Defining Word Boundaries and Line Breaks

This section describes how the **NFindWord** procedure uses state machines and associated tables to determine word boundaries and line breaks.

The **NFindWord** procedure examines a block of text to determine the boundaries of the word that includes a specified character in the block. Usually, **NFindWord** uses different state tables to define words for word selection and words for word wrap (line breaking).

> **Note:** **NFindWord** considers offsets within a block of text to be positions between characters-for example, an offset of 1 in Roman text is between the first and second characters (or on the trailing edge of the first and on the leading edge of the second).

NFindWord uses a state machine to determine word boundaries. The state machine must start at a point at or before the beginning of the word that includes the specified character. This can be accomplished in two ways. First, if the specified character is sufficiently close to the beginning of the text buffer, the state machine simply starts from the beginning of the buffer. This is determined by the *doBackupMin* parameter in the tables: if the *offset* parameter is less than *doBackupMin*, the state machine starts from the beginning. Otherwise, **NFindWord** uses a second state table, BackwardTable, for backward processing. With BackwardTable, **NFindWord** starts at the specified character, moving backward as necessary until it encounters a word break.

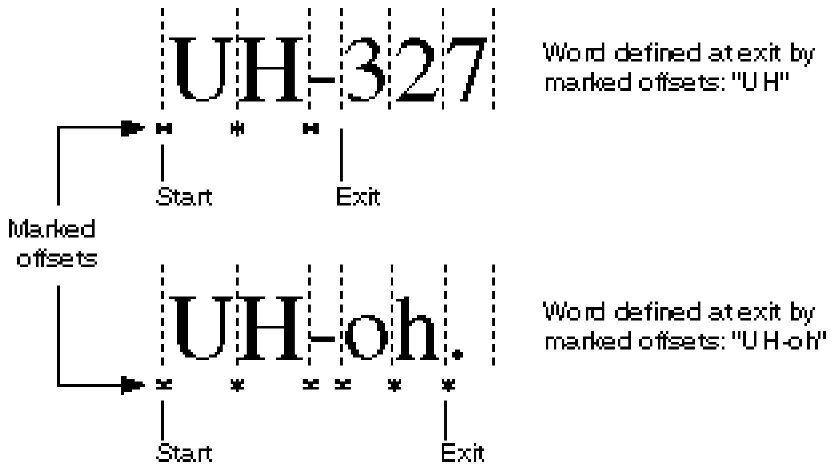
Once determined, this starting point is saved as an initial word break location. From this point, the **NFindWord** state machine moves forward using ForwardTable until it encounters another word break. If that word break is still before the specified character, its location is saved as the starting point, and the state machine is restarted from that location. This process repeats until the state machine finds a word break that is after the specified character. At that point, **NFindWord** returns the last saved word break location and the current word break location as the offset pair defining the word boundaries.

The state machine operates in a similar manner whether moving backward or forward; any differences in behavior are determined by the tables. The machine begins in the start state (state 1). It then cycles one character at a time until it finds a word break and exits. Each cycle proceeds as follows: the current character is mapped to a class number. The character class and the current state are used as indices into a two-dimensional array of byte-length action codes. Each action code specifies the following:

- whether to mark the current offset
- the next state, which may be the exit state (state 0)

When the state machine exits, it has encountered a word break. The location of the word break is the last character offset that was marked. In general, the state machine marks a character offset when it determines that the word that began at its starting point extends at least to the marked offset.

The Figure below gives two examples of the forward operation of the state machine for word selection. In each case, the state at a given offset and the class of the character following the offset determine (1) whether to mark that offset, (2) whether to exit at that point, and, if not exiting, (3) what the state at the next offset will be. When the state machine exits, the first and last marked offsets are returned as the word boundaries.


UH-327
Word defined at exit by marked offsets: "U H"
Start
Exit
Marked offsets
UH-oh.
Word defined at exit by marked offsets: "U H-oh"
Start
Exit

