Get/Set Note Property Sparse Array routines in NoteProperties.plg and utils.plg (post 2020.9)

Bob Zawalich November 10, 2020

Update History

Version 01.60.00 published 13 December 2020.

Mostly support for **ChangeDurationNoteRest** plugin
Many bug fixes and reworkings, and several new routines are added

These will not be cloned to **utils.plg**

- General purpose properties routines:
  - // **AddNoteRestSetProperties**
  - // **SetNoteRestAndNotePropertiesFromSparseArray**
- Routines used in **ChangeDurationNoteRest**
- Most of these are routines I have used in many plugins, but some have been fine-tuned and improved
  - // **GetNoteDottedState**
  - // **IsDotted**
  - // **IsValidDuration**
  - // **IsValidNoteDuration**
  - // **LimitDotted**
  - // **GetDurationBetween**
  - // **GetDurationToEndOfScore**
  - // **DurationFitsInScore**

Changes propagated to utils.plg

- Support added to the Note Properties routines to Get and Set the **AccidentalStyle** value

(This replaces a similar document that described a model where properties were held in strings for storage, and converted to arrays when the fields were applied to NoteRests and Notes.)

Many plugins need to write notes to scores, and they will often do this when they need to replace notes or chords. This happens when you want to change the duration of a NoteRest, or just to copy a NoteRest to another score when copy and paste does not work. Early plugins written before **Transpose** was available will do this to change the pitch of a note as well.

Some time ago I came up with a scheme of storing the properties of NoteRests and Notes as strings, where fields are separated by semicolons (These are referred to as **NotePropertyStrings**). These strings are then stored in arrays (or arrays of arrays), and the arrays are passed to routines that actually write the NoteRests and Notes.

As in all cases of creating notes, the plugins need to enumerate all the original Note or NoteRest property fields so they can be restored. If new fields are added (like the recent **LVTies** and **TieInto** fields), all plugins that use **NotePropertyStrings** should be updated. They have currently, however, not been updated, and these fields will be lost when the plugins are used.

Dozens of plugins use this mechanism, and they often use the fields in slightly different ways. In addition to the plugins that use Get/Set Note Property String, there are many more that create notes using other mechanisms,

and these will likely also have to be updated to ensure that the properties are properly transferred. My hope is to be able to convert most of those to use these mechanisms.

Some downloadable Plugins that use **NotePropertyStrings**

- LvFake
- CreateTupletDifferentUnits
- Tuplet Over Barlines
- Add Interval
- Arpeggiate Chords
- Dot Undot Rhythms
- RespellAccidDoubles
- Delete Duplicate Text Lines Symbols
- Percussion Pitch Map
- Ties Into Second Ending
- LV Fake
- Add Interval
- Arpeggiate Chords
- DotUndotRhythms

Downloadable Plugins that call AddNote but not PropertyStrings

- Respell Uncommon Accidentals
- Notation from Live Playback
- Fix Tab 5 String Banjo
- ImportColoredNoteheadStyles
- 12 Tone Matrix
- AddSimpleClickTrack
- AddKeySigToSelection
- Explode Specific Staves
- Invert Chords
- Add Pickup Bar
- Copy Notes As Grace Notes
- Fill Selection With Slash Notes
- Input Duration
- HideShowInvisibleNoteheads
- ExportAudioWithCountIn
- UnfilterSelection
- FixTabCollisions
- HarpGlissAndOctaveNotes
- HarpGlissPitches
- AddPickupbarWithDeletedRests
- AddSlashNoteheadsForParts
- Create Trailing Pseudo-Grace Notes
- FillWithTiedNotes
- LowerPitchChromatically
- Scordatura
- SlashNotesAtMidline
- StringHarmonics*
- TieExtyendedStable
- Unarpeggiate
- ArrangeBrassVoicings
- ArrangeVoicings
- ListScaleTones
- CombineRestsEmptyVoices
- TupletOverBarlines

- CreateTupletDifferentUnits (done)
-
-

Shipping plugins that use **NotePropertyStrings**

- Divide Durations
- Repitch
- Rerhythm

Retrograde, **Double/HalveNoteValues**, and **Combine Tied Notes and Rests**, among other plugins, do not use **NotePropertyString** code but they have similar embedded code.

Centralizing Note Property code in NoteProperties.plg and utils.plg

Rather than updating each plugin every time a change in properties is made, my plan is to create a new library plugin (**NotePropertiesLib.plg**) that contains standard routines to store and apply property strings. I will edit plugins to replace their internal code with calls to **NotePropertiesLib.plg**, and "normalize" their use of this resource. This way, when changes are made to Note or NoteRest properties, **NotePropertiesLib.plg** can be updated, and for the most part, plugins that call these routines will just need to get an updated **NotePropertiesLib.plg** file.

