TouchTreeGrid

TouchTreeGrid is an extremely versatile and easy to implement Sencha Touch 2.1x, 2.2x custom component that supports Tree Grids, Standard Grids and traditional Accordion view layouts (all by the same component). Column sorting and custom data renderings such as comma formatting, currency, percents and custom colors for negative vs. positive values are included. **TouchTreeGrid** utilizes Sencha's Ext.dataview.List component and Disclosure events are supported for leafs and optionally for category rows.

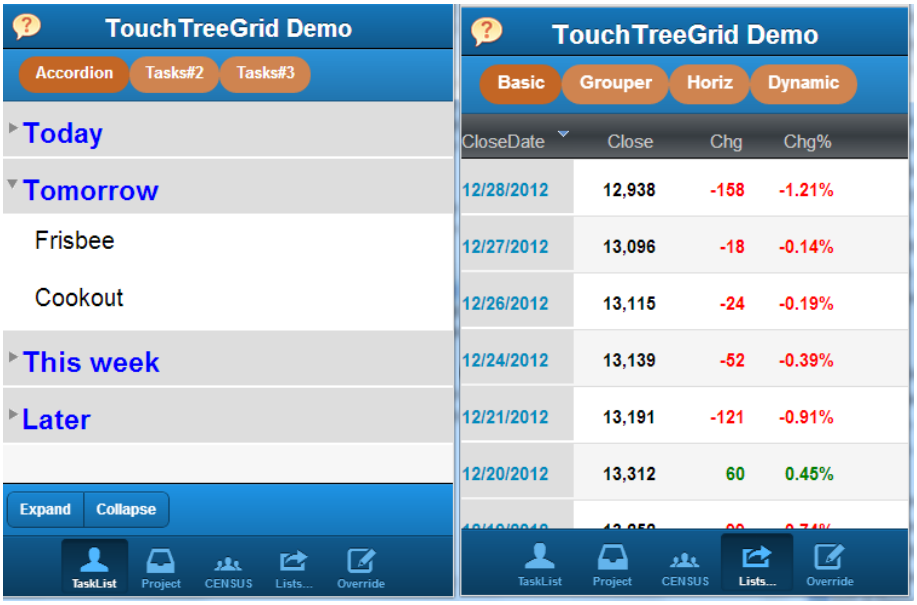
TouchTreeGrid was developed entirely within Sencha Architect (v2.2) designer product and can also be used without Architect. An Architect component is provided for import into your toolbox (TouchTreeGrid.xdc). Using Sencha Architect you can implement this component for new grid implementations very rapidly. Both basic and advanced examples are included in the download. Scrolling and overall functionality was found to be very fast for a larger data example containing +3000 rows and 3 category levels[1]. This software can be downloaded at https://github.com/swluken/TouchTreeGrid and is free to use (refer to associated MIT.LISCENSE) .

Links to live demo contained within READ.ME file on the github site.

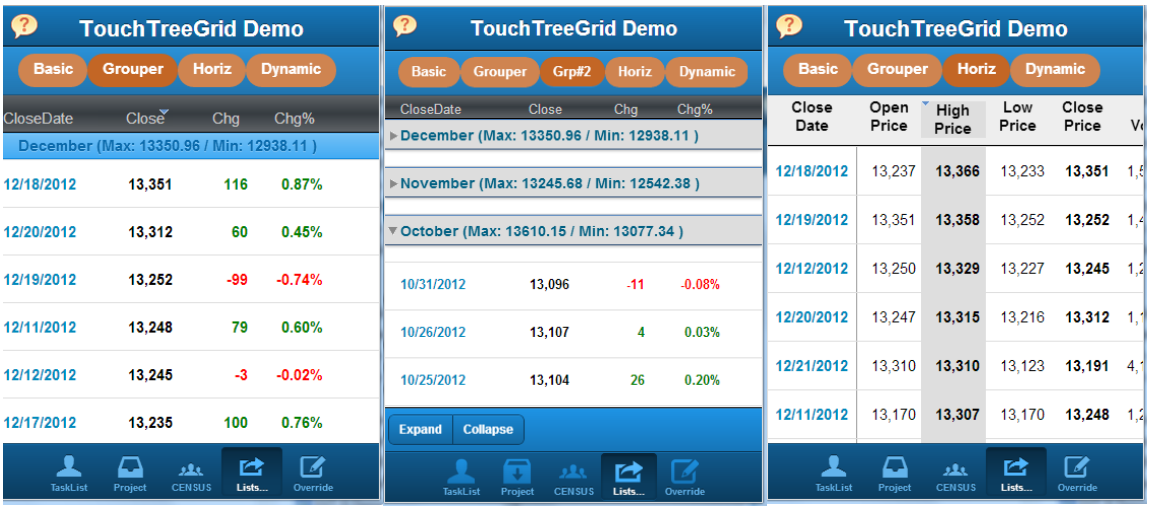
Contents

[1] Devices tested: Android 4, IPhone 5 and IPad (3rd generation)

SAMPLE: Basic Accordion and Standard Grid Examples (with Column Sort)

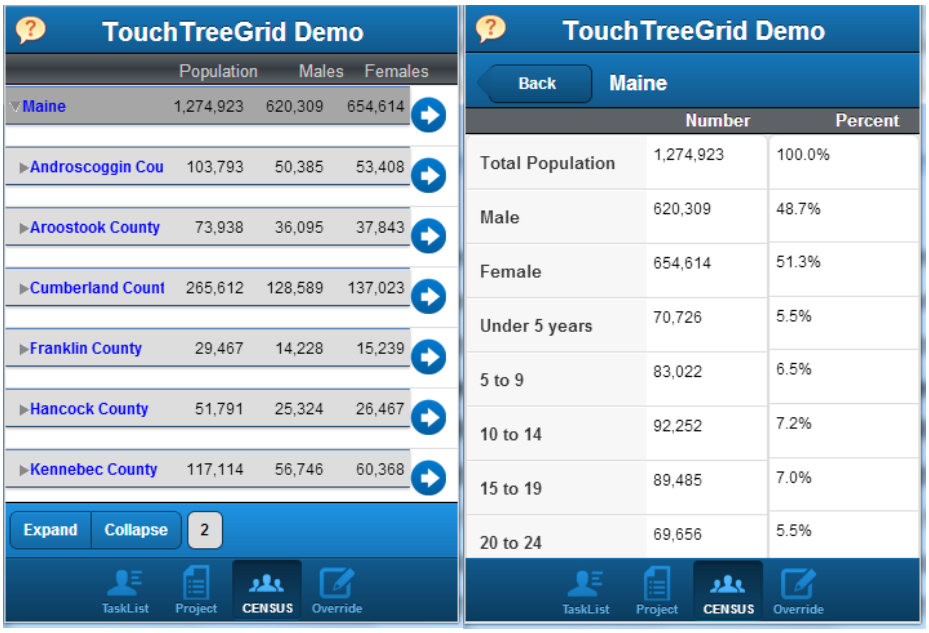


SAMPLE: Standard Grid with Grouper, Horizontal Scrolling

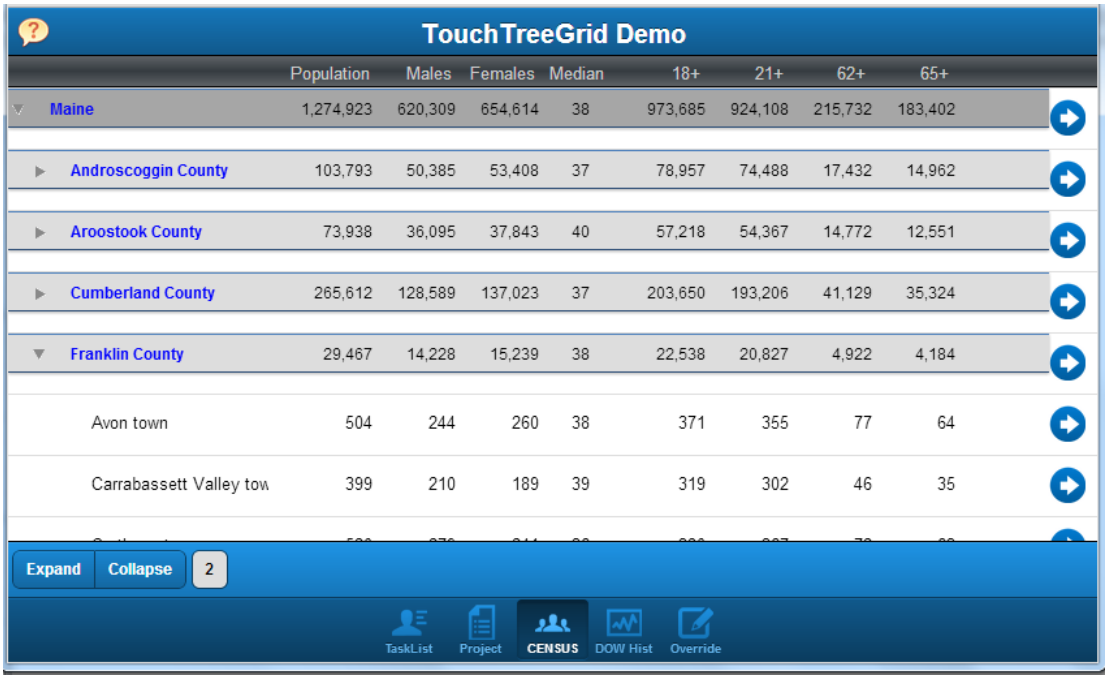


SAMPLE: Project Status – 3 Variations using CSS

SAMPLE: US Census – Phone Portrait



SAMPLE: US Census – Tablet Landscape



SAMPLE: US Census – Tablet Portrait

(# of columns auto-updated as orientation is changed)

Intended Audience

This document is laid out in such a way that it is hopefully useful for beginner and advanced developers as well as for designers who want to learn what the capabilities are.

Credits

TouchTreeGrid design was modeled after Mitchell Simoens's extremely popular Ext.ux.touch.grid (https://github.com/mitchellsimoens/Ext.ux.touch.grid and http://www.sencha.com/forum/showthread.php?150431-Ext.ux.touch.grid)

Many thanks to Shinobu Kawano's initial work on how tree stores can be used in Sencha Touch for this purpose (https://github.com/kawanoshinobu/Ext.ux.AccordionList).

Thank you to Sencha Team for their continued work on such an awesome suite of products! Of course if you are reading this then you already know that ☺

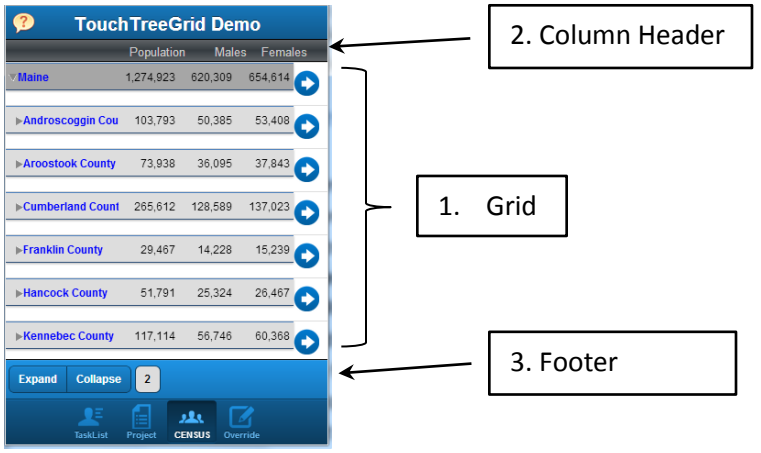
TouchTreeGrid

TouchTreeGrid includes the following **Basic** and **Advanced** features:

Basic Features

- The TouchTreeGrid component automatically creates following components simply by linking it as a child to a parent container and specifying a few configurations:

    1. Grid displaying Detail and collapsible Category rows
    2. Optional header row for column headers
    3. Optional footer containing buttons to expand all, collapse all, collapse to specific depth (aka “level”).



