POSITION(2.609  1.35)
A           MAXI17        30A         POSITION(2.670  1.35)
```

This DDS prints the Maxicode symbol with its upper left corner 1.70 inches down and 1.35 inches in on a page. Each of the successive 16 lines of output is manually positioned 0.0606 inches lower that the previous one (16.5 lines per inch, rounded to the nearest thousandth of an inch).

The CDEFNT function selects the font – in this case the X0MAXI3A font. The X0MAXI3A font is for use with 300, 600, and 1200 DPI page printers. If your printer is a 240 DPI printer, use the X0MAXI2A font.

# 6 Appendix

## 6.1 Structured Message Append

When using Structured Message Append to generate a set of two or more symbols containing a single long message, the following rules apply:

1. For Structured Carrier Message symbols, all the symbols in the set must have the same Primary Message data (postal code, country code and service class). The overall secondary message must be formatted as shown in **Primary and Secondary Message Formats** on page 13, but then may be divided across the symbols at any point that is convenient. The decoder will automatically combine the secondary message from the symbol set during the decode process.
2. For Generic Data Messages (mode 4 or 5) the message may be broken at any convenient point.

Thus, suppose the following message were being encoded:

[)>$^{R}_{S}$01$^{G}_{S}$96339010000$^{G}_{S}$840$^{G}_{S}$001$^{G}_{S}$1Z12345678$^{G}_{S}$UPSN$^{G}_{S}$06X610$^{G}_{S}$159$^{G}_{S}$1234567$^{G}_{S}$
1/2$^{G}_{S}$3.1$^{G}_{S}$Y$^{G}_{S}$2201 SECOND ST, SUITE 600$^{G}_{S}$FT MYERS$^{G}_{S}$FL$^{R}_{S}$$^{E}_{O_T}$

According to the rules in **Primary and Secondary Message Formats** on page 13, the postal code, country code and class of service are removed when constructing the secondary message. Thus, the resulting secondary message is:

[)>$^{R}_{S}$01$^{G}_{S}$961Z12345678$^{G}_{S}$UPSN$^{G}_{S}$06X610$^{G}_{S}$159$^{G}_{S}$1234567$^{G}_{S}$1/2$^{G}_{S}$3.1$^{G}_{S}$Y$^{G}_{S}$
2201^SECOND^ST,^SUITE^600$^{G}_{S}$FT^MYERS$^{G}_{S}$FL$^{R}_{S}$$^{E}_{O_T}$

(For readability, caret characters ('^') have been inserted where spaces would normally go.) At 93 characters, this secondary message is too long to fit in a single symbol. To encode this into a pair of symbols, the appropriate SCM function could be called twice, once to generate a first symbol and once to generate a second symbol.

In this case, the parameters passed to the SCM function would be:

| Structure Element | Symbol 1 | Symbol 2 |
|---|---|---|
| symbolNum | 1 | 2 |
| symbolCount | 2 | 2 |
| postalCodeLen | 9 | 9 |
| postalCode | "339010000" | "339010000" |
| countryCode | 840 | 840 |
| serviceClass | 001 | 001 |
| secondaryMsgLen | 48 | 45 |
| secondaryMsg | [)>$^{R}_{S}$01$^{G}_{S}$961Z12345678$^{G}_{S}$UPSN$^{G}_{S}$06X610$^{G}_{S}$159$^{G}_{S}$1234567$^{G}_{S}$1/2$^{G}_{S}$ | 3.1$^{G}_{S}$Y$^{G}_{S}$2201^SECOND^ST,^SUITE^600$^{G}_{S}$FT^MYERS$^{G}_{S}$FL$^{R}_{S}$$^{E}_{O_T}$ |

Note that, as stated before, the segmentation of the secondary message is arbitrary. Note also that, although the primary message information is repeated between the first and second symbol,

the Structured Carrier Message header string (`[)>`$^{R}_{S}$`01`$^{G}_{S}$`96`) is not repeated in Symbol 2's secondary message.

## 6.2 Font Initialization Values

The Maxicode Encoder is shipped with all available printer fonts. Supported platforms include IBM AFP, Xerox, and Hewlett-Packard PCL compatible printers.

The tables on the following pages list various combinations of the supported platforms, printers and interconnect methods. The tables provide two key pieces of information: the name of the font(s), and the code points to use with the encoder. The code points are the characters that must be output by the encoder for it to work successfully with the font, assuming that no other character conversion is applied by the application programmer. Refer to the `MaxInitC` (for C programming) or `MAXINIT` (for COBOL programming) section for more information on setting the code points. Failure to set the code points properly in the program can cause unpredictable output and can even cause printer reboots.

The information has been broken into two tables; one for UNIX and PC Platforms (ASCII platforms) and a second for IBM S/370, S/390, and AS/400 systems (EBCDIC platforms).

