Slot Text Annotations in RiverWare 5.2

Internal Design

Author: Phil Weinstein

This document describes the internals for a redesign of Text Annotations on Slots. The design satisfies text annotations requirements for the new USBR Green Book document generation capability in RiverWare 5.2.

0.1 Document Status

5-24-2009: Ready for Review; Does not yet include serialization format specification. That will be added later.

0.2 Related Documents

- (USBR RiverWare) Green Book Functionality Draft Design (Neil Wilson, 5-13-2009, 18 pages) /projects/riverware/doc/accounting/GreenBook/GreenBookDraftDesign.fm
- Series Slot Text Annotations in RiverWare, Requirements and High-Level Design (Phil Weinstein, 2-17-2007, 8 pages) .. /projects/riverware/doc/accounting/simlib/SeriesSlotAnnotations.fm.

0.3 Contents

1.0 Requirements Overview

The original implementation of Series Slot Text Annotations in RiverWare 4.9 (Sept. 2007) provides sharing of a particular annotation note across multiple Series Slots, but that annotation can occur only once on any Slot, and must be at the same timestep on all Slots. A new feature soon to be developed -- *USBR Green Book document generation capability in RiverWare 5.2* -- has requirements which are not supported by the original Series Slot Text Annotations implementation. These are:

- It must be possible to apply a given note to *multiple timesteps* on any Series Slot.
- The timestep *date/times* of a note can be *different on different Series Slots.*

So, fundamentally, a timestep date/time can no longer be associated (only) with the note. The date/time must be associated with the *application* (or *association*) of a note to a Series Slot. And multiple annotation associations for a given note must be supportable on any Series Slot.

Also,

- Slot Annotations within an existing RiverWare model must be *automatically converted* to the new Slot Annotation architecture upon model load.
- The *performance and memory usage* for the feature should be reasonably optimal for a large number of annotations which could realistically be defined explicitly by the user. (It is not a requirement that the mechanism be sufficiently efficient for handling myriad automatically-generated notes. Only manually maintained notes are currently supported).

Additional capabilities supported by this new Slot Annotation design:

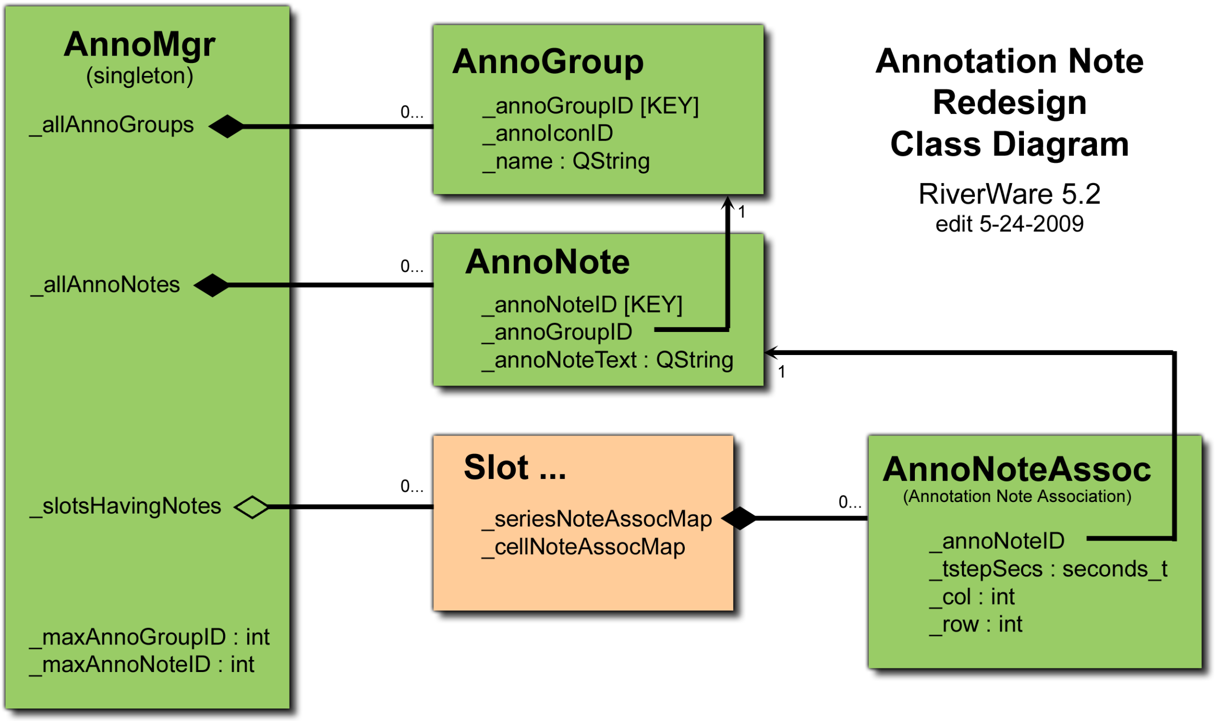
- Support for text annotations on slot values of *any directly-editable slot type* is possible -- not just on Series Slots. The initial development of the new Slot Annotations architecture will not necessarily actually provide that capability. For example the internal design supports adding notes to ListSlot elements, but GUI support for that capability will not be part of this development.
- More than one note can appear on any single Slot value (timestep or table cell).

