Tabular Browser Design Documentation

Nicholas DiOrio, National Renewable Energy Laboratory

March 3, 2016

1 Introduction

The tabular results browser in SAM provides the user the capability to easily display and compare matrix data. This document is intended for internal use only to provide reference for the design of the tabular browser in case of future problems. Basic code paths will be listed, as well as intended behavior. For functions which have been particulary difficult to get correct, inputs, outputs, and additional details will be explained.

2 Code Paths

2.1 Running a simulation

The general code path which occurs when running a simulation is:

```
MainWindow::OnCaseMenu
   CaseWindow::RunBaseCase
      Simulation::DispatchThreads
      CaseWindow::UpdateResults
         CaseWindow::Setup
            TabularBrowser::Setup
               TabularBrowser::UpdateAll
         CaseWindow::GetTabularBrowser
            TabularBrowser::SelectVariables
               TabularBrowser::UpdateAll
```

As can be seen, the tabular browser is updated twice when a simulation is run. More investigation needs to be done to understand if this is correct. Within TabularBrowser, a new or updated simulation results in calls to UpdateAll. This function encapsulates checks for adding variables, removing variables, and updating variable sizes on the existing grids. The following illustrates the order of the calls within UpdateAll, with the understanding that some logic exists to choose when to call these functions or not.

```
TabularBrowser::UpdateAll
   TabularBrowser::UpdateSelectionList
   TabularBrowser::RemoveUnusedVariables
   TabularBrowser::ProcessAdded
   TabularBrowser::UpdateSelectionExpansion
```

2.2 OnPageClosed

When a page in the tabular browser is closed, a page-closed event is generated, at which point internal data structures need to be cleaned up, and the existing grids need to be updated. The primary tasks which must occur are to remove all of the variables on the closed-grid and then redraw the existing grids.

```
TabularBrowser::OnPageClosed
   TabularBrowser::ProcessRemovedAll
   TabularBrowser::UpdateAll
```

2.3 OnPageChanged

Every time a user clicks on a notebook tab which results in a change of page from the previously selected tab to the new selection, a page-changed event is generated. The event handling for this is relatively simple. The code simply points local variables to reflect the currently selected grid.

2.4 OnVarSel

Every time a user clicks to add or remove a variable from the current grid, a variable-select event is generated. To handle this event, the code simply checks whether the check-box is selected or deselected and then processes the addition or removal:

```
TabularBrowser::OnVarSel
   TabularBrowser::ProcessAdded
   TabularBrowser::ProcessRemoved
   TabularBrowser::SetLastSelection
```

3 Data Structures

The internal data structures relevent to the TabularBrowser are:

m_notebook	the wxAuiNotebook which holds a wxExtGridCtrl on every tab
m_grid	the currently selected wxExtGridCtrl
m_gridTable	the underlying table containing the data for the currently selected grid
m_gridMap	a map which associates grid size to a pointer to that grid
m_gridTableMap	a map which associates grid size to a pointer to the underlying data table
m_tabLabelsMap	a map which associates grid size to the label of the tab
m_selectedVars	a string array containing names of every variable selected across all grids
m_selectedVarsBySizeMap	a map which associates variable name to the grid size
m_selectedVarsMap	a map which associates grid size to the array of variable names on that particular grid
m_numberOfTabs	the number of tabs (grids)
m_lastSize	the size of the last selected grid
m_key	an integer which increments when a matrix is added, allowing a way to keep track of grids of the same size

4 Main Functions

4.1 ProcessAdded

Inputs
`wxString name` - the name of the variable to add

Outputs
`none`

Description The `ProcessAdded` function is called for one variable at a time. The function checks whether or not the size of the grid associated with the variable has changed. If the size has changed, `ProcessRemoved` is called before proceeding. If `ProcessAdded` is called and a variable needs to be added, it will be added to `m_selectedVars`, `m_selectedVarsMap`, and `m_selectedVarsBySizeMap`. The variable size will be used to update `m_lastSize`. Finally, the notebook, grid, and case will be updated.

4.2 ProcessRemoved

Inputs
`wxString name` - the name of the variable to remove
`bool update_grid` - boolean choice on whether to call for grid update

Outputs
`none`

Description The `ProcessRemoved` function is called for one variable at a time. The variable is removed from `m_selectedVars`, `m_selectedVarsMap`, and `m_selectedVarsBySizeMap`. In the event that more variables exist on the current grid, the variable size will be used to update `m_lastSize`, and if `update_grid` is true, the grid and case will be updated. If no more variables exist on the current grid, a call is made to `ProcessRemovedAll`.

