SCT Series Table Qt3 to Qt4 Port -- March 2012

Phil Weinstein, CADSWES, edit 3-12-2012; Status: Ready for review.

(1) Qt3 Implementation Overview

The SCT's Series Slot Table -- on the first tab of the SCT -- provides four series data views:

1. Time Horizontal / without Timestep Aggregation Rows
2. Time Vertical / without Timestep Aggregation Rows
3. Time Horizontal / with Timestep Aggregation Rows
4. Time Vertical / with Timestep Aggregation Rows

SCT Configuration Data is composed primarily of these components:

- An ordered list of Slot / Column reference items supporting references to SeriesSlots (including all SeriesSlot subclasses) and Table Series Slot columns -- plus "slot divider" items. Each slot (or slot divider) item has a user-editable display label which defaults to the full slot name, plus a few other properties to support, for example, aggregation states and user-set column widths. See:
    - class SctConfig
    - struct SctConfigSlotData
    - class SctSlotColRefData (could be replaced by SlotColRef)
- A timestep aggregation configuration, dependent on both properties stored in the main SCT configuration (SctConfig) and SCT timestep size.

SCT Components / Layers used in the Qt3 implementation:

- SctManager -- singleton which keeps track of all open SCTs (SctDialogs).
- SctDialog -- an SCT dialog
- SctView class hierarchy -- four instances. Manages the table layouts for one particular view:
    - SctViewAgg -- aggregated views:
        - SctViewAggTHorz
        - SctViewAggTVert
    - SctViewNoAggTHorz
    - SctViewNoAggTVert
- SctQTable -- Q3Table subclass used for each SctView's row header table and series data table.
- SctModelData -- interface to actual Slot instance references in the loaded model based on the symbolic slot lists defined in the SctConfig.
- Support classes:
    - SctSlotTstepSet -- a cell selection set in terms of (1) slots (indices) and (2) timesteps (indices). Generally, this is constructed from the native Qt table cell selection.
    - SctCopySet and SctCopySource -- a characterization of an SCT cell selection used for copy/paste operations.
    - SctFile and SctFileInterpreter -- SctConfig serialization
    - SctSumFuncSpec -- timestep aggregation function for a particular slot item
- Support widgets and dialogs:
    - EditSctSlotListPanel / TreeView -- deployed on the 2nd SCT tab to edit the SctConfig's series slot list
    - SctClipboardExportDlg -- Export Copy
    - SctConfigDlg -- general editor for an SctConfig instance

- SctTimestepAggCfgDlg -- editor for aggregation parameters within an SctConfig instance
- SctFindSlotDialog -- to search for an SCT's slot item
- SctLabelFnDialog -- minor: edit a slot item's label and summary function
- SctQLineEdit -- QLineEdit subclass used for incell editors
- SctSetSumFuncDlg / SctSumFuncAssignSpec -- support for setting multiple slot items' summary functions

(2) New Components for the Qt4 Table implementation:

- SctTableView -- a Qt4 model/view Table View subclass
- SctTableModel -- a Qt4 model/view QAbstractTableModel subclass. Base class for:
  - SctTableModel_NoAggTHorz
  - SctTableModel_NoAggTVert
  - SctTableModel_AggTHorz
  - SctTableModel_AggTVert
- SctTableDelegate -- a Qt4 model/view QAbstractItemDelegate subclass, for cell painting and editing specialization.

(3) Porting Tasks -- Overview

The display and behavioral features of the SCT's series slot views need to be migrated from the Qt3 "SctView" subclasses and the new Qt4 table view, model and delegate classes. Operations involving changes to the display table configuration (e.g. number of, and content of the rows and columns) -- which are represented in SctConfig -- will need to cause updates within the new model classes instead of within the old table classes.

Note that in the Qt4 model/view architecture, the cell selection is NOT part of the "model" (QAbstractItemModel). Rather, it is represented in a "selection model" instance which is also owned by the table view. As with the Qt3 selection model, care must be taken to implement cell selection analysis very efficiently because it must be performed on every selection change AND because one of the dimensions of the table can be huge (e.g. tens or hundreds of thousands of timesteps) -- but the algorithms devised for an efficient implementation in Qt3 will need to rewritten for the Qt4 selection model architecture. (It's different).

The following sections describe the major porting tasks.

(3.1) Moving GUI table configuration computation mechanisms to the new model classes.

Mechanisms to efficiently compute internal table structure data -- i.e. only when absolutely necessary -- will need to live in the model classes. This includes row/column mappings (see below) and intermediate timestep aggregation computations.

The row/column mappings include these:

- Horizontal Time Views
  - SCT timestep index to column. (aggregated: also to row offset).
  - SCT slot item index to row (aggregated: summary and detail rows)
  - SCT column to timestep (aggregated: first and last timestep for aggregation)
  - SCT row to slot index (aggregated: also to time agg offset).
- Vertical Time Views
  - SCT timestep index to row (for aggregated views: detail or summary).
  - SCT slot item index to column

    - SCT row to timestep or timestep range (first and last timestep; for aggregation)
    - SCT column to slot index
- All Views
    - SCT row and column to slot index and timestep (for aggregated: timestep range)
    - SCT slot index and timestep index to row (summary and detail) and column.

(3.2) Value and Ornamentation Display

Hopefully, the internals of the cell and divider rendering code can substantially remain intact. But it will need to be moved to the new Qt "item delegate" class, with a slightly different interface (e.g. where the drawing "rectangle" comes from). Cell rendering includes these aspects:

- Numeric and Flag-character and Rules-priority integer values.
- Background colors to indicate timestep flags.
- In the aggregated views, the value of Summary cells need to be computed at display time. Also, their backgrounds have complex timestep flag and selection ornamentation.
- Neither Qt3 nor Qt4 directly support any type of row or column dividers. Those need to be implemented as specially sized and painted rows and columns.
- Annotation note ornaments, and actual note text on Annotation columns.
- Read-only and (Object-) Dispatch Disabled cross-hatch ornamentation.