    Category rows are indented for each depth. A smaller arrow icon is used to minimize display consumption. By default the first column of data is truncated for indented depths so that subsequent columns line up.

- Following are minimum recommended configurations to create a Tree Grid:
    1. store- must of type Ext.data.TreeStore
    2. defaultCollapseLevel = 1,2 3, etc… (99 for fully expanded)
    3. itemId and listItemId
    4. columns array (refer below)

- Following are minimum recommended configurations to create a basic Grid:
    1. store- any store type can be used except Ext.data.TreeStore
    2. simpleList = true
    3. columnSorting = true
    4. itemId and listItemId
    5. columns array (refer below)

- Following are minimum recommended configurations to create a basic single expanded style Accordion:
    1. store- must of type Ext.data.TreeStore
    2. singleExpand = true
    3. defaultCollapseLevel = 1
    4. itemId and listItemId
    5. columns array (refer below)

- Column configurations are defined within a COLUMNS array (refer to Appendix A for examples and complete list):
    1. header - Text to include in column header
    2. dataIndex - data column from treestore/store
    3. width - CSS format width for this component (suggest % or em's)
    4. style - Additional CSS styling commands for detail rows
    5. categStyle - Additional CSS styling commands for category rows (categStyle not used/required for Simple Grids)
    6. headerStyle - Additional CSS styling commands for column header row
    7. sortable - allow sorting for this column

- Column arrays can be defined and applied based on device configuration. For example Phone portrait, Phone landscape, Tablet portrait, Tablet landscape could each display a different number of columns based on screen width. Refer to Census example to see how number of grid columns is auto-updated as device is rotated. Run phone vs. tablet in browser and resize to simulate orientation similar to:

    http://localhost/TouchTreeGrid-master/TouchTreeGrid_Advanced/app.html?deviceType=**Phone**

    http://localhost/TouchTreeGrid-master/TouchTreeGrid_Advanced/app.html?deviceType=**Tablet**

    For larger applications it is suggested that you store all column configurations in a table on the server. The column arrays could be returned in the same back-end request that returns the data. Definitions would include column configurations, identifier for each grid, device/orientation, and possibly client-specific configurations. This reduces the footprint size that is downloaded on the device and allows for easy configuration changes without a code change.

- Disclosure icon can be included on detail and category rows where the "disclose" event would be processed within a controller (onItemDisclosure=true).

- All styling defined within CSS file (refer to Appendix B for detailed documentation)

Advanced Features

- Refer to APPENDIX D for detailed config documentation
  - Special note regarding "**additionalListConfigs**" - Any config supported by Ext.dataview.List component not directly supported in Appendix D can be applied to the auto-generated Grid list component. Configs defined via additionalListConfigs would override any similar configs defaulted by TouchTreeGrid.

- Option to assign unique colors to Category rows by expansion depth:

```
categDepthColorsArr: ['#808127', '#949569', '#C5C678'] <= Depths 1,2 and 3 colors
```

- Specify the screen width percentage stepsize for each expanded depth:
  - categIndentPct – defaults to 3%
  - colNumberToTruncateForIndents – by default column #1 is truncated by (categIndentPct * depth) to line up subsequent columns.

- Override default screen width taken by expand/collapse arrow via arrowPctWidth config:
  - Default = 4%
  - 2% would be sufficient for Tablet displays

- Item Disclosure functionality:
  - Specify display of disclose icon for leafs only via discloseProperty='leaf'
  - Use CSS class ".touchtreegrid-disclose-spacer " to provide percent width spacing in header row to line up columns when disclosure icon is displayed (refer to Appendix B)

- Custom column render functions are supported a little differently than in Mitchell Simoen's Ext.ux.touch.grid.

  - Instead of defining the function within the Columns Array for each column, an list of custom render functions can be defined in 'renderers' object within each linked instance of TouchTreeGrid[2]:

```
renderers:
 { renderer_displayIn1000s: function(value)
     {return this.formatNumbers(Math.round(Number(value)/1000), 0);},

   renderer_formatWithColor: function (value)
     {var clr = (value >= 0) ? 'green' : 'red';
       return '<span style="color:' + clr + ';">' + this.formatNumbers(value,0) +
              '</span>';}
 },
```

[2] One reason for this is that we can't easily store and pass functions to column array if stored in a table on the server. This also simplifies repetitive assignments within the array

- Examples and Notes regarding functions:
  - this.render_displayIn1000s(1234567) = 1,235
  - this.renderer_formatWithColor(-23.45) displays cell value in red (vs. green for positive)
  - custom function parameters are passed in the function calling string (refer below)
  - You could add all common functions used across your application directly in TouchTreeGrid.js renderer object and they would be available for all grid implementations (but take care when updating new version of this component!) … I will come up with better way to do this in future release to simplify upgrades.

- Note that custom rendering functions can call other built-in functions or formatNumbers() function provided with TouchTreeGrid (refer below)

- The Columns array would include a string representing the function call to make:

```
columns : [{
     header: 'Holdings(k)',
     dataIndex: 'HOLDINGS',
     width: '15%',
     style: 'text-align: right;',
     categStyle: 'text-align: right;',
     headerStyle: 'text-align: right; color: #ccc;',
     renderer: 'this.renderer_displayIn1000s(values.HOLDINGS)'
            },
     [{
     header: '%Change',
     dataIndex: 'PCT_CHG',
     width: '15%',
     style: 'text-align: right;',
     categStyle: 'text-align: right;',
     headerStyle: 'text-align: right; color: #ccc;',
     renderer: 'this.renderer_formatWithColor (values.PCT_CHG)'
            },
     Etc…
```

- Notes regarding renderer function calls:
  - 'values' is an object containing elements for every item of each data record. Hence, every data field value is available for use within renderer functions.
  - According to Ext.data.NodeInterface documentation the following fields are automatically added to the tree store upon load if not already existing and can also be referenced:
    parentId, index, **depth**, **expanded**, expandable, **checked**, **leaf**, cls, iconCls, root, isLast, isFirst, allowDrop, allowDrag, loaded, loading, href, hrefTarget, qtip, qtitle

- Built-in **formatNumbers(value, decPlaces, prefix, suffix, thouSeparator, decSeparator)** column formatting function provided with TouchTreeGrid

- Parameters (all fields optional except value)
  - value – number to format
  - decPlaces - # decimals places (default is 2 decimals if omitted)
  - prefix- use to prefix '$' sign or other to number
  - suffix – use to append '%' sign or other to number
  - thouSeparator – default = ','
  - decSeparator – default = '.'

- Examples
  - renderer: 'this.formatNumbers(values.TotalPopulation, 0)' => 1,234,567
  - renderer: 'this.formatNumbers(values.NAV_PCT, 4, "", "%")' => 12.3456%
  - renderer: 'this.formatNumbers(values.NAV_AMT, 0, "$")' => $1,234,567

- Events fired for each item tap for custom processing: "**leafItemTap**" and "**nodeItemtap**". See Task example controller functions onTask2LeafItemTap () and onTask2NodeItemTap ()

- Default sort functionality can be overridden by specifying **customColumnSortEvent** config. Event specified by this config will be fired and can be processed within controller for custom sorting. This allows for custom column sort functionality without modifying the component. Example would be when sort needs to occur on the server-side or to toggle between more states: ASC, DESC, ABS(ASC), ABS(DESC), etc... Refer to logic in handleColumnSort() method within TouchTreeGrid for parameters passed with event.

- Following additional styling configs supported in columns[] array for highlighting sorted columns:
  - styleSorted (overrides 'style' for sorted column)
  - categStyleSorted (overrides 'categStyle')
  - headerStyleSorted (overrides 'headerStyle')

- categColumns[] config array supported as means to defined category styling totally unique from columns defined by width attribute in columns[] config array. This allows us for example to use TreeGrid to support "grouper" functionality , with expand/collapse functionality. The grouper expression can be any long text totally independent from leaf rows:

- Supports **disableExpandCollapse** config to disable expand/collapse functionality.

- PullRefresh implemented as follows:
  - Define 'plugins' object in "linked" TouchTreeGrid (also refer to Example2)

```
itemId: myExample,
listPlugins: {
        xclass: 'Ext.plugin.PullRefresh',
        listeners : {
        latestfetched : function()
{this.up('touchtreegrid').fireEvent('pullrefresh', this.up('touchtreegrid'));}
    }
/* refreshFn not supported for Touch 2.2 */
/* refreshFn: function(plugin) {this.up('touchtreegrid').fireEvent('pullrefresh',
this.up('touchtreegrid'));} */
                }
```

  - Within ProjectController:

```
refs: {
            example2: '#example2',

control: {
            "container#example2": {
                pullrefresh: 'onExample2ListPullrefresh'
            },
```

  - onMyExampleListPullrefresh() function
    - calls loadExample2Store() which is same function used to initially load data

- Implement custom Expand/Collapse functionality. By default Touch's built-in Node expand/collapse methods are used. For larger data sets I found it was faster to reset the expand depths manually in the controller and reload the data. Hence, a "customExpCollapseEvent" configuration is provided for this purpose. If defined, an event by the specified name will be fired (i.e. instead of default collapse logic) which can be picked up in the controller.
  Note: actual example not provided, but following is guide for implementation:
  - Within linked TouchTreeGrid instance

- customExpCollapseEvent : 'myExampleExpCollapseEvent'
- itemId : 'myExample'

- Within Controller:

```
refs: {
            myExample: '#myExample', etc…}

control : {
"container#myExample": {
                myExampleExpCollapseEvent: 'onMyExampleExpCollapse'
            }, etc…}
```

- 'onMyExampleExpCollapse" function
  - Rebuild tree specifying 'expanded' attribute
  - Update treestore

- Define what depth to auto-expand/collapse depths to upon initialization/refresh:
  - Specify default collapse-to depth via: defaultCollapseLevel = 0 to 99
    - 0 for fully collapsed
    - 1 to collapse to 1st level (would be same as 0 if a single root node did not exist)
    - 99 for fully expanded (default)
  - Disable this feature via: applyDefaultCollapseLevel=false
    (this would be used if collapse level defined when tree is created)

- Control display of collapse-to buttons
  - Disable display via, includeFooterLevels = false (default is true)
  - Disable application of colors defined by categDepthColorsArr[] for category row colors to the collapse-to buttons via, categoryDepthColorButtons= false (default = true)

- List item **pressedCls** is supported and defaults to: 'touchtreegrid-item-pressed'
  - If disableSelection=true, the pressedCls is still applied momentarily as user is pressing a list item for disclosure.
  - TouchTreeGrid updateStore function contains logic to update this to empty string if disabledSelection=true thereby not applying any class to the press event.
  - Otherwise user can specify own pressedCls and define in CSS file if disableSelection=false.
  - Refer to TouchTreeGrid.css for examples on it's use.

- List item selectedCls is supported and defaults to: 'touchtreegrid-item-selected'
  - Refer to TouchTreeGrid.css for examples on it's use.

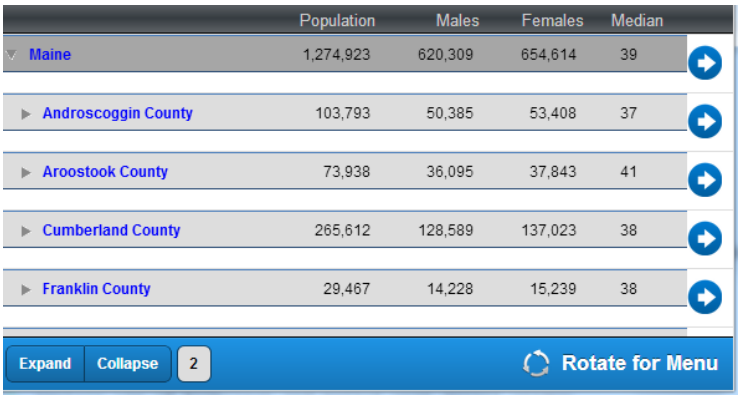
- Manual HTML specifications for Header, Category and Detail rows
  - Manually provide HTML in following configurations of linked instance of TouchTreeGrid: headerTplOverride, categItemTplOverride, contentItemTplOverride

- Within Architect's config panel you can specify the data type as Object to edit formatted text
- Refer to "Manual" example.

