Unit Schemes

August 2011 Vision

Author: Patrick Lynn

This document describes a shift in requirements for the use of unit schemes in RiverWare and analyzes the effects of these changes on the existing unit scheme design and implementation.

1 Functional Requirements

The general idea of the unit scheme work is to switch control over how slot values are displayed from individual slot configurations to a more centralized approach. The approach being considered introduces the concept of a *unit scheme* which describes how to display a set of numeric slot values in terms of four display attributes: units, scale, precision, and format. Users can view and edit unit schemes and easily switch which scheme is being used for value display.

In an attempt to simply the unit scheme concept and user experience, the detailed functional requirements for this work have evolved in the past several monthst; the essentials of the current requirements are:

- The currently active unit scheme is used everywhere a slot value is displayed, at all times. This includes the user interface, diagnostics, and output devices.
- Every unit scheme is complete, that is, it describes how every slot value shall be displayed. This is accomplished by associating with every valid unit type the desired set of display attributes. These *unit type rules* are the most general type of unit scheme rule in terms of applicability as well as the lowest priority. That is, if two rules apply to a given slot value, the rule with the narrower applicability is used.
- When a model saved with a prior RiverWare version is loaded, the *model transition* unit scheme is created which replicates the display of slot values in the prior version.
- Each DMI will continue to use the specifications contained within the DMI to interpret the units and scale of input values and to determine the desired units, scale, precision and format of output values. When the DMI does not contain such a specification, the currently active scheme will be used, though the user will be warned that this is happening and encouraged to specify the units within the DMI.
- The resource database file (riverwareDB) is now obsolete and is only used transitionally.

The major impacts of these requirements on the current (substantially complete) design and implementation are:

- Unit schemes need to support finer granularity rules (e.g., “the primary units for column 2 of slot S on account A on object O should be ...”).
- The automatic creation of the model transition unit scheme needs to be designed and implemented.
- The global slot configuration dialog needs to be redesigned in light of the overlap between the functionality supported there and that of the new unit scheme editor.
- A significant amount of the software and interface can be refactored and simplified. For example, the concept of overriding the unit scheme for a particular slot no longer exists.

Subsequent sections discusses these issues in more detail.

2 Unit scheme rule specificity

The requirement that RiverWare be able to completely describe existing models' slot configurations with a unit scheme has an important consequence for the specificity of unit scheme rules. In the current implementation, there are two types of rule, unit type and slot name. Unit type rules apply to all slot values of a given type, whereas slot name rules apply to slot values with a given unit type and a given name. In the current implementation, users can further restrict the values to those on a particular type of SimObj. That is slot name rules contain the three pieces of information:

• Unit type = u / SimObj type = t / slot name = s

• Unit type = u / SimObj name = * / slot name = s

With the new requirements, at least one type of rule needs to apply to a specific slot/column/primary triplet, because that is the specificity of the existing slot configuration support. That is, the user has been able to uniquely configure primary and alternate units for particular columns on particular slots. So the unit scheme rule support needs to be extended to include at least column and primary/alternate specifications. We will need to decide which parts of this specification are optional (i.e., can be wildcarded); the full set of possibilities is large (close to 3*2*2*2).

- Object: any/by type/by name
- Slot: any/by name
- Column: any/by name
- Primary: either/true or false

My suggestion is that we begin by adding only the most specific rule type, with all parts specified. If experience with models indicates that this leads to a large number of rules, we can add the appropriate rule types.

In addition to interface changes (primarily the Unit Scheme Editor dialog), this will require changes to the underlying classes (e.g., UnitScheme, UnitScheme::Key) including extensions to the XML schema used for serializing schemes.

2.1 Distributed versus centralized value display configuration

One possible criticism of the current functional requirements is that they do not significantly change the user experience: because unit schemes will support the finest possible granularity of specification, there is at one level an equivalence between the old and new approaches to describing how values should be displayed. For example, in theory it is possible that a unit scheme could contain a rule for every single column of every slot, and one could argue that in this case the only way that display configuration changed is where that configuration is saved, on the slot or in the unit scheme. In fact, unit scheme configuration information might even be cached on the slot, so even that would not have changed.

One response to this objection is that we expect most unit schemes to be very compact and composed mostly of unit type rules, so in fact the centralized unit scheme specification will be much simpler than distributing this specification across all of the slot interfaces. After all, what is important is the user’s experience, not the underlying implementation. More to the point, the unit scheme approach shifts the user experience from one centered around individual slots to a more model-oriented view of value display. This approach buys the user increased conceptual simplicity and the ability to easily switch all value displays.

3 Creating a model transition unit scheme

To facilitate a smooth transition for users to the new unit schemes approach to displaying slot values, RiverWare will automatically generate a unit scheme that replicates the previous display behavior. When a model is loaded, RiverWare creates the model transition unit scheme which:

- Is complete, i.e., covers all unit types.
- Describes the existing slot configurations, that is, this scheme displays all slots as configured in the model file.
- Is compact, i.e., has the minimum number of unit scheme rules.