This means that the users will need to install **NotePropertiesLib.plg** in order to run any plugin that calls into it. This will only need to happen once for any given user, though it should be updated when it changes, which I hope will not be very often.

One change I made to the model is that I no longer store properties in strings and convert the string to and from arrays. Instead, the properties are always stored in Sparse Arrays, though there do not use the "sparse" feature. They are treated as conventional arrays that can store objects, binary data, and Booleans. There is no longer a need to convert strings to and from arrays, and there are fewer methods to deal with. Conversions from plugins that used Property Strings will be a bit more complex, but really not all that much.

The first plugins I will convert will likely be those that already use some form of **NotePropertyStrings** code. I plan, though, to adapt other plugins that use more direct processes to have them call the **NotePropertiesLib.plg** routines to take advantage of the future ease of updating.

For shipping plugins, the same new routines will be added to the shipping **utils.plg**, and plugins will be adapted to use those routines, in future releases of Sibelius Ultimate.

I have rewritten the shipping plugins **Divide Durations**, **Repitch**, **Rerhythm, and Retrograde** to use these routines, (I originally wrote all of them) and I think that **Double Note Values**, and any of the other **Transformations** plugins that create notes should be adapted to at least transfer all the current fields. My feeling is that they should be adapted to use the new **utils.plg** routines so that future update will be easier.

The plugin **Combine Tied Notes And Rests** is called automatically by some of the **Rerhythm** plugins, and I have updated it as well, though it required considerably more work to do so that to convert some of the others.

This is not my decision to make, but I am happy to help support this process.

In downloadable plugins, I rewrote **CopyDoubleNoteValues**, which effectively updated **CopyHalveNoteValues** as well, since that is just a stub that calls into **CopyDoubleNoteValues.**

Where feasible, my intention is to update plugins to support quartertones and updated properties at the same time.

Methods to support NoteProperties Sparse Arrays

NotePropertiesLib.plg includes a number of routines related to **NoteProperties** that are designed to be called by other plugins. The same routines will be added to **utils.plg**, and an effort will be made to keep these routines in sync. We can't use **utils.plg** for all plugins because some plugins need to be run in earlier versions of Sibelius where **utils.plg** will not include these routines.

Some Variable Naming conventions

Sparse Array Descriptions

Some plugins, especially **Combine Tied Notes And Rests**, use multiple nested spares arrays. I find it easier to deal with code if I know what level I am dealing with, since each NoteRest has a sparse array of properties, as does each Note.

The obscurity of the name itself because less of a problem as you dig into complex code trying to figure out what you are working with.

Let's call the basic property array a NoteRestNotePropertyArray (or an **arsNRNP**). It is a sparse array of sparse arrays.

- arsNRNP[0] = sparse array of NoteRest Properties from GetNoteRestPropertySparseArray
- arsNRNP[1..n] = sparse array of Note Properties from GetNotePropertySparseArray
- The routine **GetNoteRestAndNotePropertySparseArray** produces an **arsNRNP**
- arsNRNP[0] will usually be called **arsNoteRestProperties**
- arsNRNP[1..n] will usually be called **arsNoteProperties**

Comparing Sparse Arrays

In some existing plugins, the properties were put into strings separated by semicolons. This was useful both for Storage and as a way to compare 2 notes. Sparse arrays can be compared, and since the contents are either strings, and a small number of binary fields (color and articulations), we should be able to compare the arsNoteProperties of 2 notes directly. (Early tests indicate that differences in articulations are not picked up. Color differences, the other binary field, are picked up.) On reflection, this is because Articulations are stored only as NoteRest properties, not Note properties, but color is present in both.

Voice is stored with the Note property arrays, but not Position or Duration. If you want to include Position, Duration or Articulations in a comparison, you will need to get them from the NoteRest. I suspect Position will rarely be something you want to compare. I have no real use for Note comparison yet, but I may come up with a scheme to let you pass in a range of fields to compare, or just deal directly with the objects and look at specific fields.

But it does appear that comparing the 2 sparse arrays of Note Properties is equivalent to comparing property strings.