- itemHeight and variableHeights
  - List itemHeight within TouchTreeGrid defaults to Sencha's default value of 47 pixels and can be updated. However, since only pixels can be defined at time of this writing (because of scroller implementation as I understand), I found that playing with this number does not always render as expected on some displays (Android for example). I would wait until 'em' is supported before using this.
  - variableHeights = true is the default. Refer to Sencha documentation, but as I understand setting to false along with customizing itemHeight improves performance.

- Header, Category and Detail (aka 'content') rows all utilize outer element which represents the shaded portion of category rows for instance:
  -
  - Additional custom styling can be appended to this div statement specifying one or more of the following congurations:
    - styleHeaderRow
    - styleCategRow (not applicable for simpleList = true)
    - styleContentRow

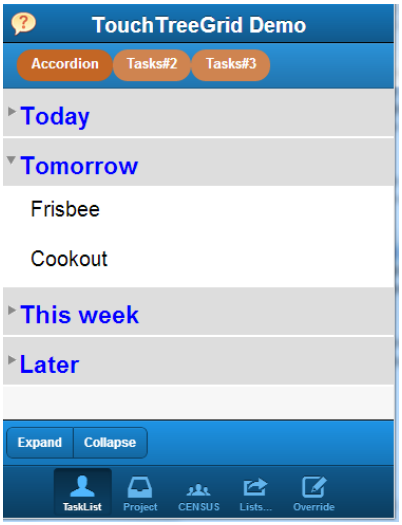
- If you want to change the "Recycle" icon when rotating phones to landscape you can specify your own url in the linked instance of TouchTreeGrid via,
  landscapeIcon = './resources/images/**myIcon**.png'



Summary of Provided Examples

(refer below for detailed configuration documentation)

TaskList => Accordion



Basic accordion example demonstrates:

i. Single expand accordion style treegrid
ii. No lines separating leaf rows

Program flow:

iii. Controller: TasksController.js
iv. View config: Main.js => TasksContainer.js => itemId = 'firstexample'
v. Data:
    1. TouchTreeGrid.store.TaskAccordion (storeId = TaskAccordionStore)
    2. Model: TouchTreeGrid.model.Task
    3. TreeStore (data defined directly in root property)
    4. Autoload = true

Notable configs:

vi. singleExpand = true
vii. defaultCollapseLevel =1
viii. style: 'text-align: left; font-size: 1.5em;'
ix. categStyle: 'font-weight: bold; font-size: 2em; padding-top: .2em; text-align: left; color: blue;'
x. cls: ['x-touchtreegrid-list', 'x-touchtreegrid-list-accordion']

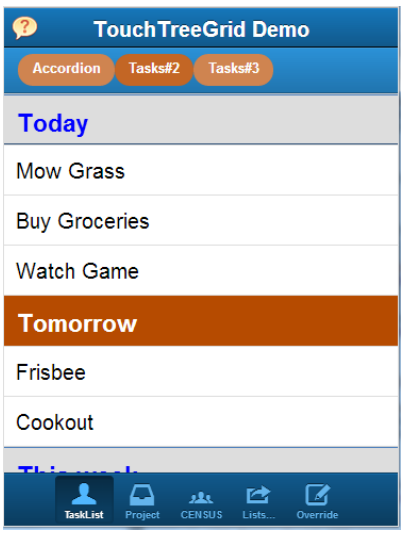
Notable CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
/* 2.1 selectors to eliminate lines  ... 2nd set is for Touch 2.2 */
.x-touchtreegrid-list-accordion .x-list-normal .x-list-item .x-dock-horizontal,
.x-touchtreegrid-list-accordion .x-list-normal .x-list-item.x-list-item-tpl{
```

```
   border: none;
}

.x-touchtreegrid-list-accordion .touchtreegrid-list-content{
    padding:.6em 0 0 0;
}

.x-touchtreegrid-list-accordion .touchtreegrid-list-categ {
  border-top: none;
  border-bottom: none;
  -webkit-box-shadow: none;
  min-height: 45px !important;
}

.x-touchtreegrid-list-accordion .touchtreegrid-details-img {
  /* category row arrow centered in row */
  margin-top: .5em !important;
}
```

Sample data stored directly in Root of store: TouchTreeGrid.store.TaskAccordion

TaskList => Tasks#2



a. Demonstrates:
   i. Disabled expand/collapse within scrolling window
   ii. Leaf plus node row selection with custom CSS styling

b. Program Flow

   i. Controller: TasksController.js
   ii. View config: Main.js => TasksContainer.js => itemId = 'task2'
   iii. Data:
      1. TouchTreeGrid.store.Task2Store (storeId = task2Store)
      2. Model: TouchTreeGrid.model.Task
      3. TreeStore (data defined directly in root property)
      4. Autoload = true
   iv. Events:

```
control: {...
        "container#task2": {  // Trapping leaf and node taps !!
           leafItemTap: 'onTask2LeafItemTap',
           nodeItemTap: 'onTask2NodeItemTap'
        },
```

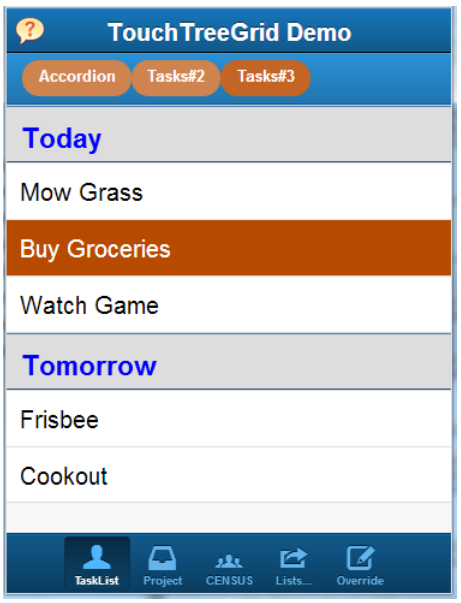
c. Notable Configs:
   i. disableSelection = false
   ii. arrowPctWidth = '0'
   iii. disableExpandCollapse = true

d. CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
.x-touchtreegrid-list-task2 .touchtreegrid-list-categ {
```

```
    min-height: 47px !important;
/* Selected row is still painted, but underneath webkit box which is sized to hide selection */
}

/** Begin Task2 CSS required to support row selection for category and content rows **/
.x-touchtreegrid-list-task2 .touchtreegrid-list-content{ /* Detail (leaf) rows  */
    padding:.6em 0 0 0;
    background-color: transparent;  /* Needed for row selection */
}

.x-touchtreegrid-list-task2 .x-touchtreegrid-item .touchtreegrid-item-selected,
.x-touchtreegrid-list-task2 .touchtreegrid-item-selected,
.x-touchtreegrid-list-task2 .touchtreegrid-item-selected .touchtreegrid-list-content-cell {
    background-color: #b64b00 !important;  /* color for selected row */
    color: white !important;  /* Change selected text color to white */
}

.x-touchtreegrid-list-task2 .touchtreegrid-item-pressed {
    background-color: #ecd2bf !important;  /* color for selected row */
}

.x-touchtreegrid-list-task2 .touchtreegrid-item-selected .touchtreegrid-list-categ {
    background-color: transparent !important;
    color: white !important;
    border-top: none;
    border-bottom: none;
    -webkit-box-shadow: none;
}

.x-touchtreegrid-list-task2 .touchtreegrid-item-selected .touchtreegrid-list-categ-cell {
    color: white !important;
}
```

TaskList => Tasks#3



a. Demonstrates
   i. Same as Tasks#2 except non-scrolling and row selection for leafs only

b. Program Flow
   i. Controller: TasksController.js
   ii. View config: Main.js => TasksContainer.js => itemId = 'task3'
   iii. Data:
      i. TouchTreeGrid.store.Task3Store (storeId = task3Store)
      ii. Model: TouchTreeGrid.model.Task
      iii. TreeStore (data defined directly in root property)
      iv. Autoload = true
   iv. Events:

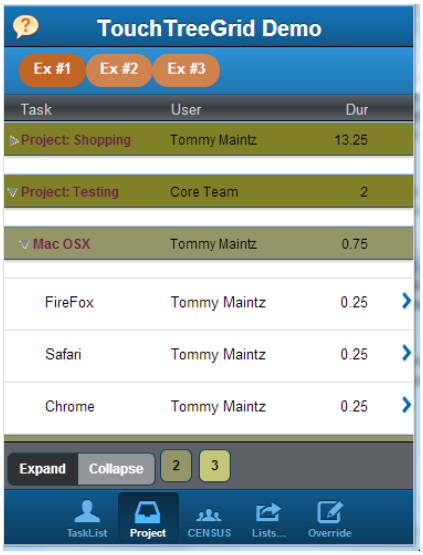
```
control: {.....
        "tabpanel#tasksTabPanel": {
           activeitemchange: 'onTasksTabpanelActiveItemChange'
        },
        "container#task3": {  // Only trapping leaf taps for this example !
           leafItemTap: 'onTask3LeafItemTap'
        }

    Activeitemchange()
    if (newcont === 'task3cont'){
    // One way to disable as list config "scrollable: false" does not work as expected
       grid.getScrollable().getScroller().setDisabled(true);
       Ext.Msg.alert('Non-scrolling!');
    }
```

c. Notable Configs
   i. disableSelection = false
   ii. disableExpandCollapse = false

d. CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
.x-touchtreegrid-list-task3 .touchtreegrid-list-categ {
    min-height: 47px !important;
/* Selected row is still painted, but underneath webkit box which is sized to hide selection */
}

/** Begin Task2 CSS required to support row selection for category and content rows **/
.x-touchtreegrid-list-task3 .touchtreegrid-list-content{ /* Detail (leaf) rows  */
    padding:.6em 0 0 0;
    background-color: transparent;  /* Needed for row selection */
}

.x-touchtreegrid-list-task3 .x-touchtreegrid-item .touchtreegrid-item-selected,
.x-touchtreegrid-list-task3 .touchtreegrid-item-selected,
.x-touchtreegrid-list-task3 .touchtreegrid-item-selected .touchtreegrid-list-content-cell {
    background-color: #b64b00 !important;  /* color for selected row */
    color: white !important;  /* Change selected text color to white */
}

.x-touchtreegrid-list-task3 .touchtreegrid-item-pressed {
    background-color: #ecd2bf !important;  /* color for selected row */
}
```

Project => Ex#1



a. Demonstrates
   - Initially expands nodes as individually defined in JSON tree store
   - color-specific category rows with matching colors on collapse-to buttons
   - overridden color scheme for expand/collapse button and toolbar (treegriddemo.css)
   - custom IPhone-style disclose icon (treegriddemo.css)
     Note: you may or may not want to use this icon depending on your experience with sensitivity when touching icon on your particular device.
   - The example also implements handler for itemtaphold event which triggers when user presses anywhere on a disclosure row for more than one second as an alternative (refer to onExample2ListItemTaphold() function within controller.
   - On Disclose slides in detail panel showing all data columns for selected row
   - Disclose only implemented for detail (leaf) rows for this particular example.
   - Implements pull-refresh which sends “pullrefresh” event to controller to re-load data

b. Program Flow
   - Controller: ProjectController.js
   - View Config: Main.js => ProjectContainer.js => itemId = ‘example2’
   - Data:
     i. TouchTreeGrid.store.ProjectsStore (storeId = projectsStore)
     ii. Model: TouchTreeGrid.model.Projects
     iii. TreeStore (./data/treegrid.json). Data loaded via CommonController within onMainTabpanelActiveItemChange() method

```
55 if (newcont === 'projecttab'){
56     // Check store for data and load if empty (only)
57     numNodes = grid.getStore().getData().length;
58     if (numNodes === 0) {projectController.loadExample2Store(gridcont);}
59 }
```

projectController.loadExample2Store => commonController. loadStore => commonController.postLoadProcess =>

