In June 2004, I developed several new plug-ins for use with the music notation software package Sibelius, widely used by professionals in Europe and increasingly in the U.S. Below is a five-page sample (of 13 pages) of the ManuScript programming language code I wrote for a plug-in that automatically adds crescendo and decrescendo marks to a piece of music based on its dynamic markings. This is one of three plug-ins I developed summer 2004.

```
{
   Initialize "() {
AddToPluginsMenu(_PlugInMenuName,'Run'); }"
   Run "() {
/* Add Hairpins version 1.0
June 2004 Mark Feezell
Usual total disclaimers of any liability whatsoever apply*/

if (Sibelius.ScoreCount = 0)
{
   Sibelius.MessageBox(_NoScoreError);
   return False;
}

score = Sibelius.ActiveScore;
selection = score.Selection;

if (selection.IsPassage = False)
{
   Sibelius.MessageBox(_NoPassageError);
   return False;
}

//get and process user settings
if (Sibelius.ShowDialog(settings,Self) = False) {
   return False;
}

None = 0;  //allows you to have 'zero' offsets and still use constants (Eighth, etc.)

if (MaxBarlinesCrossed='No Limit') {
   MaxLengthBars=99999;
} else {
   MaxLengthBars=MaxBarlinesCrossed+1;  //makes calculations simpler below
}

//default to no filtering
VoiceFilter1=-1;
VoiceFilter2=-1;
switch (VoiceFilterString) {  //from dropdown in settings dialogue
   case ('1 or ALL only (recommended)') {
      VoiceFilter1=1;
      VoiceFilter2=0;
   }
   case ('2 or ALL') {
      VoiceFilter1=2;
      VoiceFilter2=0;
   }
   case ('3 or ALL') {
      VoiceFilter1=3;
      VoiceFilter2=0;
   }
   case ('4 or ALL') {
      VoiceFilter1=4;
      VoiceFilter2=0;
   }
   case ('1 only') {
      VoiceFilter1=1;
      VoiceFilter2=1;
   }
   case ('2 only') {
```

```
      VoiceFilter1=2;
      VoiceFilter2=2;
   }
   case ('3 only') {
      VoiceFilter1=3;
      VoiceFilter2=3;
   }
   case ('4 only') {
      VoiceFilter1=4;
      VoiceFilter2=4;
   }
   case ('1 & 2') {
      VoiceFilter1=1;
      VoiceFilter2=2;
   }
   case ('1 & 3') {
      VoiceFilter1=1;
      VoiceFilter2=3;
   }
   case ('1 & 4') {
      VoiceFilter1=1;
      VoiceFilter2=4;
   }
   case ('2 & 3') {
      VoiceFilter1=2;
      VoiceFilter2=3;
   }
   case ('2 & 4') {
      VoiceFilter1=2;
      VoiceFilter2=4;
   }
   case ('3 & 4') {
      VoiceFilter1=3;
      VoiceFilter2=4;
   }
   case ('No voice filtering') {
      VoiceFilter1=-1;
      VoiceFilter2=-1;
   }
   case ('ALL only') {
      VoiceFilter1=0;
      VoiceFilter2=0;
   }
}

score.Redraw = False;

//create progress dialog
Sibelius.CreateProgressDialog('Add Hairpins Plug-in',1,(selection.BottomStaff-
selection.TopStaff+1)*(selection.LastBarNumber-selection.FirstBarNumber+1)+1);
ProgressIndex=1;

for each Staff staff in selection {
   //note prevbar is location of last dynamic, not necessarily previous bar
   prevbar=staff[selection.FirstBarNumber];
   prevposition=0; //pos in prevbar of last dynamic marking
   prevDy=0;
   prevDx=0;
   prevdynamic=''; //'mp', 'ff', etc.
   elapsed=0; //duration elapsed since last dynamic marking

   for barnum = selection.FirstBarNumber to selection.LastBarNumber+1 {
      //update progress dialog
      ProgressIndex=ProgressIndex+1;
      if (staff.IsSystemStaff) {
         StaffName='[System staff]';
      } else {
         StaffName=staff.InstrumentName;
      };
```

```
     if (Sibelius.UpdateProgressDialog(ProgressIndex,'Processing '&StaffName&', measure
'&barnum&'...')=0) {
        Sibelius.DestroyProgressDialog();
        return false;
     };

     //if we've moved to second or later bar of selection,
     // AND we have found a dynamic in an earlier bar
     if ((prevdynamic != '') and (barnum != selection.FirstBarNumber)) {
        //add length of remainder of bar we just examined if last dynamic was found there
        if (prevbar.BarNumber = (barnum-1)) {
           elapsed = elapsed + (staff[barnum-1].Length - prevposition);
        } else {
           //else add the whole elapsed bar
           elapsed = elapsed + staff[barnum-1].Length;
        }
     }
     bar = staff[barnum];

     for each object in bar {
        //if we find a BarRest, reset prev dynamic if no crossing allowed
        if ((object.Type='BarRest') and (CrossBarRests!=true)) {
           //AND if the barrest is in the filtered voice(s)!
           if ((object.VoiceNumber=VoiceFilter1) or (object.VoiceNumber=VoiceFilter2) or
(VoiceFilter1=-1)) {
              prevbar=staff[barnum];
              prevposition=0; //pos in prevbar of last dynamic marking
              prevDy=0;
              prevDx=0;
              prevdynamic=''; //'mp', 'ff', etc.
              elapsed=0; //duration elapsed since last dynamic marking
           }
        }

        //if found text, see if it is a dynamic
        if (object.Type='Text' and ((object.VoiceNumber=VoiceFilter1) or
(object.VoiceNumber=VoiceFilter2) or (VoiceFilter1=-1))) {
           text=object;
           if (text.StyleAsText = 'Expression') {
              nextdynamic=ExtractDynamic(text.Text,0);

              //skip over expression text not containing dynamics

              //also skip over if sfz, fz, sf AND excludesfz='true'
              if ((nextdynamic != '') and (nextdynamic !='0') and ((Excludesfz='false') or
((Instring(nextdynamic,'s')='-1') and (Instring(nextdynamic,'z')='-1')))) {
                 if ((prevdynamic != '') and (prevdynamic !='0')) { //not first dyn on staff
                    if (Instring(Lowercase(text.Text),'sub')!=-1) {
                       //if 'subito' or 'sub' in expression, no hairpin preceding
                       elapsed=0;
                    } else {
                       //else draw hairpin

                       //calculate length of cresc/dim line
                       if (prevbar.BarNumber = barnum) {
                          //same bar, simple difference of position
                          linelen=text.Position-prevposition-@LeftDurationOffset-
@RightDurationOffset;
                       } else {
                          //different bars, make sure not longer than MaxLengthBars setting
                          if ((barnum-prevbar.BarNumber) >= MaxLengthBars) {
                             //longer than allowed, set to start after prevdynamic
                             //calculate new elapsed Length
                             elapsed=0;
                             for x = (barnum-MaxLengthBars)+1 to barnum {
                                elapsed=elapsed+staff[x].Length;
                             }
                             linelen=text.Position+elapsed-@LeftDurationOffset-
@RightDurationOffset;
                             prevposition=0;
```

```
                           prevbar=staff[(barnum-MaxLengthBars+1)];
                           elapsed=0;
                        } else {
                           //smaller than max allowed len, so just add elapsed value and
reset
                           linelen=text.Position+elapsed-@LeftDurationOffset-
@RightDurationOffset;
                           elapsed=0;
                        }
                     }

                     //draw cresc/dim
                     //note:AddLine doesn't respond to prevDx, prevDy fields!  So for now,
leave out.
                     if (DynamicIndex(prevdynamic) > DynamicIndex(nextdynamic)) {

   prevbar.AddLine(prevposition+@LeftDurationOffset,linelen,DiminuendoLineStyle);
                     } else {
                        if (DynamicIndex(prevdynamic) < DynamicIndex(nextdynamic)) {

   prevbar.AddLine(prevposition+@LeftDurationOffset,linelen,CrescendoLineStyle);
                        }
                     }
                  }
               }

               //now that we have processed it, mark 'next' dynamic as previous
               prevbar=staff[barnum];
               prevposition=text.Position;
               prevDy=text.Dy;
               prevDx=text.Dx;

               //this is how we handle 'fp', 'mfpp':
               //1. find the location of the dynamic in the expr text
               //   and look immediately after for another dynamic
               Lowertext=Lowercase(text.Text);

   tempstr=ExtractDynamic(Lowertext,(Instring(Lowertext,nextdynamic)+Length(nextdynamic)));

               //2. if find another dynamic, set it as 'prevdynamic'
               //   otherwise, just use nextdynamic
               if (tempstr!='0') {
                  //BUT only set if continuous ('fp' vs. 'f-p' for 1st/2nd time through)
                  if ((Instring(Lowertext,tempstr)-
Instring(Lowertext,nextdynamic))<=Length(nextdynamic)) {
                     prevdynamic=tempstr;
                  } else {
                     prevdynamic=nextdynamic;
                  }
               } else {
                  prevdynamic=nextdynamic;
               }

               //if 'never after sempre' and contains 'sempre'...
               if ((AllowAfterSempre='false') and
(Instring(Lowercase(text.Text),'sempre')!=-1)) {
                  //no hairpins after 'sempre', don't consider dynamic value
                  prevdynamic='';
               }
            }
         }
      }
   }
}

Sibelius.DestroyProgressDialog();
score.Redraw = True;
}"
   _NoScoreError "You must have a score open to run this plug-in."
```

```
_NoPassageError "Please select a passage and try again."
_PlugInMenuName "Add Hairpins"
settings "Dialog"
{
   Controls
   {
      Button
      {
         Title "OK"
         X "163"
         Y "127"
         Width "50"
         Height "15"
         DefaultButton "1"
         Value
         Method
         SetFocus "0"
         EndDialog "1"
      }
      Button
      {
         Title "Cancel"
         X "105"
         Y "127"
         Width "50"
         Height "15"
         DefaultButton "0"
         Value
         Method
         SetFocus "0"
         EndDialog "0"
      }
      Button
      {
         Title "Help"
         X "7"
         Y "127"
         Width "50"
         Height "15"
         DefaultButton "0"
         Value
         Method "Help"
         SetFocus "0"
      }
      ComboBox
      {
         Title "Eighth"
         X "93"
         Y "6"
         Width "120"
         Height "13"
         ListVar "_RhythmicValuesList"
         Value "LeftDurationOffset"
         Method
         SetFocus "1"
      }
      Text
      {
         Title "Left end offset:"
         X "5"
         Y "9"
         Width "54"
         Height "12"
         RightAlign "0"
         Value
         Method
         SetFocus "0"
      }
      Text
      {
```

```
   Title "Right end offset:"
   X "5"
   Y "25"
   Width "58"
   Height "13"
   RightAlign "0"
   Value
   Method
   SetFocus "0"
}
ComboBox
{
   Title "DottedSixteenth"
   X "93"
   Y "25"
   Width "120"
   Height "13"
   ListVar "_RhythmicValuesList"
   Value "RightDurationOffset"
   Method
   SetFocus "0"
}
ComboBox
{
   Title "2"
   X "93"
   Y "43"
   Width "120"
   Height "13"
   ListVar "_ListOfNumbers"
   Value "MaxBarlinesCrossed"
   Method
   SetFocus "0"
}
Text
{
   Title "# barlines allowed to cross:"
   X "4"
   Y "44"
   Width "86"
   Height "13"

   RightAlign "0"
   Value
   Method
   SetFocus "0"
}
CheckBox
{
   Title "Cross bar rests"
   X "93"
   Y "80"
   Width "67"
   Height "12"
   Value "CrossBarRests"
   Method
   SetFocus "0"
}
CheckBox
{
   Title "Exclude sf, fz, sfz, etc."
   X "93"
   Y "92"
   Width "91"
   Height "19"
   Value "Excludesfz"
   Method
   SetFocus "0"
}
CheckBox
```

```
         {
            Title "Allow after sempre"
            X "93"
            Y "111"
            Width "87"
            Height "12"
            Value "AllowAfterSempre"
            Method
            SetFocus "0"
         }
         ComboBox
         {
            Title "1 or ALL only (recommended)"
            X "93"
            Y "62"
            Width "120"
            Height "13"
            ListVar "_VoiceOptionList"
            Value "VoiceFilterString"
            Method
            SetFocus "0"
         }
         Text
         {
            Title "Consider dynamics in voice:"
            X "4"
            Y "64"
            Width "88"
            Height "16"
            RightAlign "0"
            Value
            Method
            SetFocus "0"
         }
      }
      Title "Add Hairpins Plug-in: Hairpin Settings"
      X "228"
      Y "141"
      Width "226"
      Height "169"
   }
   Lowercase "(str) {
if (Length(str)=0) {
   return '';
}

lowstr='';

for x = 0 to Length(str) {

   CurChar=CharAt(str,x);

   switch (CurChar) {

      //if upper case char, convert to lower
      case ('A') { lowstr=lowstr & 'a'; }
      case ('B') { lowstr=lowstr & 'b'; }
      case ('C') { lowstr=lowstr & 'c'; }
      case ('D') { lowstr=lowstr & 'd'; }
      case ('E') { lowstr=lowstr & 'e'; }
      case ('F') { lowstr=lowstr & 'f'; }
      case ('G') { lowstr=lowstr & 'g'; }
      case ('H') { lowstr=lowstr & 'h'; }
      case ('I') { lowstr=lowstr & 'i'; }
      case ('J') { lowstr=lowstr & 'j'; }
      case ('K') { lowstr=lowstr & 'k'; }
      case ('L') { lowstr=lowstr & 'l'; }
      case ('M') { lowstr=lowstr & 'm'; }
      case ('N') { lowstr=lowstr & 'n'; }
      case ('O') { lowstr=lowstr & 'o'; }
```

```
      case ('P') { lowstr=lowstr & 'p'; }
      case ('Q') { lowstr=lowstr & 'q'; }
      case ('R') { lowstr=lowstr & 'r'; }
      case ('S') { lowstr=lowstr & 's'; }
      case ('T') { lowstr=lowstr & 't'; }
      case ('U') { lowstr=lowstr & 'u'; }
      case ('V') { lowstr=lowstr & 'v'; }
      case ('W') { lowstr=lowstr & 'w'; }
      case ('X') { lowstr=lowstr & 'x'; }
      case ('Y') { lowstr=lowstr & 'y'; }
      case ('Z') { lowstr=lowstr & 'z'; }

      //otherwise, don't change the character
      default { lowstr=lowstr & CurChar; }
   }
}

return lowstr;}"
   Instring "(StrSearched, StrSought) {
//returns 1st position (0-based) of StrSought within StrSearched
//returns -1 if StrSearched does not contain StrSought
//CASE SENSITIVE!

//if either string is 0-length, or StrSought longer than StrSearched, return -1
if (Length(StrSearched)=0 or Length(StrSought)=0 or Length(StrSought)>Length(StrSearched)) {
   return -1;
}

MatchPosition=-1;

//outer while loop (x) moves through StrSearched char by char
//inner while loop moves through StrSought char by char at each x position

x=0;
while ( x < (Length(StrSearched)-Length(StrSought)+1) ) {
   Offset=0;  //offset to step through chars of StrSought at each char
           // position (x) within StrSearched
   KeepLooking=1;
   while ( KeepLooking=1 and Offset<Length(StrSought) ) {
      if ( CharAt(StrSearched,x+Offset)=CharAt(StrSought,Offset) ) {
         if (Offset = (Length(StrSought)-1)) {
            //found it!
            MatchPosition=x;
            //stop both loops since we found it
            KeepLooking=0;
            x=Length(StrSearched)-Length(StrSought)+1;
         }
      } else {
         //one of the characters didn't match, try next position
         KeepLooking=0;
      }
      Offset=Offset+1;
   }
   x=x+1;  //increment outer while to step through StrSearched
}

return MatchPosition;}"
   ExtractDynamic "(ExprStr, Position) {
//returns lowercase text representing the first dynamic marking
//found in an expression text ExprStr; or '0' if none found

//CASE-INSENSITIVE; always returns lower case

//recognized dynamic markings:ppppppp, pppppp, ppppp, pppp, ppp, pp, p,
// mp, mf, f, ff, fff, ffff, fffff, ffffff, fffffff


//prepare/verify string passed in for search
if (Position>=Length(ExprStr)) {
   return '0';
```

```
}
StrToSearch=Substring(ExprStr,Position);
if (Instring(StrToSearch,'P')!=-1) {
   //if it has capital P character, this is a PED marking, not a dynamic; ignore
   return '0';
} else {
   //convert to lowercase
   StrToSearch=Lowercase(StrToSearch);
}


//note:order of 'if' statements is significant; sempre and multiple fff/ppp cause issues

//check for 'niente'
if (Instring(StrToSearch,'n')!=-1) {
   if ((((StrToSearch='n') or (StrToSearch='n.')) and (Position=0)) or
(Instring(StrToSearch,'niente')!=-1)) {
      return 'niente';
   }
}

//if 'sempre', can only find 'mp' if 'mp ' or ' mp'
if (Instring(StrToSearch,'mp')!=-1) {
   if (Instring(StrToSearch,'sempre')!=-1) {
      if ((Instring(StrToSearch, 'mp ')!=-1) or (Instring(StrToSearch,' mp')!=-1)) {
         return 'mp';
      }
   } else {
      return 'mp';
   }
};
if (Instring(StrToSearch,'ppppppp')!=-1) { return 'ppppppp'; };
if (Instring(StrToSearch,'pppppp')!=-1) { return 'pppppp'; };
if (Instring(StrToSearch,'ppppp')!=-1) { return 'ppppp'; };
if (Instring(StrToSearch,'pppp')!=-1) { return 'pppp'; };
if (Instring(StrToSearch,'ppp')!=-1) { return 'ppp'; };
if (Instring(StrToSearch,'pp')!=-1) { return 'pp'; };
if (Instring(StrToSearch,'sfffffffz')!=-1) { return 'sfffffffz'; };
if (Instring(StrToSearch,'sfffffff')!=-1) { return 'sfffffff'; };
if (Instring(StrToSearch,'fffffffz')!=-1) { return 'fffffffz'; };
if (Instring(StrToSearch,'fffffff')!=-1) { return 'fffffff'; };
if (Instring(StrToSearch,'sffffffz')!=-1) { return 'sffffffz'; };
if (Instring(StrToSearch,'sffffff')!=-1) { return 'sffffff'; };
if (Instring(StrToSearch,'ffffffz')!=-1) { return 'ffffffz'; };
if (Instring(StrToSearch,'ffffff')!=-1) { return 'ffffff'; };
if (Instring(StrToSearch,'sffffffz')!=-1) { return 'sffffffz'; };
if (Instring(StrToSearch,'sfffff')!=-1) { return 'sfffff'; };
if (Instring(StrToSearch,'fffffz')!=-1) { return 'fffffz'; };
if (Instring(StrToSearch,'fffff')!=-1) { return 'fffff'; };
if (Instring(StrToSearch,'sffffz')!=-1) { return 'sffffz'; };
if (Instring(StrToSearch,'sffff')!=-1) { return 'sffff'; };
if (Instring(StrToSearch,'ffffz')!=-1) { return 'ffffz'; };
if (Instring(StrToSearch,'ffff')!=-1) { return 'ffff'; };
if (Instring(StrToSearch,'sfffz')!=-1) { return 'sfffz'; };
if (Instring(StrToSearch,'sfff')!=-1) { return 'sfff'; };
if (Instring(StrToSearch,'fffz')!=-1) { return 'fffz'; };
if (Instring(StrToSearch,'fff')!=-1) { return 'fff'; };
if (Instring(StrToSearch,'sffz')!=-1) { return 'sffz'; };
if (Instring(StrToSearch,'sff')!=-1) { return 'sff'; };
if (Instring(StrToSearch,'ffz')!=-1) { return 'ffz'; };
if (Instring(StrToSearch,'ff')!=-1) { return 'ff'; };
if (Instring(StrToSearch,'sfz')!=-1) { return 'sfz'; };
if (Instring(StrToSearch,'sf')!=-1) { return 'sf'; };
if (Instring(StrToSearch,'fz')!=-1) { return 'fz'; };

//this scripting allows it to return 'f' given 'fmf', and return 'mf' given 'mfp' etc.