List of routine names

NotePropertiesLib.plg includes a "comment only" method aa_INDEX_OF_ROUTINES that gives an alphabetical list of its methods in one place. Here is the list as of this writing

```
// *AddNote
// *AddNoteFromPropertySparseArray
// *AddNoteSetProperties
// *AddPrivatePropertiesNote
// *AddPrivatePropertiesNoteRest
```

// *FormatPlayOnPass
// *GetNotePropertiesVersionNumber
// *GetNotePropertySparseArray
// *GetNoteRestPropertySparseArray
// *GetNoteRestAndNotePropertySparseArray
//* IndexArsNoteProperty
// *IndexArsNoteRestProperty
// *SetNotePropertiesFromSparseArray
// *SetNoteRestPlayOnPassFromString
// *SetNoteRestPropertiesFromSparseArray
// *StringCharactersToArray
// *TraceNotePropertiesFromSparseArray
// *TraceNoteRestPropertiesFromSparseArray

// CopyArray
// CopySparseArray
// BuildSibVersionText = make Sibelius.ProgramVersion displayable with periods between components
// IsNotePropertiesVersionValid
// IsPluginAvailable
// IsValidObject
// MyMessageBox = Sibelius.MessageBox + pluginMenuName
// MyYesNoMessageBox
// NoteRestWithNotesAtBarPosVoice
// NoteRestWithNotesAtTupletPosVoice
// TrueFalseAsNumber

Basic Set and Get Routines

- **GetNotePropertySparseArray**
- **GetNoteRestPropertySparseArray**
- **GetNoteRestAndNotePropertySparseArray**

    - These take a Note or NoteRest and store their properties into a sparse array.
    - The order in which the fields are stored are determined in these routines, and the indexes used to access the fields are set up in **IndexArsNoteProperty** and **IndexArsNoteRestProperty**

- **SetNotePropertiesFromSparseArray**
- **SetNoteRestPropertiesFromSparseArray**
    - These take a (usually newly created) Note or NoteRest and a Sparse Array produced by **GetNotePropertySparseArray** or **GetNoteRestPropertySparseArray** and apply the properties to the object.
    - There are a few parameters that allow some callers to limit which properties are transferred.

Basic Get and Set Routines not used by and not cloned into utils.plg

- // **AddNoteRestSetProperties**
    - creates a NoteRest and all its notes from an arsNRNP
    - returns the created NoteRest, not a Note
- // **SetNoteRestAndNotePropertiesFromSparseArray**
    - Sets both NoteRest and Note properties for all included notes from an arsNRNP

Support Routines not used by and not cloned into utils.plg

- **// GetNoteDottedState**
- **// IsDotted**
- **// IsValidDuration**
    - Calls IsValidNoteDuration
- **// IsValidNoteDuration**

    - Checks that a duration is numeric and neither too small nor too big, with no parent context
  - **// LimitDotted**
  - **// GetDurationBetween**
  - **// GetDurationToEndOfScore**
  - **// DurationFitsInScore**

Support Routines

- **IndexArsNoteProperty**
- **IndexArsNoteRestProperty**
  - These produce a list of indexes from standard Note or NoteRest field names ("Pitch", "Accidental", "VoiceNumber"...) so the actual numeric values describing the order of fields in the property strings are hidden from routines outside of **NoteProperties.plg**.
  - The actual numeric values of the indexes can thus be changed without disturbing calling plugins.

- **AddNoteFromPropertySparseArray** (objParent, position, duration, voice, arsNoteProperties, fSetDiatonicPitch)
- **AddNote**
- **AddNoteSetProperties** (objParent, position, duration, voice, arsNoteRestProperties, arsNoteProperties, fSetDiatonicPitch, fSetNoteRestProps)

  - These are routines that actually add Note objects to a parent Bar, NoteRest, or Tuplet, simplifying the various AddNote routines in ManuScript.
  - **AddNoteFromPropertySparseArray** gets the properties from the property string, and **AddNote** takes a subset of these properties as parameters.
  - **AddNoteFromPropertySparseArray** calls **AddNote** to actually create the Note.
  - **AddNoteSetProperties** takes both the NoteRest and Note properties as arguments. Unlike the other routines, it not only creates the note, it also applies NoteRest and Note properties to the new note at the same time.
- **TraceNotePropertiesFromSparseArray**
- **TraceNoteRestPropertiesFromSparseArray**

Special Purpose Support Routines

- **Play On Pass Routines for NoteRests**
  - **Play On Pass** is not stored as a property, but it can be represented as a string of zeroes and ones. This routine converts the properties to strings that consists of ones when the property is enabled, and 0 when disabled.
  - This routine is called by **Get/SetNotePropertiesFromSparseArray** and **Get/SetNoteRestPropertiesFromSparseArray**
    - ☐ **FormatPlayOnPass**
      - Encodes the 8 play on pass settings into a string of 8 zeroes and ones. Called by **GetNoteRestPropertiesFromSparseArray.**
    - ☐ **SetNoteRestPlayOnPassFromSparseArray**
      - Sets the Play on Pass properties of a NoteRest using a string produced by **FormatPlayOnPass.**