```
else if ((gridListItemId ==='example2list') ||
(gridListItemId ==='example2Blist') ||
(gridListItemId ==='example2Clist')) {
    if (gridListItemId ==='example2list') {
        Ext.Msg.alert('Custom Expand levels!');
    }

    projectController.getProjectContainer().down('#example2list').up('touchtreegrid').doRefreshList();
    projectController.getProjectContainer().down('#example2Blist').up('touchtreegrid').doRefreshList();
    projectController.getProjectContainer().down('#example2Clist').up('touchtreegrid').doRefreshList();

    // workaround to get Touch 2.2 pullrefresh plugin to auto-snapBack
    scroller.scrollTo(0,1);
}
```

- Events:

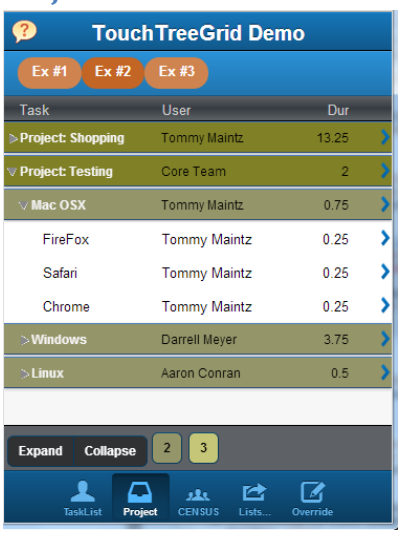
```
control: {
    "list#example2list": {
        disclose: 'onExample2ListDisclose',
        itemtaphold: 'onExample2ListItemTaphold'
    },
"container#example2": {
    pullrefresh: 'onExample2ListPullrefresh'
}
```

c. Notable Configs
   - categDepthColors = true
   - onItemDisclosure = true
   - disclosureProperty = 'leaf' (disclosure for leafs, but not categories)
   - categDepthColorsArr = ['#808127', '#949569', '#C5C678']
   - applyDefaultCollapseLevel = false (expand levels defined in store)
   - listPlugins = see code

d. CSS CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
/* Remove -or- rename this to use default Sencha disclosure icon if concerns with sensitivity of
smaller iphone style icon */
 .x-touchtreegrid-list-example2 .x-list-disclosure{
        width: 12px;
        height: 1.5em;
        margin: .4em 0 0 .5em;
        -webkit-mask: none;
        -webkit-mask-box-image:  (see treegriddemo.css…)
}
.x-touchtreegrid-list-example2 .x-list-disclosure:before{
/* Disable Touch 2.2 arrow picto for disclosure icon */
        content: '';
        font-family: '';
}
```

Project => Ex#2



a. Demonstrates
   a. Variation of Project #1 where itemHeight=32 and variableHeights=false to tighten space between rows.
   b. onDisclosure for leaf and category rows

b. Program Flow
   a. Controller: ProjectController.js
   b. View Config: Main.js => ProjectContainer.js => itemId = 'example2B'
      i. TouchTreeGrid.store.ProjectsStore (storeId = projectsStore)
      ii. Model: TouchTreeGrid.model.Projects
      iii. TreeStore. Data loaded via CommonController within onMainTabpanelActiveItemChange() method

```
55 if (newcont === 'projecttab'){
56     // Check store for data and load if empty (only)
57     numNodes = grid.getStore().getData().length;
58     if (numNodes === 0) {projectController.loadExample2Store(gridcont);}
59 }
```

   c. Data:
      i. TouchTreeGrid.store.ProjectsStore (storeId = projectsStore)
      ii. Model: TouchTreeGrid.model.Projects
      iii. TreeStore (./data/treegrid.json). Data loaded via CommonController within onMainTabpanelActiveItemChange() method

```
55 if (newcont === 'projecttab'){
56     // Check store for data and load if empty (only)
57     numNodes = grid.getStore().getData().length;
58     if (numNodes === 0) {projectController.loadExample2Store(gridcont);}
59 }
```

projectController.loadExample2Store => commonController. loadStore => commonController.postLoadProcess =>

```
else if ((gridListItemId ==='example2list') ||
(gridListItemId ==='example2Blist') ||
(gridListItemId ==='example2Clist')) {
    if (gridListItemId ==='example2list') {
        Ext.Msg.alert('Custom Expand levels!');
    }

    projectController.getProjectContainer().down('#example2list').up('touchtreegrid').doRefreshList();
    projectController.getProjectContainer().down('#example2Blist').up('touchtreegrid').doRefreshList();
    projectController.getProjectContainer().down('#example2Clist').up('touchtreegrid').doRefreshList();

    // workaround to get Touch 2.2 pullrefresh plugin to auto-snapBack
    scroller.scrollTo(0,1);
}
```

d. Events:

```
"list#example2Blist": {
    disclose: 'onExample2BListDisclose'
},
```

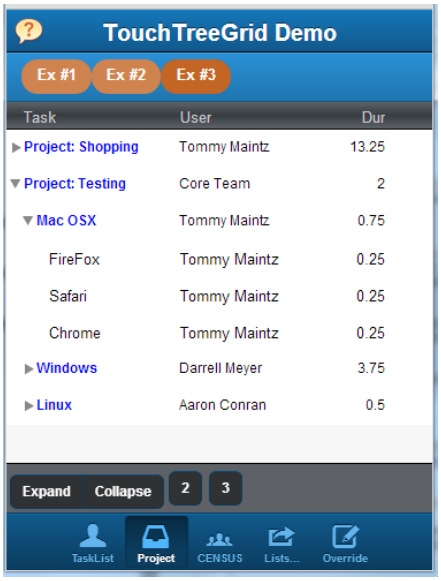
c. Notable Configs

- categDepthColors = true
- onItemDisclosure = true
- categDepthColorsArr = ['#808127', '#949569', '#C5C678']
- applyDefaultCollapseLevel = false (expand levels defined in store)

d. CSS CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
.x-touchtreegrid-list-example2B .x-list-disclosure{
        margin-top: .1em;
}
.x-touchtreegrid-list-example2B .x-list-normal .x-list-item .x-dock-horizontal,
.x-touchtreegrid-list-example2B .x-list-normal .x-list-item.x-list-item-tpl {
   border: none;
}
.x-touchtreegrid-list-example2B .touchtreegrid-list-content{ /* Detail (leaf) rows  */
    padding:.6em 0 0 0;
}
```

Project => Ex#3



a. Demonstrates

- Variation of Project #2 where row shading is removed.
- Disclosure icons removed from all rows
- Expand-to buttons not colored to match category rows

b. Program Flow

- Controller: ProjectController.js
- View Config: Main.js => ProjectContainer.js => itemId = 'example2C'
- Data:
  i. TouchTreeGrid.store.Task3Store (storeId = task3Store)
  ii. Model: TouchTreeGrid.model.Task
  iii. TreeStore (./data/treegrid.json). Data loaded via CommonController within onMainTabpanelActiveItemChange() method

```
55 if (newcont === 'projecttab'){
56     // Check store for data and load if empty (only)
57     numNodes = grid.getStore().getData().length;
58     if (numNodes === 0) {projectController.loadExample2Store(gridcont);}
59 }
```

projectController.loadExample2Store => commonController. loadStore => commonController.postLoadProcess =>

```
else if ((gridListItemId ==='example2list') ||
(gridListItemId ==='example2Blist') ||
(gridListItemId ==='example2Clist')) {
    if (gridListItemId ==='example2list') {
        Ext.Msg.alert('Custom Expand levels!');
    }

    projectController.getProjectContainer().down('#example2list').up('touchtreegrid').doRefreshList();
    projectController.getProjectContainer().down('#example2Blist').up('touchtreegrid').doRefreshList();
    projectController.getProjectContainer().down('#example2Clist').up('touchtreegrid').doRefreshList();

    // workaround to get Touch 2.2 pullrefresh plugin to auto-snapBack
    scroller.scrollTo(0,1);
}
```

- Events: (none)

c. Notable Configs

i. categDepthColors = true

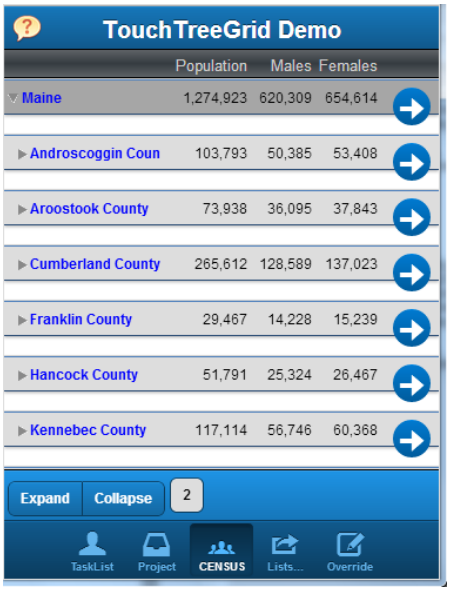
ii. categDepthColorsArr = [‘white’, ‘white’, ‘white’]

iii. categDepthColorButtons = false

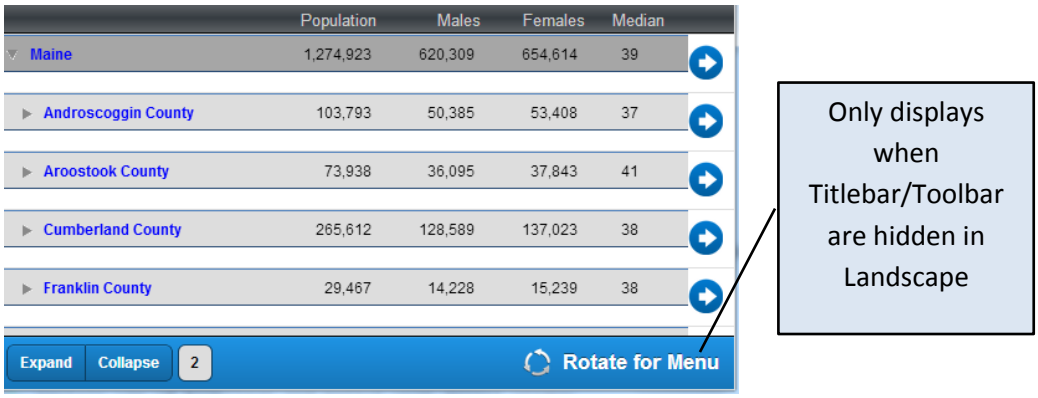
d. CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
.x-touchtreegrid-list-example2C .touchtreegrid-list-categ {
  border-top: none;
  border-bottom: none;
  -webkit-box-shadow: none;
}
.x-touchtreegrid-list-example2C .x-list-normal .x-list-item .x-dock-horizontal,
.x-touchtreegrid-list-example2C .x-list-normal .x-list-item.x-list-item-tpl {
   border: none;
}
.x-touchtreegrid-list-example2C .touchtreegrid-list-content{ /* Detail (leaf) rows  */
    padding:.6em 0 0 0;
}
```

Census Grid



a. Demonstrates
   i. ~600 total row grid example with 28 data columns per row (no performance issues)
   ii. Uses TouchTreeGrid default color schemes for category rows and collapse-to buttons
   iii. Loads data within controller via JSON tree store
   iv. Initially collapses to depth 2.
   v. Disclose implemented for category and detail rows to view all 28 columns in custom formatted grid. Grid layout created using Architect layouts and onCensusMaineListDisclose() function within controller defines column textfield details.
   vi. Grid and detail window utilize formatNumbers() function included with TouchTreeGrid to format numbers with commas.
   vii. Different COLUMN configurations for Phone-Portrait, Phone-Landscape, Tablet-Portrait, Tablet-Landscape. Refer to loadColumnsCensusMaine() function in CensusController.js for example on how to specify different column configurations. Test using webkit-enabled browser by manually resizing window:
      i. http://localhost/TouchTreeGrid/TouchTreeGrid_Advanced/app.html?deviceType=**Phone**
      ii. http://localhost/TouchTreeGrid/TouchTreeGrid_Advanced/app.html?deviceType=**Tablet**
   viii. Also note custom implementation in Phone-Landscape mode where Titlebar and TabBars are hidden allowing more display (rotate to Portrait to restore):