2.0 User Interface Support

The Green Book Functionality Design document describes the modifications to the user interface to support the new Annotations architecture.

There will be *two versions* of each of the eight (8) differently-colored icons used to designate the existence of an annotation on a particular slot value. A modified set of icons will be added to indicate the existence of *more than one note* on the particular slot value. The color of the multiple-note icon shown on a slot value will arbitrarily correspond to the AnnoGroup of one of the notes on that slot value.

3.0 Data Model



File: AnnoNoteClassDiagram-5p2.psd

3.1 Overview

Virtually none of the prior internal data model design is used by the new architecture. The prior design did not maintain objects for Annotation Notes or for associations of notes to Slots -- both of these were represented only as entries in a map.

The new design specifies use of an AnnoMgr (Annotation Manager) *singleton* and distinct objects for these three entities:

- **AnnoGroup** *(Annotation Group)* consisting of a unique ID, an icon/color reference, and a name.
- **AnnoNote** *(Annotation Note)* consisting of a unique ID, a reference to an AnnoGroup (using an ID), and a note text string.
- **AnnoNoteAssoc** *(Annotation Note Association)* consisting of a reference to an AnnoNote (using an ID) and value-indexing information sufficient to support Series Slots (of all types), Table Slots (of all types), List Slots, and trivially, Scalar Slots. The implementation will also contain a generally-redundant Slot pointer, not illustrated above. Read more below.

These entities implement a relational architecture using integral IDs rather than pointers to refer to related objects. AnnoGroups and AnnoNotes are owned by the AnnoMgr. AnnoNote Associations are owned by individual Slots.

3.2 Annotation Manager (AnnoMgr)

The AnnoMgr is a singleton. It maintains ...

- A map containing (owning) all defined AnnoGroups, indexed by AnnoGroupID,
- A map containing (owning) all defined AnnoNotes, indexed by AnnoNoteID,
- A list of all Slots in the model having at least one AnnoNoteAssoc (Annotation Note Association),
- A mechanism to allocate new unique AnnoGroupIDs and AnnoNoteIDs. There is no need to insure that all used IDs remain contiguous.

Tentative class declaration:

```
typedef int AnnoGroupID;        // Unique Annotation Group IDs
typedef int AnnoNoteID;         // Unique Annotation Note IDs


// Annotation Manager, Singleton.
class AnnoMgr
{
  private:
    AnnoGroupID  _maxAnnoGroupID;   // Seed for new AnnoGroup IDs
    AnnoNoteID   _maxAnnoNoteID;    // Seed for new AnnoNote IDs

    QMap<AnnoGroupID, AnnoGroup>  _allAnnoGroups;  // All instantiated Groups
    QMap<AnnoNoteID,  AnnoNote>   _allAnnoNotes;   // All instantiated Notes

    cwSlist<Slot*>  _slotsHavingNotes;  // Slots having Note Associations

    // methods not enumerated here.
};
```

3.3 Annotation Group (AnnoGroup)

One AnnoGroup object is allocated for each Annotation Group. AnnoGroups are owned by the Annotation Manager. An Annotation Group contains a unique key (AnnoGroupID), an Annotation Icon (Color) ID (enumerated type with eight values, same as the prior design), and an arbitrary Annotation Group Name defined by the user.