3.1 Solution outline

At the end of model load:

- Create a list of all slots (see note below).
- Use this list to create a data structure with describes the model's display attributes, i.e., those that need to be described by the model transition unit scheme.
- Initialize the model transition unit scheme with default display attributes for each unit type.
- Modify the scheme's rules (which are all unit type rules) to "cover" as many slots as possible. That is, change each rule's set of display attributes to that which is most frequently used for the rule's unit type.
- Add slot name rules of increasing specificity to the scheme until all slots are covered, that is, until every display configuration on every slot is described by the scheme.
- To keep the scheme small, more general slot name rule "formats" should be added first, though we might want to avoid some unusual formats. One possible order:
  - Unit type = u / SimObj type = t / slot name = s
  - Unit type = u / SimObj name = * / slot name = s
  - Unit type = u / full slot name = s / column = c, primary = p
- Point the new scheme out to the user so that they can review it, modify it, give it a name, and decide whether or not to make it the active scheme.

3.2 Notes

I expect that many existing models will contain unintentional slot configurations which would show up as rules in the model transition scheme, typically rules with narrow applicability. If this is in fact common, we might want to draw the user's attention to these rules so that they can delete them.

The model transition scheme could conceptually describe every set of slot values which might potentially have been configured by the user to have the same display attributes, That is columns of values on slots. Slots are managed by the following types of objects: SimObj, Subbasin, Account, Supply, Exchange, and Paybacks. Thus the loop over all slots above, could include slots on all of these objects, though it is perhaps acceptable to skip slots on Exchanges and Paybacks. Note also that for slots associated with a SlotProxy, we need only consider those which are instantiated.

The easiest way to implement this plan requires retaining the local NumDisplayAttribs slot data structures as well as the machinery to set them from old model files, forever. These data structures can be emptied once the model transition unit scheme is created and need not be saved in the model file.

This plan does not address non-slot uses of NumDisplayAttribs, which is fine. The only such example might be the numeric dimension on some TableSlots.

A default unit scheme rule has the following NumDisplayAttribs:

- Scale = 1.0
- Units = standard units for the type, e.g., cms for flow.
- Precision = 2
- Format = 'f' (floating point, e.g., 123.456)

All unit schemes will be complete, so requiring that the model transition unit scheme be complete represents no additional effort.

3.3 Implementation notes

The data structure describing the display information which needs to be captured by the model transition unit scheme could be a map that associates with each <slot, column, primary> triplet that exists in the model, the locally configured NumDisplayAttribs.

To build this "attributes to add to scheme" map:

```
for each slot s
    for each column c on s
        add NumDisplayAttribs for <s, c, true>
    end for
    if s supports alternate units
        for each column c on s
            add NumDisplayAttribs for <s, c, false>
        end for
    end if
end for
```

As the model transition unit scheme is modified, entries in the map that are described by the changes can be removed from the map.

Relevant public Slot methods:

- int colCount() const
- bool supportsAltUnits() const
- unit_type primUnitType(int col) const
- unit_type altUnitType (int col) const
- const NumDisplayAttribs& localNumDisplayAttribs_prim (int col) const
- const NumDisplayAttribs& localNumDisplayAttribs_alt (int col) const
- const NumDisplayAttribs& activeNumDisplayAttribs_prim(int col) const
- const NumDisplayAttribs& activeNumDisplayAttribs_alt (int col) const

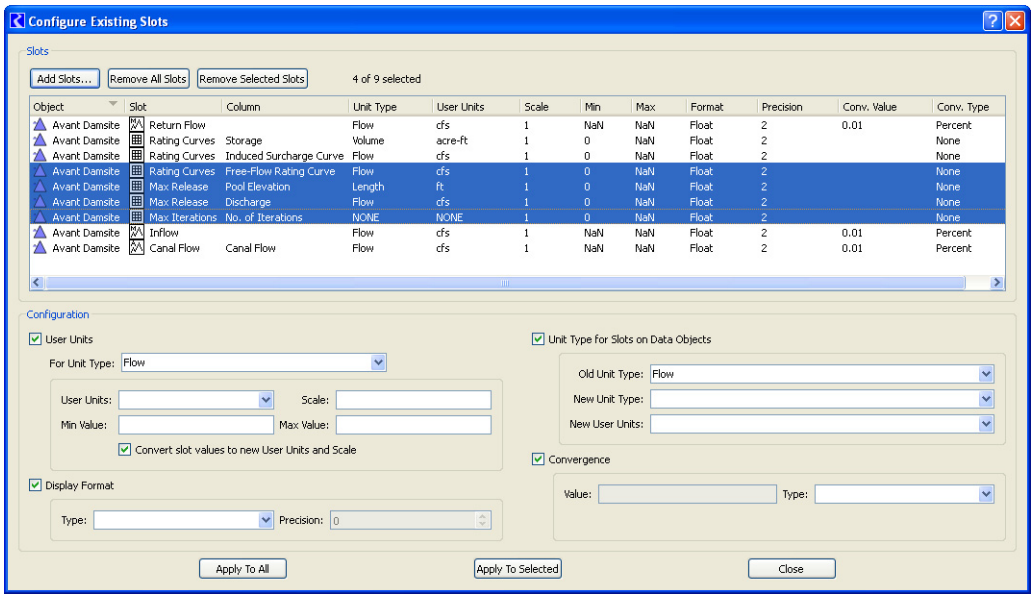
Note that the local NumDisplayAttribs getter methods are no longer needed for anything except the creation of the model transition unit scheme.

