```
   // The character after %g is the degree sign
   Printf "The temperature is %g%s\r", temperature, kDegreeSignCharacter
End
```

This web page is a useful resource for comparing MacRoman and Windows-1252 text encodings: https://kb.iu.edu/d/aesh

The MacRoman column is labeled "Mac OS" and the Windows-1252 column is labeled "Windows". The Latin1 column is a subset of Windows-1252.

You can see from the table that there are many characters that are available in MacRoman but not in Windows-1252 (e.g., Greek small letter pi) and vice-versa (e.g., multiplication sign). This is one of the reasons for switching to Unicode.

The left column of the table shows UTF-16 character codes. The table does not show UTF-8. You can convert a UTF-16 character code to UTF-8 text in Igor7 or later like this:

```
String piCharacter = "\u03C0"
```

For details on "\u", see **Escape Sequences in Strings** on page IV-14.

Other Text Encodings Issues

This section discusses miscellaneous issues relating to text encodings.

Characters Versus Bytes

In olden times, each character was represented in memory by a single byte. For example, in ASCII, A was represented by 0x41 (0x means hexadecimal) and B was represented by 0x42. Life was simple.

A single byte can represent 256 unique values so it was possible to represent 256 unique characters using a single byte. This was enough for western languages.

There came a time when people wanted to represent Asian text in computers. There are far more than 256 Asian characters, so multi-byte character sets (MBCS), such as Shift JIS, were invented. In MBCS, the ASCII characters are represented by one byte but most Asian characters are represented by two or more bytes. At this point, the equivalence of character and byte was no more and it became important to distinguish the two.

For example, in Igor6, the documentation for strlen said that it returns the number of characters in a string. This was true for western text but not for Asian text. The documentation was changed in Igor7 to say that strlen returns the number of bytes in a string.

In Igor7, Igor was changed to store text internally as UTF-8, a byte-oriented Unicode encoding form. In UTF-8, ASCII characters are represented by the same single-byte codes as in ASCII, thus providing complete backward compatibility for ASCII text. However, all non-ASCII characters, including accented characters, Greek letters, math symbols and special symbols, are represented by multiple bytes. A given character can consist of 1, 2, 3 or 4 bytes. The payback for this complexity is that UTF-8 can represent nearly every character of nearly every language.

In Igor6, this command prints 1 but in Igor7 or later, it prints 2:

```
Print strlen("µ")
```

This distinction is immaterial when you treat a string as a single entity, which is most of the time. For example, this code works correctly regardless of how characters are represented:

```
String units = "µs"
Printf "The units are %s\r", units
```

It is when you tinker with the contents of a string that it makes a difference. This works correctly in Igor6 and incorrectly in Igor7 or later: