```
{
  Initialize "(){ AddToPluginsMenu('Rhythm Paste','Run'); }"
  Run "() {

//Sets the current selection to a variable
thisscore = Sibelius.ActiveScore;
selection = thisscore.Selection;

//Creates all the necessary arrays for our pitch, articulations, tuplets and ties
sourcePositions = CreateSparseArray();
sourceDurations = CreateSparseArray();
sourceNoteCounts = CreateSparseArray();
sourceArticulations = CreateSparseArray();
sourceTupletLeft = CreateSparseArray();
sourceTupletRight = CreateSparseArray();
sourceTupletPosition = CreateSparseArray();
sourceTupletUnit = CreateSparseArray();
sourceTies = CreateSparseArray();
destinationPositions = CreateSparseArray();
destinationNoteCounts = CreateSparseArray();
destinationPitchs = CreateSparseArray();
isFirstLoop=1;

//Sets the information needed to determine how many times to run the plugin (once per stave)
topStaff = selection.TopStaff;
bottomStaff = selection.BottomStaff;
firstBar = selection.FirstBarNumber;
lastBar = selection.LastBarNumber;

//Checks to see if more than one measure is selected.  This plugin is only designed to copy one measure at a time
if (lastBar-firstBar>0)
  {
  Sibelius.MessageBox('Please copy only one measure.');
  ExitPlugin();
  }


numberOfStaves= bottomStaff-topStaff+1;
staveIncrementer = 0;

//Everything after this is inside this while loop.  It resets for each new stave
while (numberOfStaves>staveIncrementer)

  {
  //selects the next stave
  selection.SelectPassage(firstBar,lastBar,topStaff+staveIncrementer,topStaff+staveIncrementer);

  //Fills the arrays with the necessary note information from the destination measures
  x = 0;

  for each NoteRest n in selection
    {
    if (n.NoteCount>0)
      {
      destinationPositions[x] = n.Position;
      destinationNoteCounts[x] = n.NoteCount;
      x=x+1;
      }
   }


  x=0;

  for each Note o in selection
    {
```

```
    destinationPitchs[x] = o.Pitch;
    x=x+1;
    }

  selection.Paste(0);

  //Checks again to see if multiple measures have been selected in the source bar after it is pasted in
  sourceLastBar = selection.LastBarNumber;
  sourceFirstBar = selection.FirstBarNumber;
  if (sourceLastBar-sourceFirstBar>0)
  {
  Sibelius.MessageBox('Please copy only one measure.');
  ExitPlugin();
  }

  x=0;

  //Copies the source material on the first instance of the loop to avoid recopying the source arrays.
  if (isFirstLoop=1)
  {
    for each NoteRest n in selection

      {
      sourcePositions[x] = n.Position;
      sourceDurations[x] = n.Duration;
      sourceNoteCounts[x] = n.NoteCount;
      sourceArticulations[x] = n.Articulations;
        if (sourceNoteCounts[x]>0)
        {
        topNote = n.Highest;
        sourceTies[x] = topNote.Tied;
        }

      x=x+1;
    }

    x=0;
    for each Tuplet n in selection
      {
        sourceTupletLeft[x] = n.Left;
        sourceTupletRight[x] = n.Right;
        sourceTupletPosition[x] = n.PositionInTuplet;
        sourceTupletUnit[x] = n.Unit;
        x=x+1;
      }
  isFirstLoop=0;
  }

  //Clears the destination measure to allow for populating it with new information
  for each Bar n in selection
      {
         n.ClearNotesAndModifiers();
         barRhythmLength = n.Length;
      }

  x=0;

  //Copies in the tuplets first, if there are any
  numberOfTuplets = sourceTupletLeft.Length;

  for each Bar n in selection
    {
      while (x<numberOfTuplets)
        {
        n.AddTuplet(sourceTupletPosition[x],1,sourceTupletLeft[x],sourceTupletRight[x],sourceTupletUnit[x]);
        x=x+1;
```

```
        }
    }

//Creates a set of incrementer for use in the measure creation
currentPlayhead=0;
sourceIncrementer=0;
destinationIncrementer=0;
pitchIncrementer=0;


//Creates the new notes by moving through the bar one position at a time.  Checks to see if the source position requires a note and applies the destination pitches.
while (currentPlayhead<barRhythmLength)
  {
    if (currentPlayhead=0)
    {
      //Checks to see if the source is a rest.  If it is, it will skip this part, not creating any notes
      if (sourceNoteCounts[sourceIncrementer]>0)
      {
      //Makes notes if the source NoteRest is a note or chord
      while (pitchIncrementer<destinationNoteCounts[destinationIncrementer])
        {
        n.AddNote(sourcePositions[sourceIncrementer],destinationPitchs[pitchIncrementer],sourceDurations[sourceIncrementer],sourceTies[sourceIncrementer]);
        pitchIncrementer=pitchIncrementer+1;
        }
      //Resets the pitch incrementer after creating notes
      pitchIncrementer=pitchIncrementer-destinationNoteCounts[destinationIncrementer];
      }
      //Move the playhead and source incrementer ahead.  Position 0 must contain a NoteRest, but other positions may be empty
      currentPlayhead=currentPlayhead + 1;
      sourceIncrementer=sourceIncrementer + 1;
    }
    else {
          //Checks to see if the destination notes have changed
          if (currentPlayhead = destinationPositions[destinationIncrementer+1])
            {
                //Move to the next set of notes
                pitchIncrementer=pitchIncrementer+destinationNoteCounts[destinationIncrementer];
                //Move the destination incrementer forward
                destinationIncrementer=destinationIncrementer+1;
            }
          if (currentPlayhead = sourcePositions[sourceIncrementer])
            {
            //Checks to see if the source Note Rest is a rest or chord
            if (sourceNoteCounts[sourceIncrementer]>0)
              {
                  //Sets the current pitch incrementer to create the number of notes equivalent to the destination note count
                  currentPitchIncrementer=pitchIncrementer;
                  while(pitchIncrementer<(destinationNoteCounts[destinationIncrementer]+currentPitchIncrementer))
                    {
                    n.AddNote(sourcePositions[sourceIncrementer],destinationPitchs[pitchIncrementer],sourceDurations[sourceIncrementer],sourceTies[sourceIncrementer]);
                    pitchIncrementer=pitchIncrementer + 1;
                    }
                  //Reset the pitch incrementer
                  pitchIncrementer = pitchIncrementer - destinationNoteCounts[destinationIncrementer];

              }
            sourceIncrementer=sourceIncrementer+1;
            }
        currentPlayhead=currentPlayhead + 1;
      }
  }


//Now we add the articulations
sourceIncrementer = 0;

```

```
  for each NoteRest n in selection
    {
    n.Articulations = sourceArticulations[sourceIncrementer];
    sourceIncrementer = sourceIncrementer + 1;
    }

  //resets the destination arrays to prepare for copying on the next stave
  x=destinationNoteCounts.Length;
  while (x>0)
    {
    destinationNoteCounts.Pop();
    destinationPositions.Pop();
    x=x-1;
    }

  staveIncrementer=staveIncrementer+1;
  }
}




}"
}
```