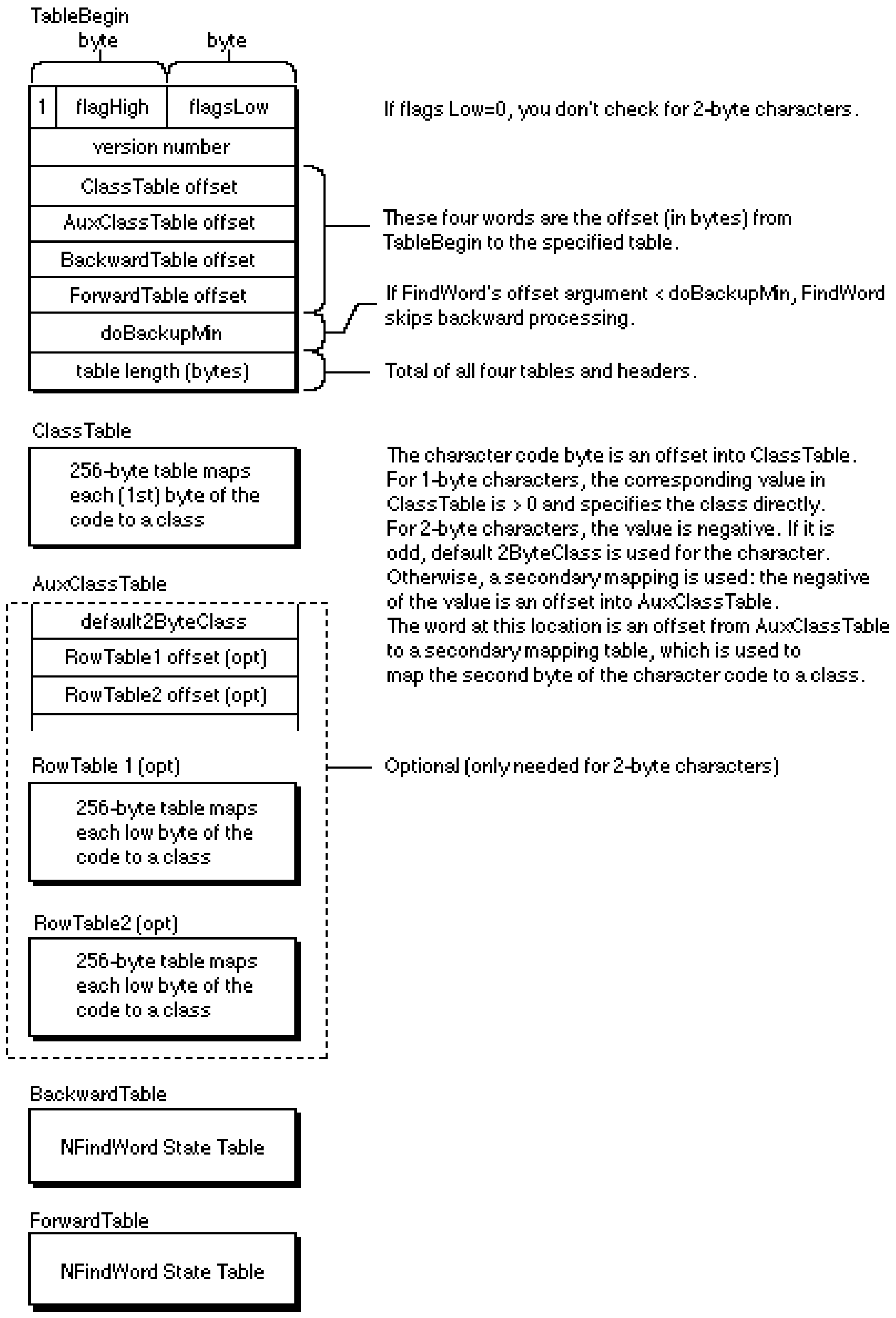
Forward operation of the state machine for word selection

Mapping characters to classes is simple for single-byte characters, but the process gets a little more involved for double-byte character codes. The byte value at the current character offset is used as an offset into the ClassTable array, an array of 256 signed bytes. If the value in ClassTable is positive, it signifies that the byte at the current character offset is a single-byte character, and the value in ClassTable is the class number for the character. If the value in ClassTable is negative, it signifies that the byte at the current character offset is the first byte of a double-byte character code, and AuxClassTable must be used to determine the character class.

The AuxClassTable begins with a variable-length word array. The first word contains a default class number for double-byte character codes. The following words are offsets to RowTables, which have the same format as ClassTable, but are used by **NFindWord** for mapping the second byte of a double-byte character code to a class number. If the value in ClassTable was -1 (or any odd, negative number), the double-byte character code is assigned the default class from the first word of AuxClassTable. For other double-byte characters, the value in ClassTable is an even negative number; **NFindWord** negates this value to provide an offset from the beginning of AuxClassTable to the appropriate RowTable offset. The RowTable table specified by this offset in this way is used to map the second byte of the character to a class number.