Note: if app is launched in phone landscape mode the menu will not be hidden until you rotate to portrait, then back to landscape.

b. Program Flow
   - Controller: CensusController.js
   - View Config: Main.js => itemId = 'censusmaine'
   - Data:
     i. TouchTreeGrid.store.CensusMaine2000 (storeId = censusmaine2000store)
     ii. Model: TouchTreeGrid.model.Census
     iii. TreeStore (./data/censusmaine2000TREE.json)
     iv. Data loaded via CommonController within onMainTabpanelActiveItemChange() method

```
if (newcont === 'censusmainecontainer') {

    // Check store for data and load if empty (only)
    numNodes = grid.getStore().getData().length;
    if (numNodes === 0) {censusController.loadCensusMaine2000Store();}
}
```

censusController. loadCensusMaine2000Store=> commonController. loadStore => commonController.postLoadProcess =>

```
if (gridListItemId === 'censusmainelist') {
    // Collapse nodes to defined level
    var depth = gridcont.getDefaultCollapseLevel();
    if (depth !== 99) {gridcont.doExpandDepth(depth);}

    censusController.loadColumnsCensusMaine(); // also refreshes list
}
```

     v. censusController.loadColumnsCensusMaine() loads columns[] array for list based on phone vs. tablet and portrait vs. landscape

- Events:

```
control: {
    "list#censusmainelist": {
        disclose: 'onCensusMaineListDisclose'
    },
    "button#censusdetailbackbtn": {
        tap: 'onCensusDetailBackButtonTap'
    }
}
```

c. Notable Configs
   i. categIndentPct = '2' (could create 2 different linked instances, tablets 2% and phones 4% are practical)
   ii. onItemDisclosure = true
   iii. categDepthColors = true
   iv. defaultCollapseLevel = 2

d. CSS – uses default CSS defined in TouchTreeGrid.css

Lists => Basic (DOW History - 2012 Closing Prices data)

a. Demonstrates
   i. Standard Grid example using non-TreeStore.
   ii. Row selection with required CSS to support
   iii. Expand/Collapse footer toolbar omitted as not applicable
   iv. Column sorting with custom column shading
   v. Zebra striping supported by following CSS defined in treegriddemo.css:

```
/******************* DOW2012 GRID OVERRIDES ***********************/
.x-touchtreegrid-list-dow2012 .touchtreegrid-simplelist-cell{ /* simple list cells  */
    background-color: transparent;  /* needed for alternate shading */
}

.x-touchtreegrid-list-dow2012 .x-touchtreegrid-item:nth-child(even) {  /* for alternate shading */
    background-color: #f5f5f5 !important;
}
```

   vi. Custom renderer functions easily supported:
      i. Format date as mm/dd/yyyy
      ii. Comma formatted closing price
      iii. Change and Chg% rendered as red for negative and green for positive

b. Program Flow
   i. Controller: ListsController.js
   ii. View Config: Main.js => ListsContainer.js => itemId = 'dow2012'
   iii. Data:
      i. TouchTreeGrid.store.Dow2012 (storeId = dow2012store)
      ii. Model: TouchTreeGrid.model.Dow2012
      iii. TreeStore (./data/dow2012.json)

1. commonController. onMainTabpanelActiveItemChange()

```
if (newcont === 'listscontainer'){
    numRecords = grid.getStore().getData().length;
    if (numRecords === 0) {
        Ext.Viewport.setMasked({
            xtype: 'loadmask',
            message: 'Loading Basic...'
        });
        grid.getStore().load();
        gridcont.doRefreshList();
        Ext.Viewport.setMasked(false);
    }
}
```

2.

iv. Events: (none defined)

c. Notable Configs
   i. simpleList = true
   ii. columnSorting = true
   iii. disableSelection = false
   iv. header : {…, minHeight: '2.6em', …} (increase height to improve touch for sorting)
   v. listPlugin : {xclass: 'Ext.plugin.PullRefresh'}
   vi. refer to columns[] array for:
      i. style vs. styleSorted definitions by column
      ii. sorted=true

d. CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
.x-touchtreegrid-list-dow2012 .touchtreegrid-simplelist-cell{ /* simple list cells  */
    background-color: transparent;  /* needed for alternate shading */
    padding-top: .5em !important;
    line-height: 2.6em;
}

.x-touchtreegrid-list-dow2012 .x-touchtreegrid-item:nth-child(even) {  /* for alternate shading
*/
    background-color: #f5f5f5 !important;
}

/** Begin DOW2012 CSS required to support row selection for simplelist rows **/
.x-touchtreegrid-list-dow2012 .touchtreegrid-item-selected,
.x-touchtreegrid-list-dow2012 .touchtreegrid-item-selected.x-touchtreegrid-item:nth-child(even) {
/* Notice no space b/t last 2 selectors .. not sure why this is necessary but it works for shaded
rows*/
    background-color: #006bb6 !important;  /* color for selected row */
}

.x-touchtreegrid-list-dow2012 .touchtreegrid-item-selected .touchtreegrid-simplelist-cell{
    background-color: #006bb6 !important;  /* cells need to match row color */
    color: white !important;  /* Change selected text color to white */
}

.x-touchtreegrid-list-dow2012 .touchtreegrid-item-pressed,
.x-touchtreegrid-list-dow2012 .touchtreegrid-item-pressed.x-touchtreegrid-item:nth-child(even),
.x-touchtreegrid-list-dow2012 .touchtreegrid-item-pressed .touchtreegrid-simplelist-cell {
    background-color: #b6e1ff !important;  /* color for selected row */
}

/* Below would control selected color of Disclosure Icon if onDisclosure=true */
```

```
.x-touchtreegrid-list-dow2012 .touchtreegrid-item-selected .x-list-disclosure{
    background-color: white;
}
.x-touchtreegrid-list-dow2012 .touchtreegrid-item-selected .x-list-disclosure:before{
    color: #006bb6;
}
/** End DOW2012 CSS required to support row selection for simplelist rows **/
```

Lists => Grouper (DOW History Example with Grouper feature)

a. Demonstrates
   i. Same as “Basic” DOW example but with Grouper feature.
   ii. Grouper defined by custom field provides min/max Closing prices for each month
   iii. Columns sort within each group when column header is tapped (as opposed to sorting across all data)
   iv. “grouped: true” must be defined within additionalListConfigs config:

```
simpleList: true,
columnSorting: true,
additionalListConfigs: {
    grouped: true
},
cls: [
    'x-touchtreegrid-list',
    'x-touchtreegrid-list-dow2012-grouper'
],
itemId: 'dow2012grouper'
```

   v. Store must define grouper “property” and “sortProperty” and initial sorters “direction” and “property” (if columnSorting: true):

```
Ext.define('TouchTreeGrid.store.Dow2012grouper', {
    extend: 'Ext.data.Store',

    requires: [
        'TouchTreeGrid.model.Dow2012'
    ],

    config: {
        model: 'TouchTreeGrid.model.Dow2012',
        storeId: 'Dow2012grouper',
        proxy: {
            type: 'ajax',
            url: './data/dow2012grouper.json',
            reader: {
                type: 'json',
                rootProperty: 'datalist'
            }
        },
        grouper: {
            direction: 'DESC',
            property: 'grouper',
            sortProperty: 'YearMonth'
        },
        sorters: {
            direction: 'DESC',
            property: 'CloseDate'
        }
    }
});
```

b. Program Flow
    a. Controller: ListsController.js
    b. View Config: Main.js => ListsContainer.js => itemId = 'dow2012grouper'
    c. Data:
        i. TouchTreeGrid.store.Dow2012grouper (storeId = Dow2012grouper)
        ii. Model: TouchTreeGrid.model.Dow2012
        iii. TreeStore (./data/dow2012grouper.json)
            1. listsController. onListsTabpanelActiveItemChange()

```
if (newcont === 'dow2012grouperCont'){
    numRecords = grid.getStore().getData().length;
    if (numRecords === 0) {
        Ext.Viewport.setMasked({
            xtype: 'loadmask',
            message: 'Loading Grouper ...'
        });
        grid.getStore().load();
        gridcont.doRefreshList();
        Ext.Viewport.setMasked(false);
    }
}
```

            2.

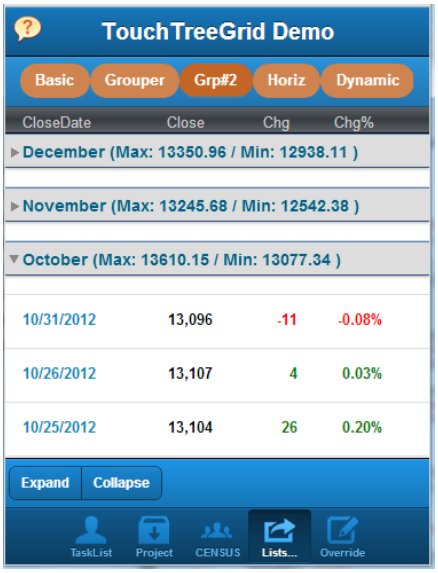
    d. Events:

```
control: {
    "tabpanel#liststabpanel": {
        activeitemchange: 'onListsTabpanelActiveItemChange'
    },
```

c. Notable Configs
    a. additionalListConfigs : {grouped : true}
    b. simpleList = true
    c. columnSorting = true
    d. disableSelection = true

d. CSS – uses default CSS defined in TouchTreeGrid.css

Lists => Grouper#2 (TreeGrid serving 'grouper' feature with expand/collapse)



a. Demonstrates
   i. Uses TreeGrid for 'grouper' type feature but supports expand/collapse
   ii. Leverages optional categColumns[] array to define column widths and styling for category row independent of leaf rows (which are defined in columns[] array)
   iii. Generates TreeStore from flat file containing ID and PARENT_ID columns
   iv. Demonstrates use of custom customExpCollapseEvent config to improve performance over Sencha default Node expand/collapse methods. Essentially we are rebuilding the TreeStore using the provided TreeStore JavaScript generation methods and reloading the store based on the desired collapse-to level.
   v. Note: Currently implemented in this example when running under Touch 2.1 framework until Sencha 2.2 bug can be isolated, reported and resolved (or workaround developed). I noticed in 2.2 framework when tabbing back and forth between Grouper and Grouper#2 examples, then clicking to expand Grouper#2 category the display is altered such that leafs are displayed as repetitive category rows. Debugging I see that Sencha is adding multiple stores to the same Node Interface object which is causing the display issue. This does not occur under 2.1 framework. However, I have implemented this feature in other 2.2 examples that don't exhibit the same problem. Seems if you mix this new feature with other lists in the same tab panel you will have a problem. If you isolate your implementation using this new feature under 2.2 it works just fine. More to come on this.

b. Program Flow
   i. Controller: ListsController.js
   ii. View Config: Main.js => ListsContainer.js => itemId = 'dow2012grouper2'
   iii. Data:
      i. TouchTreeGrid.store.dow2012TreeStore (storeId = dow2012treestore)
      ii. Model: TouchTreeGrid.model. Dow2012
      iii. TreeStore (./data/dow2012categ.json)

1. Here we are demonstrating provided tree generation methods which create TreeStore from flat file containing **ID** and **PARENT_ID** columns to link children to parents:

{"success": true, "datalist":[
{ "grouper":"December (Max: 12294 \/ Min: 11766.26)",
"Open":null,...."ID":"201112","PARENT_ID":null}
,{"CloseDate":"1\/10\/2012","Open":"12394.51",...."ID":"44","PARENT_ID":"201201"}
,{"CloseDate":"1\/11\/2012","Open":"12459.52",...."ID":"43","PARENT_ID":"201201"}
etc...

2. listsController. onListsTabpanelActiveItemChange()