t1=Instring(StrToSearch,'f');
if (t1!=-1) {
   t2=Instring(StrToSearch,'mf');
   if (t2!=-1) {
```

```
      //found 'mf' too; return earlier one
      if (t1<t2) { return 'f'; } else { return 'mf'; }
   } else {
      return 'f';
   }
}

//if 'sempre' or 'ped' or 'poco' or 'più' or 'piu', can only find 'p' if 'p ' or ' p'
if (Instring(StrToSearch,'p')!=-1) {
   if ((Instring(StrToSearch,'sempre')!=-1) or (Instring(StrToSearch,'ped')!=-1) or
(Instring(StrToSearch,'poco')!=-1) or (Instring(StrToSearch,'più')!=-1) or
(Instring(StrToSearch,'piu')!=-1)) {
      if ((Instring(StrToSearch, 'p ')!=-1) or (Instring(StrToSearch,' p')!=-1)) {
         return 'p';
      }
   } else {
      return 'p';
   }
};

//if none of the 'ifs' were triggered; no valid dynamic markings
return '0';}"
   LeftDurationOffset "Sixteenth"
   _RhythmicValuesList
   {
       "None"
       "Long"
       "Breve"
       "DottedBreve"
       "Whole"
       "DottedWhole"
       "Half"
       "DottedHalf"
       "Quarter"
       "DottedQuarter"
       "Eighth"
       "DottedEighth"
       "Sixteenth"
       "DottedSixteenth"
       "ThirtySecond"
       "DottedThirtySecond"
       "SixtyFourth"
       "DottedSixtyFourth"
       "OneHundredTwentyEighth"

       "DottedOneHundredTwentyEighth"
       "Semibreve"
       "DottedSemibreve"
       "Minim"
       "DottedMinim"
       "Crotchet"
       "DottedCrotchet"
       "Quaver"
       "DottedQuaver"
       "Semiquaver"
       "DottedSemiquaver"
       "Demisemiquaver"
       "DottedDemisemiquaver"
       "Hemidemisemiquaver"
       "DottedHemidemisemiquaver"
       "Semihemidemisemiquaver"
       "DottedSemihemidemisemiquaver"
   }
   RightDurationOffset "Sixteenth"
   _ListOfNumbers
   {
       "0"
       "1"
       "2"
       "3"
```

```
      "4"
      "5"
      "6"
      "7"
      "8"
      "9"
      "10"
      "11"
      "12"
      "13"
      "14"
      "15"
      "16"
      "17"
      "18"
      "19"
      "20"
      "21"
      "22"
      "23"
      "24"
      "25"
      "26"
      "27"
      "28"
      "29"
      "30"
      "No Limit"
   }
   MaxLengthBars "4"
   CrossBarRests "false"
   MaxBarlinesCrossed "3"
   Excludesfz "true"
   DynamicIndex "(DynamicText) {
//Returns a 'loudness index' for a given dynamic text

//Dynamics are text strings, lowercase

//recognized dynamic markings: niente, pppppppp, ppppppp, pppppp, ppppp, pppp, ppp, pp, p,
// mp, mf, f, ff, fff, ffff, fffff, ffffff, fffffff
// as well as sfz, fz, and sf for 1 to 7 f's

//passing other strings will give unpredictable results

switch (DynamicText) {
   case ('niente') { return 1; }
   case ('ppppppp') { return 2; }
   case ('pppppp') { return 3; }
   case ('ppppp') { return 4; }
   case ('pppp') { return 5; }
   case ('ppp') { return 6; }
   case ('pp') { return 7; }
   case ('p') { return 8; }
   case ('mp') { return 9; }
   case ('mf') { return 10; }

   //this returned erroneous results until I reordered them
   case ('fffffff') { return 17; }
   case ('sfffffffz') { return 17; }
   case ('sffffffff') { return 17; }
   case ('ffffffz') { return 17; }

   case ('ffffff') { return 16; }
   case ('sfffffz') { return 16; }
   case ('sffffff') { return 16; }
   case ('fffffz') { return 16; }

   case ('fffff') { return 15; }
   case ('sfffffz') { return 15; }
   case ('sfffff') { return 15; }
```

```
   case ('fffffz') { return 15; }

   case ('ffff') { return 14; }
   case ('sfffffz') { return 14; }
   case ('sffff') { return 14; }
   case ('ffffz') { return 14; }

   case ('fff') { return 13; }
   case ('sfffz') { return 13; }
   case ('sfff') { return 13; }
   case ('fffz') { return 13; }

   case ('ff') { return 12; }
   case ('sffz') { return 12; }
   case ('sff') { return 12; }
   case ('ffz') { return 12; }

   case ('f') { return 11; }
   case ('sfz') { return 11; }
   case ('sf') { return 11; }
   case ('fz') { return 11; }

   default {return 0;}
};}"
   _VoiceOptionList
   {
       "1 or ALL only (recommended)"
       "2 or ALL"
       "3 or ALL"
       "4 or ALL"