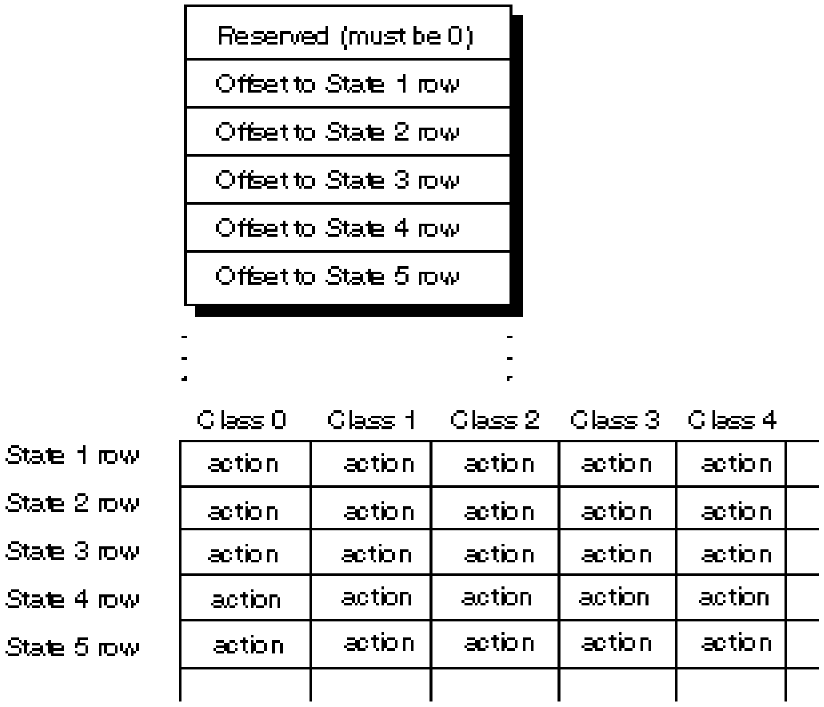
> **Note:** There is a maximum of 128 classes and 128 states (including the start and exit states).

The Figure below provides a description of the new break table. Note that the high bit of the first word is set to indicate that this table is in the new format; otherwise, **FindWord** assumes that the tables are in the old format. The new break table begins with an 8-word header, followed by the class and state tables.


TableBegin
byte
byte
1
flagHigh
flagsLow
version number
ClassTable offset
AuxClassTable offset
BackwardTable offset
ForwardTable offset
doBackupMin
table length (bytes)
If flags Low=0, you don't check for 2-byte characters.
These four words are the offset (in bytes) from TableBegin to the specified table.
If FindWord's offset argument < doBackupMin, FindWord skips backward processing.
Total of all four tables and headers.
ClassTable
256-byte table maps each (1st) byte of the code to a class
The character code byte is an offset into ClassTable. For 1-byte characters, the corresponding value in ClassTable is > 0 and specifies the class directly. For 2-byte characters, the value is negative. If it is odd, default 2ByteClass is used for the character. Otherwise, a secondary mapping is used: the negative of the value is an offset into AuxClassTable. The word at this location is an offset from AuxClassTable to a secondary mapping table, which is used to map the second byte of the character code to a class.
AuxClassTable
default2ByteClass
RowTable1 offset (opt)
RowTable2 offset (opt)
RowTable 1 (opt)
256-byte table maps each low byte of the code to a class
Optional (only needed for 2-byte characters)
RowTable2 (opt)
256-byte table maps each low byte of the code to a class
BackwardTable
NFindWord State Table
ForwardTable
NFindWord State Table

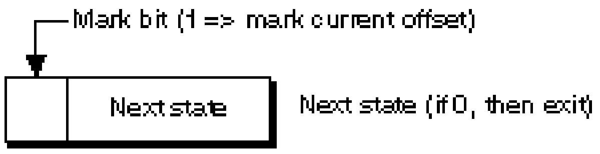
NFindWord header and class tables

The Figure below shows the **NFindWord** state table. It begins with a list of words containing byte offsets from the beginning of the state table to the rows of the state table; this is followed by a C-by-S byte array, where C is the number of classes and S is the number of states. The bytes in this array are stored with the column index varying most rapidly; that is, the bytes for the State 1 row precede the bytes for the State 2 row. Each byte in this array is an action code whose format is defined in the figure below.



NFindWord state table

The Figure below shows the format of an action code.


Mark bit (1 => mark current offset)
Next state
Next state (if 0, then exit)

Format of **NFindWord** action code