Fast string file searching

```
function StringInFile(strFind, strFileName: string): boolean;
const
  BUFSIZE = 8192;
var
  fstm: TFileStream;
  numread: Longint;
  buffer: array [0..BUFSIZE-1] of char;
  szFind: array [0..255] of char;
  found: boolean;
begin
  StrPCopy(szFind, strFind);
  found := False;
  fstm := TFileStream.Create(strFileName, fmOpenRead);
  repeat
    numread := fstrm.Read(Buffer, BUFSIZE);
    if BMFind(szFind, Buffer, numread) >= 0 then
      found := True
    else if numread = BUFSIZE then // more to scan
      fstm.Position := fstmPosition - (Length(strFind)-1);
  until found or (numread < BUFSIZE);
  fstm.Free;
  Result := found;
end;
```

The reason for backing up *fstm.Position* by nearly the length of *strFind* is in case *strFind* crosses buffer boundaries.

The *BMFind* function used above is a Boyer-Moore search as shown below. This is the fastest string search known.

```
function BMFind(szSubStr, buf: PChar; iBufSize: integer): integer;
  { Returns -1 if substring not found,
  or zero-based index into buffer if substring found }
var
  iSubStrLen: integer;
  skip: array [char] of integer;
  found: boolean;
  iMaxSubStrIdx: integer;
  iSubStrIdx: integer;
  iBufIdx: integer;
  iScanSubStr: integer;
  mismatch: boolean;
  iBufScanStart: integer;
  ch: char;
begin
  { Initialisations }
  found := False;
  Result := -1;
  { Check if trivial scan for empty string }
  iSubStrLen := StrLen(szSubStr);
  if iSubStrLen = 0 then
  begin
    Result := 0;
    Exit
  end;

  iMaxSubStrIdx := iSubStrLen - 1;
  { Initialise the skip table }
  for ch := Low(skip) to High(skip) do skip[ch] := iSubStrLen;
  for iSubStrIdx := 0 to (iMaxSubStrIdx - 1) do
    skip[szSubStr[iSubStrIdx]] := iMaxSubStrIdx - iSubStrIdx;

  { Scan the buffer, starting comparisons at the end of the substring }
  iBufScanStart := iMaxSubStrIdx;
  while (not found) and (iBufScanStart < iBufSize) do
  begin
    iBufIdx := iBufScanStart;
    iScanSubStr := iMaxSubStrIdx;
```

```
    repeat
      mismatch := (szSubStr[iScanSubStr] <> buf[iBufIdx]);
      if not mismatch then
        if iScanSubStr > 0 then
        begin // more characters to scan
          Dec(iBufIdx); Dec(iScanSubStr)
        end
      else
        found := True;
    until mismatch or found;
    if found then
      Result := iBufIdx
    else
      iBufScanStart := iBufScanStart + skip[buf[iBufScanStart]];
  end;
end;
```

I have included a *wholeword_only* flag in the *BMFind* below. This confirms or rejects the *found* result, and will cause the loop to keep searching if match is rejected.

```
function BMFind(szSubStr, buf: PChar; iBufSize: integer;
  wholeword_only: boolean): integer;
  { Returns -1 if substring not found,
  or zero-based index into buffer if substring found }
var
  iSubStrLen: integer;
  skip: array [char] of integer;
  found: boolean;
  iMaxSubStrIdx: integer;
  iSubStrIdx: integer;
  iBufIdx: integer;
  iScanSubStr: integer;
  mismatch: boolean;
  iBufScanStart: integer;
  ch: char;
begin
  found := False;
  Result := -1;
  iSubStrLen := StrLen(szSubStr);
  if iSubStrLen = 0 then
  begin
    Result := 0;
    Exit
  end;

  iMaxSubStrIdx := iSubStrLen - 1;
  { Initialise the skip table }
  for ch := Low(skip) to High(skip) do skip[ch] := iSubStrLen;
    for iSubStrIdx := 0 to (iMaxSubStrIdx - 1) do
      skip[szSubStr[iSubStrIdx]] := iMaxSubStrIdx - iSubStrIdx;

  { Scan the buffer, starting comparisons at the end of the substring }
  iBufScanStart := iMaxSubStrIdx;
  while (not found) and (iBufScanStart < iBufSize) do
  begin
    iBufIdx := iBufScanStart;
    iScanSubStr := iMaxSubStrIdx;
    repeat
      mismatch := (szSubStr[iScanSubStr] <> buf[iBufIdx]);
      if not mismatch then
        if iScanSubStr > 0 then
        begin // more characters to scan
          Dec(iBufIdx); Dec(iScanSubStr)
        end
        else
          found := True;
    until mismatch or found;
    if found and wholeword_only then
    begin
      if (iBufIdx > 0) then
        found := not IsCharAlpha(buf[iBufIdx - 1]);
      if found then
        if iBufScanStart < (iBufSize - 1) then
          found := not IsCharAlpha(buf[iBufScanStart + 1]);
```

```
    end;
    if found then
      Result := iBufIdx
    else
      iBufScanStart := iBufScanStart + skip[buf[iBufScanStart]];
  end;
end;
```

Obviously you'll be tempted to increase *BUFSIZE* on the assumption that it will improve performance. My experience is that it does not, and that 8K is pretty optimum.

Author:	Unknown
Added:	2007-06-11
Last updated:	2007-06-11