       "1 only"
       "2 only"
       "3 only"
       "4 only"
       "1 & 2"
       "1 & 3"
       "1 & 4"
       "2 & 3"
       "2 & 4"
       "3 & 4"
       "No voice filtering"
       "ALL only"
   }
   VoiceFilterString "1 or ALL only (recommended)"
   AllowAfterSempre "true"
   Help "() {
if (Sibelius.ShowDialog(helpwindow,Self) = False) {
   return False;
}}"
   helpwindow "Dialog"
   {
      Controls
      {
         Button
         {
            Title "OK"
            X "457"
            Y "308"
            Width "36"
            Height "14"
            DefaultButton "0"
            Value
            Method
            SetFocus "0"
            EndDialog "1"
         }
         Text
         {
            Title
            X "5"
```

```
            Y "4"
            Width "486"
            Height "301"
            RightAlign "0"
            Value "_HelpText"
            Method
            SetFocus "0"
          }
      }
      Title "Add Hairpins Plug-in: Help"
      X "91"
      Y "56"
      Width "504"
      Height "347"
   }
   _HelpText "Overview of Plug-in:
      -Adds hairpins by determining whether adjacent dynamics within the selected passage are
increasing or decreasing in loudness.
      -Dynamics must be in the expression text style.
      -Hairpins are placed in the default vertical position for hairpins.  Just click and
drag the center of the hairpin to adjust.
      -Adjustments to the ends of hairpins will be necessary, but hairpins should be in
approximately the right rhythmic positions.
      -Any dynamic with subito/sub./sub will not be preceded by a hairpin.
      -Expression text without dynamics, CRESC/DIM text, and existing hairpins are all
ignored.
      -Range of dynamics recognized: ppppppp-fffffff.
      -Automatically interprets combined dynamics (ex: fp approached as f, left as p).
      -Expression text is considered one staff at a time, even for multi-staff instruments.
To move all dynamics in the second staff up, triple click a bar in that staff, select
Edit>Filter>Expression text, and drag one of the dynamics up until it touches the top staff.

Left and right end offsets:
      Distance to keep left and right ends of hairpin away from dynamics on either side, in
addition to Sibelius default distances under HouseStyle>Default Positions; select NONE to
disable.

# barlines allowed to cross:
      Maximum number of barlines that a given hairpin will cross.  If the distance between
two dynamics is greater than this, the hairpin will be shortened appropriately and placed
closer to the second of the two dynamics.

Consider dynamics in voice:
      Will only examine expression text in this voice or voices.  The voice of the expression
text object is used for this purpose.  To check the voice of your dynamics, click the dynamic
and look at the voice indicator at the bottom of the Sibelius keypad.  See VOICES in the
Sibelius manual.
      For passages with 2 voices on one staff, I recommend running the plug-in twice (once
for each voice).  If you select more than one voice, or NO FILTER, each dynamic in the
selected voices is considered in order within the measure, regardless of voice.  This can
produce unpredictable results.
      Sometimes bar rests can be an issue if you are adding hairpins to dynamics in one voice
and there is a bar rest in another voice in the same bar; just turn on CROSS BAR RESTS below.

Cross bar rests:
      If deselected, hairpins will never be drawn across bar rests (empty measures).

Exclude sf, fz, sfz, etc.:
      This determines whether the plug-in would draw a crescendo from mf to sfz, for example,
or ignore the sfz as an articulation marking.  Markings up to sfffffffz, sfffffff, and
ffffffz are covered by this option.  f, sf, fz, and sfz are all considered to be the same
loudness by this plug-in.

Allow after SEMPRE:
      If deselected, no hairpin will be added to lead to the next dynamic after a dynamic
mark containing the word sempre."
   _Author "By Mark Feezell www.drfeezell.com"
}
```