Private data routines

  - These routines can be used by calling plugins to append data to the end of a property string. The new data fields are added following the last values that were created in **GetNotePropertySparseArray**and **GetNoteRestPropertySparseArray**. When the property string is used in **SetNotePropertySparseArray** and **SetNoteRestPropertySparseArray**, it is turned into an array of strings, and the index for the

first field of private data will be determined by calling **IndexArsNoteProperty("NotePropertyPrivateFirst")** or **IndexArsNoteRestProperty("NoteRestPropertyPrivateFirst").** My hope is that these will rarely need to be used, but they should be able to allow a user to add private data to a property string while still maintaining the ability to use the common routines unchanged .

- **AddPrivatePropertiesNoteRest**
  - ☐ This is passed a **NoteRestPropertySparseArray** and an array of strings, each of which will be appended to the Sparse Array. It is the caller's responsibility to provide valid data. , You can call this multiple times, and each call will append to the end of the string, but you must keep track of the indexes for each private data field, starting from **IndexArsNoteRestProperty("NoteRestPropertyPrivateFirst")** .
  - ☐ It is actually easy and sage to append data directly using **ars.Push()**, but this routine was already available for use when the properties were held in strings, so I am leaving it for the abstraction value.
  - ☐ Use at your own risk.

- **AddPrivatePropertiesNote**
  - ☐ This is passed a **NotePropertySparseArray** and an array of strings, from which each string will be appended to the Sparse Array. It is the caller's responsibility to provide valid data.
  - ☐ You can call this multiple times, and each call will append to the end of the string, but you must keep track of the indexes for each private data field, starting from
  - ☐ **IndexArsNoteProperty("NotePropertyPrivateFirst")**
  - ☐ It is actually easy and sage to append data directly using **ars.Push()**, but this routine was already available for use when the properties were held in strings, so I am leaving it for the abstraction value.
  - ☐ Use at your own risk.

Other utility routines

- **StringCharactersToArray**
  - ☐ Converts a string of characters into an array with one array entry for each character in the string. Called by S**etNoteRestArticulationsFromSparseArray** and **SetNoteRestPlayOnPassFromSparseArray** to deconstruct the formatted string.

- **CopyArray**
- **CopySparseArray**
- **BuildSibVersionText = make Sibelius.ProgramVersion displayable with periods between components**
- **IsNotePropertiesVersionValid**
- **IsPluginAvailable**
- **IsValidObject**
- **MyMessageBox = Sibelius.MessageBox + pluginMenuName**
- **MyYesNoMessageBox**
- **NoteRestWithNotesAtBarPosVoice**
- **NoteRestWithNotesAtTupletPosVoice**
  - When adding notes to a bar or tuplet
- **TrueFalseAsNumber**

Property Names used in IndexArsNoteRestProperty and IndexArsNoteProperty

At the current time (December 13, 2020), these are the names used for the property indexes, as generated the first time **IndexArsNoteRestProperty** or **IndexArsNoteProperty** are called.

g_arrNoteRestPropertyNames
{
"NoteCount"
"StaffNum"
"BarNumber"
"Position"
"Duration"
"VoiceNumber"
"Beam"
"SingleTremolos"
"DoubleTremolos"
"CueSize"
"Hidden"
"Articulations"
"FeatheredBeamType"
"HasStemlet"
"StemletType"
"RestPosition"
"ArpeggioType"
"ArpeggioDx"
"ArpeggioTopDy"
"ArpeggioBottomDy"
"ArpeggioHidden"
"FallType"
"FallDx"
"ScoopType"
"ScoopDx"
"HasCustomDrawOrder"
"DrawOrder"
"UsesMagneticLayoutSettingOverridden"
"UsesMagneticLayout"
"StemFlipped"
"PlayOnPass"
"Color"
"ColorAlpha"
"NoteRestPropertyPrivateFirst"
}

g_arrNotePropertyNames
{
"VoiceNumber"
"Name"
"Pitch"
"DiatonicPitch"
"Tied"
"NoteStyle"
"UseOriginalVelocityForPlayback"
"OriginalVelocity"
"UseOriginalDeltaSrForPlayback"
"OriginalDeltaSr"
"UseOriginalDurationForPlayback"
"OriginalDuration"

"AccidentalStyle"
"Bracketed"
"Slide"
"StringNum"
"Color"
"ColorAlpha"
"SlideStyleId"
"TiedInto"
"LvTie"
"NotePropertyPrivateFirst"
}