The code point values are provided in hexadecimal and are listed in the order required by the initialization function (No Hex, Lo Hex, Hi Hex, Two Hex, Pad, Bullseye).

## UNIX and PC Platforms

| Printer Family | Attached Via | DPI | Portrait Font(s) | Landscape Font(s) | Code Points (in HEX) |
|---|---|---|---|---|---|
| IBM AFP | No PSF | 240 | X0MAXI2A | X0MAXI2A | 30, 31, 32, 33, 34, 35 |
| IBM AFP | No PSF | 300 | X0MAXI3A | X0MAXI3A | 30, 31, 32, 33, 34, 35 |
| IBM AFP | Using PSF | 240 | X0MAXI2A | X0MAXI2A | F0, F1, F2, F3, F4, F5 |
| BM AFP | Using PSF | 300 | X0MAXI3A | X0MAXI3A | F0, F1, F2, F3, F4, F5 |
| Xerox | Direct | 300 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 30, 31, 32, 33, 34, 35 |
| HP PCL | Direct | 300 | MAXHP3P | MAXHP3L | 30, 31, 32, 33, 34, 35 |

## IBM S/370, S/390, AS/400

| Printer Family | Attached Via | DPI | Portrait Font(s) | Landscape Font(s) | Code Points (in HEX) |
|---|---|---|---|---|---|
| IBM AFP | Channel | 240 | X0MAXI2A | X0MAXI2A | F0, F1, F2, F3, F4, F5 |
| IBM AFP | Channel | 300 | X0MAXI3A | X0MAXI3A | F0, F1, F2, F3, F4, F5 |
| IBM AFP | Network via US Robotics modem | 240 | X0MAXI2A | X0MAXI2A | F0, F1, F2, F3, F4, F5 |
| IBM AFP | Network via US Robotics modem | 300 | X0MAXI3A | X0MAXI3A | F0, F1, F2, F3, F4, F5 |
| IBM AFP | LAN | 240 | X0MAXI2A | X0MAXI2A | 30, 31, 32, 33, 34, 35 |
| IBM AFP | LAN | 300 | X0MAXI3A | X0MAXI3A | 30, 31, 32, 33, 34, 35 |
| IBM AFP | Agile box, using DSC character set | 240 | X0MAXI2B | X0MAXI2B | F0, F1, F2, F3, F5, F7 |
| IBM AFP | Agile box, using DSC character set | 300 | X0MAXI2B | X0MAXI2B | F0, F1, F2, F3, F5, F7 |
| IBM AFP | Agile box, using SCS character set | 240 | X0MAXI2A | X0MAXI2A | 20, 21, 22, 23, 24, 25 |
| IBM AFP | Agile box, using SCS character set | 300 | X0MAXI3A | X0MAXI3A | 20, 21, 22, 23, 24, 25 |
| IBM AFP | Agile box, using DSC APL character set | 240 | X0MAXI2B | X0MAXI2B | 32, 30, 0C, 14, 33, 22 |
| IBM AFP | Agile box, using DSC APL character set | 300 | X0MAXI2B | X0MAXI2B | 32, 30, 0C, 14, 33, 22 |
| IBM AFP | Agile box, using SCS APL character set | 240 | X0MAXI2A | X0MAXI2A | 30, 31, 32, 33, 34, 35 |
| IBM AFP | Agile box, using SCS APL character set | 300 | X0MAXI3A | X0MAXI3A | 30, 31, 32, 33, 34, 35 |

| Printer Family | Attached Via | DPI | Portrait Font(s) | Landscape Font(s) | Code Points (in HEX) |
|---|---|---|---|---|---|
| Xerox | Channel | 300 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | F0, F1, F2, F3, F4, F5 |
| Xerox | Channel | 600 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | F0, F1, F2, F3, F4, F5 |
| Xerox | Network via US Robotics modem | 300 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 30, 31, 32, 33, 34, 35 |
| Xerox | Network via US Robotics modem | 600 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 30, 31, 32, 33, 34, 35 |
| Xerox | Agile box, using DSC character set | 300 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 20, 21, 22, 23, 24, 25 |
| Xerox | Agile box, using DSC character set | 600 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 20, 21, 22, 23, 24, 25 |
| Xerox | Agile box, using SCS character set | 300 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | F0, F1, F2, F3, F4, F5 |
| Xerox | Agile box, using SCS character set | 600 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | F0, F1, F2, F3, F4, F5 |
| Xerox | Agile box, using DSC APL character set | 300 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 45, 46, 47, 48, 49, 4A |
| Xerox | Agile box, using DSC APL character set | 600 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 45, 46, 47, 48, 49, 4A |
| Xerox | Agile box, using SCS APL character set | 300 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 8F, 90, 9A, 9B, 9D |
| Xerox | Agile box, using SCS APL character set | 600 | X5PMAX, X9PMAX | X5LMAX, X9LMAX | 8F, 90, 9A, 9B, 9D |