```
   String list = WaveList("*",";","")
   Print list

   Variable numItems = ItemsInList(list), i
   Variable offset = 0
   for(i=0; i<numItems; i+=1)
      String item = StringFromList(0, list, ";", offset) // Get ith item
      Print item
      offset += strlen(item) + 1
   endfor
End
```

See **ListToTextWave** for another fast way to iterate through the items in a large string list.

Using Keyword-Value Packed Strings

A collection of disparate values is often stored in a list of keyword-value pairs. For example, the TraceInfo and AxisInfo functions return such a list. The information in these strings follows this form

```
KEYWORD:value;KEYWORD:value; ...
```

The **keyword** is always upper case and followed by a colon. Then comes the **value** and a semicolon. When parsing such a string, you should avoid reliance on the specific ordering of keywords — the order is likely to change in future releases of Igor.

Two functions will extract information from keyword-value strings. NumberByKey is used when the data is numeric, and StringByKey is used when the information is in the form of text.

Using Strings with Extractable Commands

The **AxisInfo** and **TraceInfo** readback functions provide much of their information in a form that resembles the commands that you use to make the settings in the first place. For example, to set an axis to log mode, you would execute something like this:

```
ModifyGraph log(left)=1
```

If we now look at the characters of the string returned by AxisInfo we see:

```
AXTYPE:left;CWAVE:jack; ... RECREATION:grid(x)=0;log(x)=1 ...
```

To use this information, we need to extract the value following "log(x)=" and then use it in a ModifyGraph command with the extracted value with the desired graph as the target. Here is a user function that sets the log mode of the bottom axis to the same value as the left axis:

```
Function CopyLogMode()
   String info,text
   Variable pos
   info=AxisInfo("","left")
   pos= StrSearch(info,"RECREATION:",0) // skip to start of "RECREATION:"
   pos += strlen("RECREATION:")         // skip "RECREATION:", too
                                        // get text after "log(x)="
   text= StringByKey("log(x)",info[pos,inf],"=",";")

   String cmd = "ModifyGraph log(bottom)=" + text
   Execute cmd
End
```

Character-by-Character Operations

Igor functions like **strlen** and **strsearch**, as well as Igor string indexing (see **String Indexing** on page IV-16), are byte-oriented. That is, strlen returns the number of bytes in a string. Strsearch takes a byte position as a parameter. Expressions like `str[<index>]` return a single byte.

In Igor6, for western languages, each character is represented by a single byte, so the distinction between bytes and characters is insignificant, and byte-by-byte operations are sufficient.

In Igor7 and later, which use the UTF-8 text encoding to represent text in strings, all ASCII characters are represented by one byte but all non-ASCII characters are represented by multiple bytes (see **Text Encodings** on page III-459 for details). Consequently, byte-by-byte operations are insufficient for stepping through characters. Character-by-character operations are needed.

There is no straightforward way to step through text containing non-ASCII characters. In the rare cases where this is necessary, you can use the user-defined functions shown in this section.

The following functions show how to step through UTF-8 text character-by-character rather than byte-by-byte.

```
// NumBytesInUTF8Character(str, byteOffset)
// Returns the number of bytes in the UTF-8 character that starts byteOffset
// bytes from the start of str.
// NOTE: If byteOffset is invalid this routine returns 0.
//       Also, if str is not valid UTF-8 text, this routine return 1.
Function NumBytesInUTF8Character(str, byteOffset)
   String str
   Variable byteOffset

   Variable numBytesInString = strlen(str)
   if (byteOffset<0 || byteOffset>=numBytesInString)
      return 0          // Programmer error
   endif

   Variable firstByte = char2num(str[byteOffset]) & 0x00FF

   if (firstByte < 0x80)
      return 1
   endif

   if (firstByte>=0xC2 && firstByte<=0xDF)
      return 2
   endif

   if (firstByte>=0xE0 && firstByte<=0xEF)
      return 3
   endif

   if (firstByte>=0xF0 && firstByte<=0xF4)
      return 4
   endif

   // If we are here, str is not valid UTF-8.
   // Treat the first byte as a 1-byte character.
   // This prevents infinite loops.
   return 1
End

Function UTF8CharactersInString(str)
   String str

   Variable numCharacters = 0

   Variable length = strlen(str)
   Variable byteOffset = 0
   do
      if (byteOffset >= length)
         break
      endif
      Variable numBytesInCharacter = NumBytesInUTF8Character(str, byteOffset)
```