4.3 ProcessRemovedAll

Inputs
`ArraySizeKey removed_size` - the size of grid to remove variables for

Outputs
`none`

Description The `ProcessRemovedAll` function is called to remove all similiarly-sized variables from internal data structures, including the deletion of the underlying grid, grid table, and notebook page. The variables are removed from `m_selectedVars`, `m_selectedVarsMap`, `m_selectedVarsBySizeMap`, `m_tabLabelsMap`, and `m_gridMap`, `m_gridTableMap`. The variable `m_numberOfTabs` is decremented.

4.4 GetVariableSize

Inputs
`int index` - the index of the variable within m_selectedVars to retrieve size information for

Outputs
`ArraySizeKey var_size` - the variable size. If the variable size is not found, the returned size

will have 0 rows, 0 columns

Description The function first calls `GetStoredVariableSize` to search internal TabularBrowser data structures for the variable size. If the information is not stored, the function calls `GetSimulationVariableSize`, which queries the simulation object for information about the variable.

4.5 CheckSizeChanged

Inputs
`int index` - the index of the variable within m_selectedVars to retrieve size information for

Outputs
`bool changed` - true if the variable size is changed, else false

Description This function was designed to detect a variable size change in the event that a simulation is re-run with the Lifetime setting changed. The function first calls `GetStoredVariableSize` to search internal TabularBrowser data structures for the variable size. If the information is stored, the function calls
`GetSimulationVariableSize`, which queries the simulation object for information about the size of the variable after the most recent simulation. Of note is that if the variable size is for a matrix, there is also an associated key. The `ArraySizeKey` retrieved from the simulation object duplicates the key present in the stored object so that the two sizes can be compared. If the stored size does not match the simulation size, the function returns true, else false.

4.6 RemoveUnusedVariables

Inputs
`none`

Outputs
`none`

Description This function was designed to account for the scenario where a simulation is re-run with the Lifetime setting changed, such that some variables will have changed size and disappear from the grids they were previously on. Previously, SAM would crash under this scenario. The function loops over `m_selectedVarsMap`. In each loop, it extracts the variable size and list of variables at that size. The simulation object is queried for the each variable on the grid. If the variable is not found within the simulation, `ProcessRemoved` is called for that variable. If the variable is still found, a call is made to `CheckSizeChanged`. If the variable has changed size, `ProcessRemoved` is called, followed by `ProcessAdded` to update internal data structures with the correct size information. Finally, the grid and case are updated.

4.7 UpdateNotebook

Inputs
`ArraySizeKey grid_size`
`wxString name`

Outputs
`none`

Description This function was designed to update the tabs available within the notebook. If `m_gridMap` does not contain a grid at the input size, a new grid is created and placed onto a new page. The `m_tabLabelsMap` is updated with the appropriate label for that variable size, and `m_numberOfTabs` is incremented.

4.8 UpdateGridSpecific

Inputs
`wxExtGridCtrl *& grid` - a pointer to the grid to update, called by reference
`ResultsTable *& grid_table` - a pointer to the ResultsTable, called by reference
`wxArrayString selected_vars` - the selected variable to load on the grid

Outputs
`none`

Description This function was designed to update the variables on the selected grid. Of note, is that the grid and grid map are passed as references so they can be internally allocated, modified, and returned. The function always deletes the underlying ResultsTable for the current grid, and reloads the information with a call to `LoadData`. Grid properties such as the size of row labels, column size, column labels, font, and font weight are specified. When the data is loaded, the grid is drawn and refreshed on the screen.

4.9 UpdateCase

Inputs
`none`

Outputs
`none`

Description This function is designed to update the current case data browser variables with the variables selected within the TabularBrowser.

4.10 LoadData

Inputs
`Simulation *results` - the simulation object obtained from running a new simulation
`wxArrayString &vars` - the variables to load

Outputs
`none`

Description This function is called from `UpdateGridSpecific` and is designed to load the selected variables from the simulation object onto the appropriately sized grid and grid table. The variables are loaded from the simulation object and classified as matrices, arrays, or single values. For matrix data, optional user-interface hints are parsed and the appropriate row and column labels are assigned.