Note that in Qt4, we no longer have a way of providing custom selection ornamentation -- that is performed automatically in Qt after custom cell drawing has been applied. This poses a bit of a problem for our special selection ornamentation in aggregation summary cells. Fortunately, that ornamentation is important primarily in the case of the summary cell not itself being selected (but rather when a subset of its detail cells are selected). This should work out OK.

(3.3) Asynchronous Value Updates

When series slot values or flags change in the model, the SCT needs to update automatically (including a deferred column width expansion, as needed). Actually, updates of display data as a response to value edits in the SCT are *also* asynchronous by default -- i.e. the SCT value display will update in response to a value-change notification from the slot being edited.

Slot change callbacks are maintained by the SctModelData object, and that can remain intact. Those will need to result in generation of "data changed" notifications from the various QAbstractItemModel instances in the new design.

(3.4) Selection Representation

Computing the SCT's internal slot / timestep oriented cell selection from the QTableView's row / column oriented cell selection -- and the reverse operation of setting the QTableView's selection -- need to be reimplemented for Qt4. The internal slot / timestep selection is used to sensitize (enable/disable) and perform various high-level operations on slot data, including the various copy/paste operations. These need to be very efficient, including with large timestep-count models; table selection needs to be re-analyzed with every change to the cell selection. As mentioned above, a QTableView's cell selection is represented in a separate "selection model" object -- and not in the QAbstractItemModel object -- and is quite different from the Qt3 table selection architecture.

It is difficult to simultaneously support certain "selection" features for both the old Qt3 and new Qt4 table implementations, i.e. for the purpose of development testing. At some point during this development step, the "hooks" to the old selection implementation will be removed.

(3.5) Re-deployment of SCT user operations.

Most of the GUI controls to initiate operations exist in the SctDialog class, e.g. menu and toolbar items. These fall into the following general categories:

- External operations which interact with components outside of the SCT, e.g. Run Control and Analysis.
- Internal operations which don't involve the series display tables, e.g. showing Diagnostic Output as a panel in the SCT.
- SCT series table reconfiguration operations: slots and timesteps, time aggregation config, colors, dividers, etc.
- SCT series table view selection: switching between the various SctViews.
- SCT series table navigation features: scrolling to a timestep or slot item.
- Series data edit operations: setting and clearing values and flags, interpolate, incell editing (applied to the full cell selection), etc.

Several of these categories of operations will need to be moved to appropriate component within the Qt4 item view architecture: generally to the new SctTableView or item model subclasses.

(3.6) Precise display behaviors / Usability issues.

Once all the basic functionality is implemented, certain precise table-related behaviors will need to be revisited for Qt4. These include:

- Cell navigation features (e.g. with arrow and tab keys, and cell-edit completion).
- Scroll position and cell selection "soft" persistence. (i.e. not "persistence" in the sense of being saved in an SCT configuration file or model, but "things" staying put through an operation which would be expected to stay put).
- Column widths. The SCT has a very elaborate mechanism for remembering user width setting on individual columns until a higher-level width-setting operation is applied.

(3.7) GUI performance

Bottlenecks in run-time performance, especially in models with huge time series, will need to be examined and optimized. One source of difficulty is that hiding and showing table rows and columns can be very expensive in huge tables, as Qt recomputes the positions of all rows and columns. In some cases, it was found to be more efficient to "RESET" the entire QAbstractItemModel rather than change the hidden state of many individual rows. The performance of operations which are of particular concern in the Qt4 architecture (at least with our current version, Qt 4.6.3) include the following:

- Opening and closing timestep aggregations (in the two aggregated views).
- Modifying the list of slots in the SCT (though the importance of this consideration was substantially reduced with the introduction of the "Edit Series Slot List" tab -- developed, in part, to address this sort of performance problem in the Qt3 tables).
- Actual value updates during a model run. We especially need to confirm that the existence of an SCT doesn't slow down a run when the SCT window has been minimized.

(4.0) Development

I'm recommending that we substantially complete the first several development steps with one of the four SCT views before proceeding with the other three views. This initial development will be coded with the intermediate base Qt4 item model class, SctTableModel. When that is complete, aspects of that model which are particular to the actual view will be moved to the subclass for that view (mainly via virtual methods -- e.g. for the slot / timestep / row / column mappings), and the model subclasses for the other three views will then be implemented.

Task	Est [days]	Compl.	Description
Phase I -- with only the Non-Aggregated Vertical Time View			
I (3.1)	2.0		Moving GUI table configuration computation mechanisms to the new model classes.
I (3.2)	1.5		Value and Ornamentation Display
I (3.3)	0.5		Asynchronous Value Updates
I (3.4)	2.5		Selection Representation
Phase II -- with the remaining three views			
II (3.1)	1.5		Moving GUI table configuration computation mechanisms to the new model classes.
II (3.2)	2.0		Value and Ornamentation Display -- *including support for complex aggregation divider "cells"*
II (3.3)	0.0		Asynchronous Value Updates -- *no additional development expected*
II (3.4)	1.0		Selection Representation
II (3.5)	1.5		Re-deployment of SCT user operations.
II (3.6)	1.5		Precise display behaviors / Usability issues.
II (3.7)	2.0		GUI performance
	16.0		**TOTAL [days]**