```
if (newcont === 'dow2012grouper2Cont'){
    numRecords = grid.getStore().getData().length;
    if (numRecords === 0) {
        Ext.Viewport.setMasked({
            xtype: 'loadmask',
            message: 'Loading Grouper#2 ...'
        });
        this.loadDow2012Grouper2Store(gridcont, grid);
    }
}
```

3. listsController. loadDow2012Grouper2Store() => commonController.loadStore() => commonController.postLoadProcess

```
else if (gridListItemId === 'dow2012grouper2list') {

    // For each field define level "up to which" values will be included on category rows.
    // '0' means highest root row will include non-nullable values (or as defined in flat file) for this field.
    // '1' means level 1 categories will include data (if defined)
    // '2' means data will only be included for up to level 2.  Level 1 and root will not show data for this column.
    // etc...
    fldListArr = [['CloseDate', 1], ['Open', 1], ['High', 1], ['Low', 1], ['Close', 1],
    ['Volume', 1], ['AdjClose', 1], ['Chg', 1], ['ChgPct', 1], ['YearMonth', 0],
    ['grouper', 0]];

    collapseLvl = (Ext.isEmpty(gridcont.collapseLevel) ? 1 : gridcont.collapseLevel);
    // Refer to expCollapse() method where collapseLevel could be updated for manual expand processing

    commonController.loadTree(collapseLvl, griddata.datalist, fldListArr, gridcont, null); // collapse on initial load

    // Sort by YearMonth, then CloseDate
    gridlist.getStore().sort([{property: 'YearMonth', direction: 'DESC'},
    {property: 'CloseDate', direction: 'DESC'}]);

    Ext.Viewport.setMasked(false);
}
```

4. commonController.loadTree() => commonController.getTree() => commonController. createTreeStructure() {recursively generates child nodes}

iv. Events:

```
control: {
    "tabpanel#liststabpanel": {
        activeitemchange: 'onListsTabpanelActiveItemChange'
    },
    "container#dow2012grouper2": {
        expCollapse: 'onDow2012grouper2ExpCollapse'
    },
```

c. Notable Configs
   i. simpleList = false
   ii. categIndentPct = '0'
   iii. customExpCollapseEvent = 'expCollapse'
   iv. applyDefaultCollapseLevel = false
   v. categColumns: [{

```
    dataIndex: 'grouper',
    width: '90%',
    categStyle: 'text-align: left; color: #006083; font-weight: bolder; font-size: 1.2em;'
  }
```

d. CSS (defaults defined in TouchTreeGrid.css)

Lists => Horiz (DOW History Example with Horizontal Scrolling)

a. Demonstrates
   i. Horizontal scrolling simpleList grid
   ii. Custom header height, style and wrapping content
   iii. Column sorting with customized shading
   iv. Traps cell tap on CloseDate for controller processing.
   v. Vertical lines defined via styling
   vi. Column widths defined in pixels or em's instead of percent
   vii. Total grid width specified in pixels or em'
   viii. Parent Container must have layout = 'hbox' and horizontal scrolling:

```
xtype: 'container',
title: 'Horiz',
itemId: 'dow2012HorizCont',
layout: {
    type: 'hbox'
},
scrollable: {
    direction: 'horizontal',
    directionLock: true
},
items: [
    {
        xtype: 'touchtreegrid',
        store: 'Dow2012Horiz',
        columns: [
            {
```

ix. Vertical lines defined via border-right in Columns array. Example:

```
items: [
    {
        xtype: 'touchtreegrid',
        store: 'Dow2012Horiz',
        columns: [
            {
                header: 'CloseDate',
                dataIndex: 'CloseDate',
                width: '6em',
                style: 'text-align: left; font-weight: bold; color: #008abc; border-right: 1px solid #9b9b9b',
                headerStyle: 'text-align: left; color: #ccc;',
                renderer: 'Ext.Date.format(values.CloseDate, "n/j/Y")',
                sortable: true
            },
            {
                header: 'Open',
                dataIndex: 'Open',
                width: '4.375em',
                style: 'text-align: right;font-weight: normal;',
                headerStyle: 'text-align: right; padding-right: .5em !important; color: #ccc;',
                renderer: 'this.formatNumbers(values.Open, 0)',
                sortable: true
            },
            {
```

x. Also notice “width” defined in em’s in above example.

xi. Height = 1000 (dummy value, Sencha fails to scroll if percents or em’s used)

xii. Width and minWidth must be defined in linked instance for Horiz scrolliing:

```
    simpleList: true,
    columnSorting: true,
    cls: [
        'x-touchtreegrid-list',
        'x-touchtreegrid-list-dow2012Horiz'
    ],
    height: 1000,
    itemId: 'dow2012horiz',
    minWidth: '36em',
    width: '36em'
}
```

b. Program Flow
   a. Controller: ListsController.js
   b. View Config: Main.js => ListsContainer.js => itemId = ‘dow2012horiz’
   c. Data:
      i. TouchTreeGrid.store. Dow2012Horiz (storeId = Dow2012Horiz)
      ii. Model: TouchTreeGrid.model. Dow2012
      iii. TreeStore (./data/dow2012horiz.json)
         1. listsController. onListsTabpanelActiveItemChange()

```
if (newcont === 'dow2012HorizCont'){
    numRecords = grid.getStore().getData().length;
    if (numRecords === 0) {
        Ext.Viewport.setMasked({
            xtype: 'loadmask',
            message: 'Loading Horiz ...'
        });
        grid.getStore().load();
        gridcont.doRefreshList();
        Ext.Viewport.setMasked(false);
    }
}
```

         2.

d. Events:

```
control: {
    "tabpanel#liststabpanel": {
        activeitemchange: 'onListsTabpanelActiveItemChange'
    },
    "container#dow2012horiz": {
        leafItemTap: 'onHorizGridLeafItemTap'
    },
```

e. Code to trap click event on specific cell (CloseDate in this case)

```
onHorizGridLeafItemTap: function(me, list, index, target, record, e) {
127 // Example of how we can code to only act if CloseDate element touched
128 var myDate = Ext.Date.format(record.get('CloseDate'), "n/j/Y");
129 if (myDate !== e.target.innerText) {return;}
130
131 Ext.Msg.alert('You tapped: ' + myDate);
132 console.log('onHorizGridLeafItemTap record tapped:');
133 console.log(record);
134
```

c. Notable Configs
   a. simpleList = true
   b. columnSorting = true
   c. minWidth = '36em', width = '36em'
   d. header : { …, minHeight: '3.2em', …} to accommodate expanded header
   e. refer to columns[] array for style, header, styleSorted and categStyleSorted

d. CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
.x-touchtreegrid-list-dow2012Horiz .touchtreegrid-simplelist-cell{ /* simple list cells  */
   height: 47px !important;
}

.x-touchtreegrid-list-dow2012Horiz .x-grid-sort-desc,
.x-touchtreegrid-list-dow2012Horiz .x-grid-sort-asc {
    margin-top: .2em !important;
    background-position: left top;
}
.x-touchtreegrid-list-dow2012Horiz .touchtreegrid-header {  /* Titlebar with column headers  */
  /*  width:100%;  */
    padding:0 0 0 0;
    margin:.2em 0 0 0;
    color: black;
    font-size: 12px;
    background-color: #e5e5e5;
    background: none;
    font-weight: bold;
}
```

Lists => Dynamic (Same as Horiz but defined dynamically from Server)

a. Demonstrates
   - Same as Horiz example accept columns and fields are retrieved from single AJAX call and Store is created dynamically and loaded with data (from same call)

b. Program Flow
   - Controller: ListsController.js
   - View Config: Main.js => ListsContainer.js => itemId = 'dow2012Dynamic'
   - Data:
     i. TouchTreeGrid.store. DynamicStore (storeId = dynamicstore)
     Note: this is a dummy store reference to avoid console warnings when creating linked instance of view. It will be overridden when creating and loading store dynamically.
     ii. Model: no model used as fields will by dynamically defined when creating store
     iii. TreeStore (./data/dow2012dynamic.json)
     Contains 3 arrays of data:
        1. datalist[]
        2. columnsPhonePortrait[]
        3. fields[]
     iv. listsController. onListsTabpanelActiveItemChange()

```
if (newcont === 'dow2012DynamicCont'){
    // Reload each time pressed for dynamic grid example
    this.loadDow2012DynamicStore(gridcont, grid);

}
```

     v. listsController. loadDow2012DynamicStore() => commonController.loadDynamicStore()

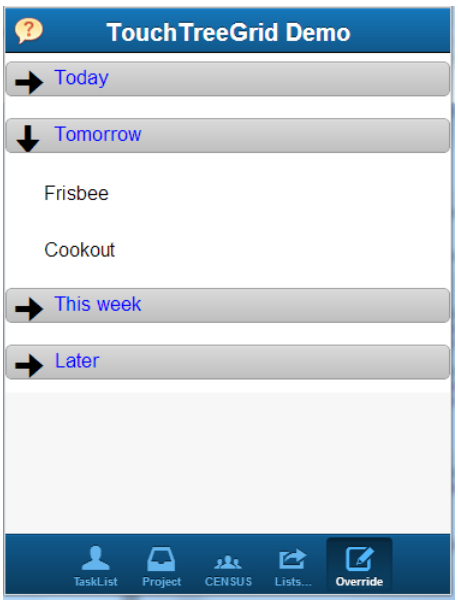
```
success: function(response) {
    var alldata = Ext.JSON.decode(response.responseText);
    var griddata             = alldata.datalist;
    var columnsPhonePortrait = alldata.columnsPhonePortrait;
    var fields               = alldata.fields;

    // NOTE:  Could load different device and orienation column configurations
    //        and apply based on device/orientation here .. and update again
    //        within onOrientationChange().  If you load multiple configurations
    //        It is suggested that you store each of the column arrays to gridcont
    //        component for easy retrieval when switching.
    //
    //        Ex:  gridcont.columnsPhonePortrait = columnsPhonePortrait;
    //             gridcont.columnsTabletLandscape = columnsTabletLandscape;
    //             etc...
    //
    //        Simply by calling doRefreshList() method after updating columns your
    //        grid will immediately reflect the new column configuration.
    //
    //        Could support customized user preferences in this same way.

    var gridListItemId = gridcont.getListItemId();
    var gridlist = gridcont.down('#'+gridListItemId);

    gridcont.setColumns(columnsPhonePortrait);

    // Note: we are defining fields directly within Store instead of creating Model
    var gridstore = Ext.create('Ext.data.Store', {fields: fields});

    gridlist.setStore(gridstore);

    if (!loadStoreInPostProcess) {
        gridstore.suspendEvents();
        var gridloaded = gridstore.add(griddata);
        gridstore.resumeEvents();
        if (loadmask) {Ext.Viewport.setMasked(false);}
    }|
```

c. Notable Configs
   - Store = ‘dynamicstore’ (refer above regarding dummy reference)
   - cls: [
     'x-touchtreegrid-list',
     'x-touchtreegrid-list-dow2012Horiz',
     'x-touchtreegrid-list-dow2012Dynamic'
     ]

d. CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

   Uses same CSS as dow2012Horiz. Could be customized further using **'x-touchtreegrid-list-dow2012Dynamic' selector**