Tentative class declaration:

```
class AnnoGroup
{
 public:
    AnnoGroupID    _annoGroupID;    // [KEY]
    AnnoIconID     _annoGroupIconID;
    QString        _annoGroupName;
};
```

Annotation Groups do not contain a collection of their member Annotation Notes. Rather, Annotation Notes contain a reference (an AnnoGroupID) to their parent Annotation Group.

3.4 Annotation Note (AnnoNote)

One AnnoNote object is allocated for each Annotation Note. AnnoNotes are owned by the Annotation Manager. An Annotation Note contains a unique key (AnnoNoteID), a reference to an AnnoGroup (via an AnnoGroupID), and the text character string for the note.

Tentative class declaration:

```
class AnnoNote
{
  public:
    AnnoNoteID   _annoNoteID;       // [KEY]
    AnnoGroupID  _annoGroupID;
    QString      _annoNoteText;
};
```

No timestep date/time or other slot value index information is defined within the AnnoNote.

3.5 Annotation Note Association (AnnoNoteAssoc)

One AnnoNoteAssoc (association) is allocated for each occurrence of an Annotation Note on a slot value (at a timestep date/time, or in a cell indexed by row and column; multiple column Series Slots also use the column index). AnnoNote Associations are owned by Slots having Annotation Notes.

AnnoNoteAssoc objects also contain a pointer to their owning Slot. This is technically redundant in the context of the collection of association objects on the Slot -- but it is useful when the AnnoNoteAssoc object is used in more general contexts, e.g. within a slot display involving multiple Slots. Also, this helps disambiguate indices into AggSeries Slot column series, which have two usable forms:

1. AggSeries Slot pointer; Column index; timestep date/time.
2. Series Slot pointer (to the AggSeries child column); Column Zero; timestep date/time.

The value of the column index value will always be coherent with the slot pointer field.

Tentative class declaration:

```
class AnnoNoteAssoc
{
   AnnoNoteID   _annoNoteID;
   Slot*        _slot;
   seconds_t    _tstepSecs;  // (for Series and TableSeries Slots)
   int          _col;        // (for Multiple Column Slots) [0-based]
   int          _row;        // (for Table and List Slots)  [0-based]
};
```

Timestep date/times are internally encoded as 64-bit integers (seconds_t, or “long long”) representing the number of seconds since the beginning of the supported “epoch” -- currently defined as the beginning of the year 1800 (in file Utils/Date_Time.cpp). This binary value is the independent field in the RiverWare Date_Time class. In serializations (e.g. in a RiverWare model file), these values are represented using a natural date/time string, e.g. “05-25-2009 12:00:00”.

3.6 Added fields to the Slot base class

The Slot class is an abstract base class of all RiverWare Slots, including Series Slots (SeriesSlot, AggSeriesSlot, MultiSlot, SubSlot), Table Slots (TableSlot, TableSeriesSlot, Periodic Slot), List Slots, and Scalar Slots.

Most of the support for Annotations on Slots will be implemented immediately within the Slot base class. An exception to this is the method to add an annotation to a Slot -- the specific Slot subclass will evaluate the validity of the AnnoNoteAssoc object's value indices -- though out-of-range index values will not be regarded as an error.

The Slot class will have pointers to two conditionally-allocated MultiMaps (QMultiMap) -- these are maps which allow multiple values at a single key value. These maps are devised for quick access by the GUI during redrawing operations, and will be of these types:

```
// Notes on Series Slots, indexed by timestep seconds_t (64-bit).
typedef QMultiMap <seconds_t, AnnoNoteAssoc> SeriesNoteQMultiMap;

// Notes on Table, List and Scalar slots, indexed by (row,col), zero-based.
typedef QMultiMap <QPair<int,int>, AnnoNoteAssoc> CellNoteQMultiMap;
```

AnnoNoteAssoc objects are stored in these maps *by value* rather than by reference to a dynamically allocated instance.