5 User Interface Hints

This powerful feature allows a programmer to specify custom output hints within `ssc` compute modules. User interface (UI) hints specified within `ssc` are passed and parsed within the Simu-

lation object for use in the TabularBrowser (and potentially other places). Currently, UI hints are used to:

- Specify custom header labels within the data selection list
- Specify custom row and column labels for matrix outputs

The set of currently available UI hints and how to use them will be documented below.

5.1 UI hints for custom group labels

UI hints for custom group labels are implemented in `SAMnt` within `PopulateSelectionList`. The key to implement a custom group label is: `GROUP`

Currently, the implemented hints are:

`UR_MTP`	Utility rate data by tierperiod
`UR_AM`	Utility rate data by year

5.2 UI hints for row labels

UI hints for row and column labels are implemented in `TabularBrowser::LoadData`. The labels are defined in custom function within `TabularBrowser` such as `MakeTimeOfDay`.

The key to implement a row label is: `ROW_LABEL`.

The currently available row labels are:

`HOURS_OF_DAY`	lists the rows as hours of the day. Assumes 24 rows, goes from 12 am to 11 pm
`UR_PERIODS`	lists the rows as utility rate periods. Assumes 12 rows, goes from period 1 to 12
`UR_PERIODNUMS`	lists the rows as period numbers, with a reduced matrix size to show only non-zero periods
`UR_TIERS`	lists the rows as tier numbers. Assumes 6 rows, goes from tier 1 to 6
`UR_TIERNUMS`	lists the rows as tier numbers, with a reduced matrix size to show only non-zero tiers
`MONTHS`	lists the rows as months. Assumes 12 rows, goes from January to February
`NO_ROW_LABEL`	lists the row labels as blanks.

5.3 UI hints for column labels

The key to implement a row label is: `COL_LABEL`.

HOURS_OF_DAY	lists the columns as hours of the day. Assumes 24 rows, goes from 12 am to 11 pm
UR_PERIODS	lists the columns as utility rate periods. Assumes 12 rows, goes from period 1 to 12
UR_PERIODNUMS	lists the columns as period numbers, with a reduced matrix size to show only non-zero periods
UR_TIERS	lists the columns as tier numbers. Assumes 6 rows, goes from tier 1 to 6
UR_TIERNUMS	lists the columns as tier numbers, with a reduced matrix size to show only non-zero tiers
MONTHS	lists the columns as months. Assumes 12 rows, goes from January to February
XY_POSITION	lists the columns as "X Position (m)" and "Y Position (m)"
OPTICAL_EFFICIENCY	lists the columns as "Azimuth Angle (deg)", "Elevation Angle (deg)", and "Optical Efficiency (-)"
FLUX_MAPS	lists the columns as "Azimuth Angle (deg)", "Elevation Angle (deg)", and "Panel i Absorbed Flux (kW/m2)" for i:N panels

5.4 UI hints for format specifiers

Format specifiers can be used to format data output in a certain way, for instance, formatting currency or time.

The key to implement a format specifier is: FORMAT_SPEC.

Only one format specifier is currently available. In the event that no format specifier is listed, matrix data will be formated as a real value.

CURRENCY	lists the data in a currency format

5.5 Adding a UI hint within SSC

To implement the UI hints in SSC, go to the variable table at the top of the compute module. The last field in the table is for UI hints. To enter a UI hint, enter the key and associated value like:

```
"KEY=VALUE"
```

To enter multiple hints, seperate them by commas with no spaces, such as:

```
"KEY1=VALUE1,KEY2=VALUE2"
```

This is illustrated by the real example from cmod_utilityrate4

```
"COL_LABEL=MONTHS,FORMAT_SPEC=CURRENCY,GROUP=UR_AM"
```

6 Bugs and Pitfalls

6.1 Changing from single year to lifetime mode

When changing from a single year to lifetime mode or vice versa, special care must be taken within the tabular browser implementation to properly update internal data structures. Crashes would previously occur when a dual-variable (one that changes size depending on the "Lifetime"

setting" was listed in the results browser under the original size and the simulation was rerun under a different setting.

6.2 Maintaining window docking between simulations

When making changes to the tabular browser, it is important to ensure that custom window docking is maintained when re-running a simulation. It is not entirely clear what causes the docking positions to be deleted, possibly calling `UpdateGridSpecific` from `ProcessRemoved`, or a change to `m_notebook`.