Override Grid



a. Demonstrates
   - How automated grid generation can be overridden with use of following configs where developer would specify own TPL for detail, category and header rows (column[] array is not used):
     i. contentItemTplOverride
     ii. categItemTplOverride
     iii. headerTplOverride

b. Program Flow
   - Controller: none
   - View Config: Main.js => itemId = 'overrideexample'
   - Data
     i. TouchTreeGrid.store.OverrideStore (storeId = overrideStore)
     ii. Model: TouchTreeGrid.model.Task
     iii. TreeStore (data defined directly in root property)
     iv. Autoload = true
   - Events: none

c. Notable Configs
   - includeFooter = false
   - includeHeader = false
   - categItemTplOverride

categItemTplOverride - object

```
<div style="display : inline-block; white-space : nowrap; width : 100%;font-size: 1.5em;">
<tpl if="this.isExpanded(values)">
<span class="x-button x-button-small ">
   <span class="x-button-icon arrow_down x-icon-mask"
      style="width: .75em; height: .75em; margin-right:0.4em;">
   </span>
<span style="color:blue; padding-left: 1.5em; padding-top: .2em; padding-bottom: .2em;">{text}</span>
</span>
<tpl else>
<span class="x-button x-button-small">
   <span class="x-button-icon arrow_right x-icon-mask"
      style="width: .75em; height: .75em; margin-right:0.4em;">
   </span>
   <span style="width: 100%; color:blue; padding-left: 1.5em; padding-top: .2em; padding-bottom: .2em;">{text}</span>
</span>
</tpl>
</div>
```

- contentItemTplOverride: ' {text} ',

d. CSS (defaults defined in TouchTreeGrid.css, customized by treegriddemo.css)

```
.x-touchtreegrid-list-override .touchtreegrid-list-categ {
  background-color: white !important;
  border: none;
  -webkit-box-shadow: none;
}
.x-touchtreegrid-list-override .x-list-normal .x-list-item.x-list-item-tpl{
   border: none;
}
```

Notes on Implementation

1. Setup if not using Architect
    a. Copy 'TouchTreeGrid.js' from ./TouchTreeGrid_Advanced/app/view/ into similar view directory for your project
    b. Include 'TouchTreeGrid' in list of 'views' in app.js or your controller
    c. Copy resource files as discussed below.
    d. I'm assuming you know what to do from here if not using Architect…

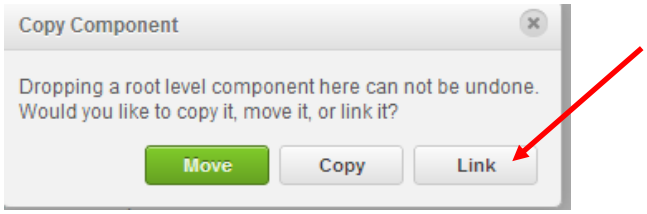
2. Setup if using Sencha Architect
    a. Import TouchTreeGrid.xdc into your toolbox (via right-click).
    Created using Architect Version: 2.2.2 Build: 991
    b. Drag this component on top of "Views" in your project inspector to create parent class TouchTreeGrid. This will add a view reference in your "Application" (app.js)
    c. Suggest copying 'resources' subdirectory from download as is into the same directory level of your project. Key files to include in your project:
        i. ./resources/css/TouchTreeGrid.css
    d. Add CSS Resource to 'Resources" section of your project inspector:
    Update url = './resources/css/TouchTreeGrid.css'
    e. Optionally add custom CSS Resource **after** this one:
        i. Update url ='./resources/css/treegriddemo.css
        ii. Definitions in subsequent stack will override prior definitions. When project is saved you will notice in APP.HTML that treegriddemo.css is loaded after TouchTreeGrid.css

APP.HTML

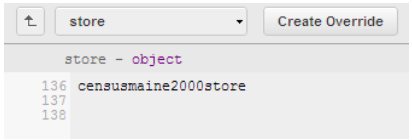
```
<!DOCTYPE html>

<!-- Auto Generated with Sencha Architect -->
<!-- Modifications to this file will be overwritten. -->
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>TouchTreeGrid_Advanced</title>
    <script src="http://cdn.sencha.com/touch/sencha-touch-2.2.1/sencha-touch-all-debug.js">
</script>
    <link rel="stylesheet" href="http://cdn.sencha.com/touch/sencha-touch-2.2.1/
resources/css/sencha-touch.css">
    <link rel="stylesheet" href="./resources/css/TouchTreeGrid.css">
    <link rel="stylesheet" href="./resources/css/treegriddemo.css">
    <script type="text/javascript" src="app.js"></script>
    <script type="text/javascript">
        if (!Ext.browser.is.WebKit) {
            alert("The current browser is unsupported.\n\nSupported browsers:\n" +
                "Google Chrome\n" +
                "Apple Safari\n" +
                "Mobile Safari (iOS)\n" +
                "Android Browser\n" +
                "BlackBerry Browser"
            );
        }
    </script>
```

```
</head>
<body></body>
</html>
```

f. Create Model and Store for this grid
   i. Store must be “Tree Store”.
   ii. Be sure to define storeId for the Store
g. Create Container to contain your grid
   i. Layout = ‘fit’ for standalone implementations
   ii. Layout = ‘card’ for tabpanel implementations
h. Drag the ‘TouchTreeGrid’ class on top of this container to make it a child component
   i. Be sure to Copy Component as a “**Link**”



i. Click on the added “linked” component under your container and start updating your configs. Minimum required definitions:
   i. **xtype** : ‘touchtreegrid’ – automatically created when you link the view
   ii. **itemId** (unique identifier for this grid)
   iii. **columns** - array defining columns for grid (refer to Appendix A)
   iv. **store** – This is storeId from store. Remove the {} when entering storeId in object edit window



   v. **listItemId** – provide if you want a unique reference to the generated grid list for reference within controller

   vi. **DO NOT** update **createAlias** until you are ready to make this linked instance an “Inline Class” (refer to Appendix C)

   vii. **NOTE**: Linked instances inherit all the configurations defined in TouchTreeGrid component. Architect configuration panel will show these parameters as you search for them and will include the pre-defined default values (which you can override). You only need to define overridden parameters when initially linking to TouchTreeGrid.

Refer to Appendix C as to how the number of configurations can automatically

change when upgrading the component.

j. Repeat this for all other views.
k. Save Often !

3. Notes for Tree Store
   a. Look at code for Task store examples to see what a tree store should look like in object format.
   b. Look at ./data/treegrid.json from examples to see what tree store should look like in JSON format when loading into a store
      i. This file was obtained from EXTJS download ./examples/tree/ directory
      ii. Good tool to validate your JSON: http://jsonlint.com/
      iii. I have found, and others have reported there is a bug with Sencha Store component loading JSON files directly from within the Store. Refer to loadStore() function within CommonController for workaround. This approach would be preferred anyway if you were also loading column arrays from server in same call. One requirement I found for this to work is to initialize root config within the Tree Store as follows: **root: {children: []}**

```
Ext.define('TouchTreeGrid.store.Task2', {
    extend: 'Ext.data.TreeStore',  <= Tree Store required !
    requires: ['TouchTreeGrid.model.Task2'],
    config: {
        autoLoad: true,
        model: 'TouchTreeGrid.model.Task2',
        storeId: 'Task2Store',  <= referenced in linked grid config
        root: {children: []}
    }
});
```

4. Test your application on actual devices
   a. If you didn't already know this, you can easily test your application on your own devices as you develop if you have flexible use of your company router or most certainly on your home router network as follows:
      i. Need local webserver running of course and all source code needs to be underneath webserver directory. For XAMPP Apache server this would be: c:/xampp/htdocs/
      ii. Get your local IP by running 'ipconfig' in your CMD window
      iii. From phone/tablet browser enter address similar to this (for provided examples assuming they were extracted as such): http://192.168.1.??/TouchTreeGrid/TouchTreeGrid_Advanced/app.html

Additional Functionality included in Advanced Example:

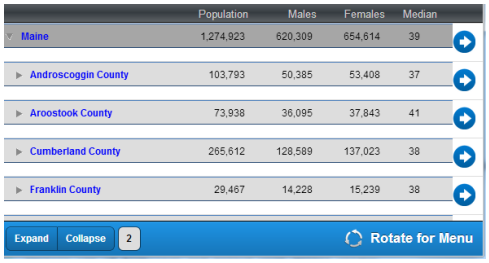
1. Refer to “Summary of Provided Examples” for how JSON data is loaded into TreeStore in varying ways for each of the examples.
2. Example for handling changes in device orientation

    a. “orientationchange” event for viewport is processed within CommonController

```
control: {
            "viewport": {
                orientationchange: 'onOrientationChange'
            }, etc...
```

    b. onOrientationChange() function calls censusController.loadColumnsCensusMaine () function which loads different column configurations for phone (portrait vs. landscape) and tablet (portrait vs. landscape). The number of grid columns dynamically change simply by rotating the device.

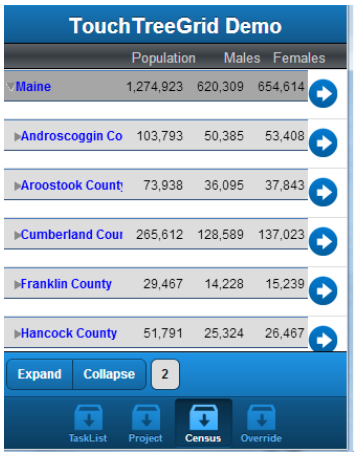
    c. onOrientationChange() function calls commonController.hideShowPanels() function to hide titlebar and bottom toolbar for phones (only) when changing from portrait to landscape orientation to allow more vertical display when navigating the grid.



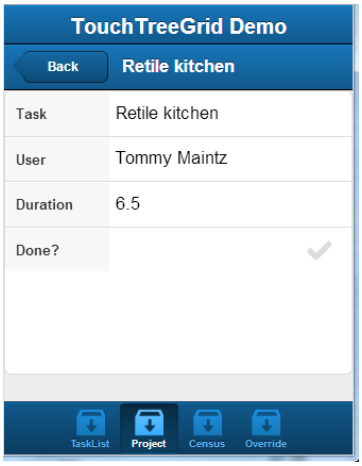
    d. “Rotate for Menu” text can be customized in hideShowPanels()

    e. Recycle icon can be customized by **landscapeIcon='./resources/images/Recycle.png'** config parameter for the linked TouchTreeGrid component.

f. Titlebar and Tabbars are unhidden when rotating back to portrait.



3. **Griddetailpanel.js** is an example of generic component that can be re-used to display selected row detail fields for multiple grids



a. TouchTreeGrid.view.griddetailpanel created using Architect and multiple instances of this view can be instantiated. This defines the layout and dummy fieldset which will be overwritten for each implementation. Suggest exporting to file (griddetailpanel.xdc) and importing into toolbox for project re-use

b. CommonController defines references to parent grid panel (main) and griddetailpanel:

```
refs: {
  main: {
       selector: 'main',
       xtype: 'main'
  },
  griddetailpanel: {
       autoCreate: true,
       selector: 'griddetailpanel',
       xtype: 'griddetailpanel'
  },
```

c. ProjectController and CommonController includes control definitions:

```
control: {
  "list#example2list": {  <= grid listItemId = 'example2list'
      disclose: 'onExample2ListDisclose',
      itemtaphold: 'onExample2ListItemTaphold'  <= also trapping taphold
  },
  "button#example2detailbackbtn": { <= back button to navigate back
      tap: 'onExample2GridDetailBackButtonTap'
  },
```

d. onExample2ListDisclose() and onExample2ListItemTaphold() functions both call onExample2ListDiscloseOrHold()