The various AnnoNoteAssoc object value-index fields are applicable to the various Slot types as follows:

Slot Type	**_tstepSecs (seconds_t)**	**_col (int, zero-based)**	**_row (int, zero-based)**
SeriesSlot	**USED**	---	---
AggSeriesSlot / MultiSlot	**USED**	**USED**	---
Table Slot	---	**USED**	**USED**
Periodic Slot	---	**USED**	**USED**
Table Series Slot	**USED**	**USED**	---
List Slot	---	---	**USED**
Scalar Slot	---	---	---

Unused values will be 0 if that does not result in ambiguity. Otherwise (-1) will be used (e.g. for the _row value in slots having a time series).

Tentative class declaration (additions to the Slot class):

```
class Slot
{   ... ... ...

   // ************************************************
   // ***  Slot Annotation Note Associations Data  ***
   // ************************************************

 private:
   // Series and Cell Slot Annotation Associations; Possibly Not Allocated.
   SeriesNoteQMultiMap* _seriesNoteAssocMap;  // Time Series Note Associations
   CellNoteQMultiMap*   _cellNoteAssocMap;    // Table Cell Note Associations

   // **********************
   // ***  Note Setters  ***
   // **********************

 public:
   // Insert note methods: Return error if not supported by the slot class.
   virtual okstat addNote (const AnnoNoteAssoc& newNote);

   void clearNotes();
   bool removeNote (AnnoNoteID);
   bool removeNoteGroup (AnnoGroupID);

   // **********************
   // ***  Note Getters  ***
   // **********************

   bool hasNote () const;
   bool hasNoteAt (seconds_t, int col=(-1)) const;
   bool hasNoteInCell (int row, int col=(-1)) const;
   bool hasMultipleNotesAt (seconds_t, int col=(-1)) const;
   bool hasMultipleNotesInCell (seconds_t, int col=(-1)) const;

   int noteCount () const;
   int noteCountAt (seconds_t, int col=(-1)) const;
   int noteCountInCell (int row, int col=(-1)) const;

   QString firstNoteAt (seconds_t, int col=(-1)) const;
   QString firstNoteInCell (int row, int col=(-1)) const;

   int getNotesAt (QList<AnnoNoteAssoc>&, seconds_t, int col=(-1)) const;
   int getNotesInCell (QList<AnnoNoteAssoc>&, int row, int col=(-1)) const;

   // References to the Annotation Note Time or Cell Association QMultiMaps.
   const SeriesNoteQMultiMap* seriesNoteAssociations() const  // MAY BE NULL.
     { return (_seriesNoteAssocMap); }
   const CellNoteQMultiMap* cellNoteAssociations() const      // MAY BE NULL.
     { return (_cellNoteAssocMap); }
};
```

4.0 Handling Old-Style SeriesSlot Annotations during Model Loading

The classes supporting the Old-Style SeriesSlot Annotations will be maintained in RiverWare. These have been renamed in the RiverWare 5.2 development build:

```
AnnoGroup       -->   AnnoGroupOld
AnnoGroupMgr    -->   AnnoGroupMgrOld
```

The Tcl-based commands in the RiverWare model file (and exported object file) supporting the Old-Style SeriesSlot Annotations will not be changed -- they will load annotation data into the AnnoGroupMgrOld and AnnoGroupOld objects. After loading, analogous representations are loaded into the new annotation objects as they are removed from the old annotation structures.

A single Old-Style SeriesSlot Annotation will be potentially result in the creation of AnnoNoteAssoc objects across many Slots. For all practical purposes, this process is not reversible: New-Style annotations cannot be converted to Old-Style annotations (with the preservation of the original AnnoGroups).

As mentioned above, the format of Tcl-based commands supporting the Old-Style annotations will not be (and cannot be) modified. Distinct Tcl commands will be devised for the New-Style annotation data. Only New-Style annotation data will be written out to model files. That is, the Old-Style annotations read in from a previously saved model file will be preserved in newly saved model files only in the form of New-Style annotations.

--- (end) ---