The "Slot Name Chooser" dialog creates its of slots by first getting a list of SimObj's from rwSimWorkspace, then asksing each object for its slots and accounts (using some very ugly code which was copied from OpenObject-Dlg::addObjectsSlotItems). There is also the methodology used by the --slotlist command line argument, which dumps information about all slots to a file. The is implemented in Sim/ModelFile.cpp. How best to do this will probably require thought followed by coding effort.

4 Changes to the multiple slot configuration dialog

My recommendation is to remove from this dialog the aspects that apply to the display of slot valules, and provide easy access to the unit scheme editor for changes to slot value display attributes.

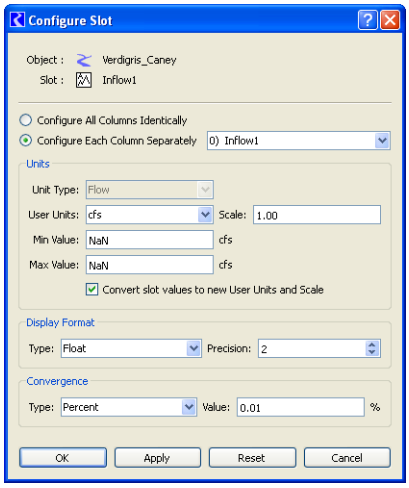
Figure 1. The multiple slot configuration dialog



4.1 Fixing incorrectly input values

When a user inputs slot values, either interactively or via an input DMI, they specify also provide the units of those values, either explicitly as in the case of some DMIs or implicitly as in the case of the open slot dialog.

Figure 2. A single slot configuration dialog



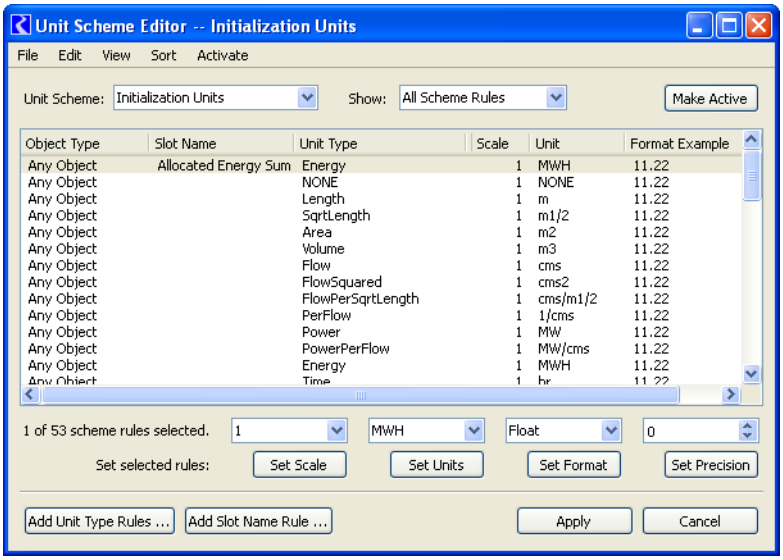
We need to continue to support a way to correct the values on an entire slot column or set of slot columns if the input units are incorrect for the data. This has been done in the slot configuration dialog for individual slots and in the multiple slot configuration dialog, but those are probably not great places for it, since the problem is really data correctness, not data display.

5 Changes to the unit scheme editor

Since all schemes will always have all unit type rules, the ability to add/delete/edit these rules can be removed.

On the other hand, we need to add support for a fully specified rule type. This could be added by selecting a new button, which would bring up a dialog in which the user selected the unit type with a combobox, slot and column with an appropriately configured GUS dialog, and primary or alternate. with a checkbox or other toggle. I'm fairly sure that GUS can be configured to allow selection of slots/columns on almost any type of object: SimObj, Account, Supply, or Subbasin (SimObj with type CompObj), but perhaps not Exchange.

Figure 3. The Unit Scheme Editor dialog



5.1 Supporting configuration of DataObj slots

For slot name rules, the current Slot Name Chooser dialog needs to be extended to provide support for existing slots on DataObjs. It is currently based on object prototypes, which of course contain no slots in the case of DataObj's. For fully specialized rules, no special treatment is necessary. See online document "Unit Scheme support for Dat Objects".

6 Making the transition away from the resource database (riverwareDB file)

The riverwareDB file (resource database) provides a mechanism for specifying the default configuration of slot values on new objects. If such a file is present, we could create an analogous unit scheme, i.e., one that displays slot values as they would be if their managing object had been created in the presence of that riverwareDB file. We expect that for most models this riverwareDB unit scheme would be consistent with the model transition unit scheme, but in theory they could be completely different. Probably, when there is a riverwareDB file present, we should create the corresponding unit scheme, and make it the currently active scheme. If the user loads a model file, this will switch the model scheme, though we could ask them for confirmation.