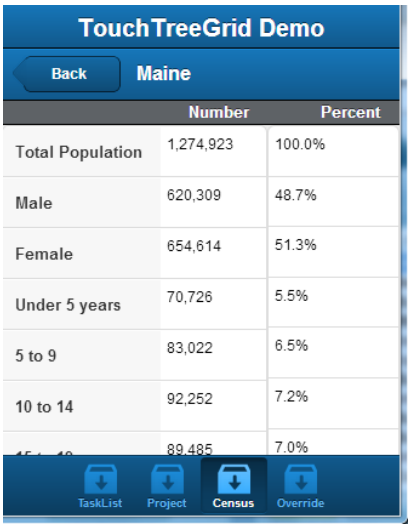
e. onExample2ListDiscloseOrHold () function instantiates "griddetailpanel" for this particular grid and updates fldSet to display all data fields for this grid.

```
onExample2ListDiscloseOrHold: function(record, target, index) {

// example2container is itemId of parent container to linked grid instance.  We will be
// swapping the grid instance with the detail panel.

var swapcont = this.getMain().down('#example2container');
if (swapcont)
{
    var newcont = this.getGriddetailpanel(<= create new instance of griddetailpanel
    {
        title : 'Example 2 Detail',
        id : 'example2detail', <= can assign unique ID (not used in this example)
        layout: {type: 'fit'},
        itemId: 'griddetailpanel'
    }
    );

    var gridItemId = swapcont.down('touchtreegrid').getItemId();
    newcont.swapcont = swapcont; <= storing reference to swapcont for generic back button
    newcont.gridItemId = gridItemId; <= storing reference to gridItemId

    if (newcont)
    {
        var newLabel = newcont.down('#griddetaillabel');
        newLabel.setHtml(record.get('task'));  <=update label with selected row name

        var fldSet = newcont.down('#griddetailfieldset');
        // Proceed to manually define fields to display all fields for selected row

        var result = fldSet.setConfig({
            items : [
            {label: 'Task', xtype: 'textfield', readOnly: true, value: record.data.task},
            {label: 'User', xtype: 'textfield', readOnly: true, value: record.data.user},
            {label: 'Duration', xtype: 'numberfield', readOnly: true,
                value: record.data.duration},
            {label: 'Done?', xtype: 'checkboxfield', disabledCls: null,
                checked: record.data.done}
        ]});

        swapcont.add(newcont); <= adding new griddetailpanel
        swapcont.setActiveItem(newcont); <= making it the active item for display
    }
}
```

f. onGridDetailBackButtonTap() function handles Back button press to navigate from griddetailpanel back to the grid

```
// Reusing the Back button for all Project Task examples by storing references when creating
detail panel (list disclose)
var swapcont = button.up('#griddetailpanel').swapcont; <= see onExample2ListDiscloseOrHold()
if (swapcont)
{
    gridItemId = button.up('#griddetailpanel').gridItemId; <= see onExample2ListDiscloseOrHold()
    var newcont = swapcont.down('#'+gridItemId);

    newcont.setShowAnimation({type :"slide", direction : "right"});
    swapcont.setActiveItem(newcont);
}
```

4. **Censusdetailpanel.js** is a custom implementation of **griddetailpanel** documented above



a. Architect is used to quickly create the layouts with dummy fieldsets for Number and for Percent columns.
b. Censusdetailpanel is then instantiated similar to griddetailpanel
c. CensusController defines the needed references and controls

```
refs: {
      censusdetailpanel: {
          autoCreate: true,
          selector: 'censusdetailpanel',
          xtype: 'censusdetailpanel'
      }
  }, etc…

  control: {
      "list#censusmainelist": {
          disclose: 'onCensusMaineListDisclose'
      },
      "button#censusdetailbackbtn": {
          tap: 'onCensusDetailBackButtonTap'
      }, etc...
  }
```

d. Fieldsets for Number and Percent are manually updated within onCensusMaineListDisclose() function.

e. Back button functionality handled within onCensusDetailBackButtonTap()

5. **onMainTabpanelActiveItemChange**() within CommonController contains logic to load Project and Census data only when user clicks on those tabs to speed up intial launch time.
   a. Data only loaded if Store is empty

6. **onTitlebarGridhelp**() function with CommonController traps touches on TitleBar and launches modal information window custom for each Grid
   a. GridHelpPanel view defines modal panel to display HTML
   b. HTML files for each grid stored in ./resources/html/ directory
   c. AJAX call made to load HTML file into panel

```
…see code
gridId = grid.getHelpHtml(); <= URL reference stored to helpHtml config
                                for each linked instance of TouchTreeGrid

Ext.Ajax.request({
    url: gridId,
    method: 'GET',
    callback: function(options, success, response) {

        var help = me.getGridHelpPanel();
        help.setHtml( response.responseText );
        help.showBy(image);
    }
});
```

7.

APPENDIX A – Columns Array Definitions

Column configurations are defined within a COLUMNS array for each linked grid instance:

- header - Text to include in column header.
  - Can embed HTML tags for formatting within the string
  - 'My Header' displays as 'My *Header*'
  - 'My Header' displays as 'My **Header**'
- dataIndex - data column from treestore. Upper/lower case must match model fields!
- width - CSS format width for this component (suggest % or em's to better support all device types)
- style - Additional CSS styling commands for detail rows
- styleSorted - style to apply when this column is sorted
- categStyle - Additional CSS styling commands for category rows (not applicable where simpleList = true)
- categStyleSorted – style to apply to catgory row when this column is sorted
- headerStyle - Additional CSS styling commands for column header row
- headerStyleSorted – style to apply to header row when this column is sorted
- renderers - refer to "Advanced Features" sections for discussion on use
- sortable - true to make this column sortable (requires columnSorting=true for linked instance of TouchTreeGrid).

Columns array example for Project list example:

```
columns: [
        {
            header: 'Task',
            dataIndex: 'task',
            width: '35%',
            style: 'text-align: left;',
            categStyle: 'font-weight: bold; text-align: left; color: blue;',
            headerStyle: 'text-align: left; color: #ccc;'
        },
        {
            header: 'User',
            dataIndex: 'user',
            width: '35%',
            style: 'text-align: left; padding-left: .5em;',
            categStyle: 'text-align: left;padding-left: .5em;',
            headerStyle: 'text-align: left; color: #ccc;padding-left: .5em;'
        },
        {
            header: 'Dur',
            dataIndex: 'duration',
            width: '15%',
            style: 'text-align: right;',
            categStyle: 'text-align: right;',
            headerStyle: 'text-align: right; color: #ccc;'
    ]
```

Refer to loadColumnsCensusMaine() function within censusController in Advanced example for additional number of column variations for Phone (portrait vs. landscape) and Tablet (portrait vs. landscape).

APPENDIX B – CSS Styling

Every project should include a copy of **TouchTreeGrid.css** which defines the default styling for TouchGreeGrid component. Any custom overrides should be included in a file loaded after TouchTreeGrid.css. **treegriddemo.css** contains examples of custom styling overrides.

Excellent reference tools:

- http://www.w3schools.com/cssref/default.asp (CSS3 properties reference)
- http://www.colorschemedesigner.com/ (provides compliment and triad colors to given color)
- http://www.color-hex.com/ (provides shades and tints of particular color which is useful for gradients)
- http://www.colorzilla.com/gradient-editor/ (provides tools to create gradient, generates styling for copy/paste into your CSS file)
- http://www.gimp.org/ (image manipulation program)

Review of the TouchTreeGrid CSS classes and how they are used by TouchTreeGrid.css:

- **.x-touchtreegrid-list** is the default class for the entire component. Each linked instance should at minimum keep this class and possibly include additional overrides. Example of secondary CSS class reference (Example2):

```
cls: [
        'x-touchtreegrid-list',
        'x-touchtreegrid-list-example2'
    ],
```

- Every entry in TouchTreeGrid.css is prefixed with .x-touchtreegrid-list. Every custom override would be written to treegriddemo.css (or similar) and prefixed with the custom class (i.e. .x-touchtreegrid-list-example2) instead of default class (.x-touchgreegrid-list), followed by the component specific class name. Example:

    - TouchTreeGrid.css contains this definition:
      .x-touchtreegrid-list **.x-touchtreegrid-item** {
        background-color: **white**;
      }

    - If treegriddemo.css contained this definition:
      .x-touchtreegrid-list-example2 .x-touchtreegrid-item {
        background-color: **red**;
      }

- Then for Example2 the background would be overridden to red.

- Default background color for grid list items:

```
.x-touchtreegrid-list .x-touchtreegrid-item {
   background-color: white;
}
```

- [Touch 2.1] treegriddemo.css contains this definition to hide or modify horizontal lines:
(both Project #2 and #3 examples have cls : 'x-touchtreegrid-list-example2B')

```
.x-touchtreegrid-list-example2B .x-list-normal .x-list-item .x-dock-horizontal {
  border: none;
}
```

- [Touch 2.2] treegriddemo.css contains this definition to hide or modify horizontal lines:
(both Project #2 and #3 examples have cls : 'x-touchtreegrid-list-example2B')

```
.x-touchtreegrid-list-example2B .x-list-normal .x-list-item .x-dock-horizontal ,
.x-touchtreegrid-list-example2B .x-list-normal .x-list-item.x-list-item-tpl{
  border: none;
}
```

- Default layout for category rows:

```
.x-touchtreegrid-list .touchtreegrid-list-categ {
 background-color: #ddd !important;
 background-image: none !important;
 color: black;
 border-top: 1px solid #4D80BC;
 border-bottom: 1px solid #2D4E76;
 font-weight: normal;
 min-height:2.6em !important;
 padding:0.6em 0 0 0 !important;
 width:100% !important;
 font-size: .7em !important;
 -webkit-box-shadow: 0px 0.1em 0.3em rgba(0, 0, 0, 0.3);
}
```

- Default layout for Detail rows

```
.x-touchtreegrid-list .touchtreegrid-list-content{
   color: black;
   background-color:white;
   width:100%;
   padding:1em 0 0 0;
```

```
    margin:0;
    font-size: .8em !important;
}
```

- Default layout for Header row

```
.x-touchtreegrid-list .touchtreegrid-header {
    width:100%;
    padding:0.4em 0 0 0;
    margin:0;
    color: #fff;
    height : 1.6em;
    font-size: .8em !important;
    see CSS file for actual gradient definitions as too large here..
}
```

- Default layout for Category row individual cells

```
.x-touchtreegrid-list .touchtreegrid-list-categ-cell{
    overflow      : hidden;
    text-overflow  : clip;
    white-space    : nowrap;
}
```

- Default layout for Header row individual cells

```
.x-touchtreegrid-list .touchtreegrid-header-cell {
    white-space: nowrap;
    overflow: hidden;
    text-overflow: clip;
}
```

- Default layout for Detail row individual “cells”

```
.x-touchtreegrid-list .touchtreegrid-list-content-cell{
    overflow      : hidden;
    text-overflow  : clip;
   white-space    : nowrap;
}
```

- Default for Simple List “cells”

```
.x-touchtreegrid-list .touchtreegrid-simplelist-cell{ /* simple list cells  */
    display       : inline-block;
    overflow      : hidden;
    text-overflow  : clip;
```

```
    white-space    : nowrap;
    color: black;
    background-color:white;
    padding:1em 0 0 0;
    margin:0;
    font-size: .8em !important;
}
```

- Handle colors for simple grid list item "selected" and "pressing" (selectors for Treegrid selection not provided as deemed not appropriate but handled similarly)

```
.x-touchtreegrid-list .touchtreegrid-item-selected,
.x-touchtreegrid-list .touchtreegrid-item-selected .touchtreegrid-simplelist-cell{
    background-color: #006bb6 !important;  /* Sencha 2.1 default color for selected */
    color: white !important;
}
.x-touchtreegrid-list .touchtreegrid-item-selected .x-list-disclosure {
  background: #fff none;  /* reverses color on disclosure icon for selected rows */
}
.x-touchtreegrid-list .touchtreegrid-item-pressed,
.x-touchtreegrid-list .touchtreegrid-item-pressed .touchtreegrid-simplelist-cell{
    background-color: #b6e1ff !important;  /* Sencha 2.1 default color for pressing */
    color: white !important;
}
```

- Default layout for Category row arrow

```
.x-touchtreegrid-list .touchtreegrid-details-img {   /* category row arrow */
  width: 18px;
  min-height: 18px;
  height: 100%;
  display: -moz-inline-box;
  vertical-align: top;
  display: inline-block;
  background-image: (see file for embedded sprite … contains both arrow left and arrow down)
  background-repeat: no-repeat;
  vertical-align: top;
  cursor: pointer;
  border: 0;
  border-image: initial;
}
```

Associated references to sprite:

```
.x-touchtreegrid-list .touchtreegrid-details-img-open {
  /* 2nd arrow sprite pointing down for open category */
  background-position: 0 -18px;
```

```
}

.x-touchtreegrid-list .touchtreegrid-details-img-close {
  /* 1st arrow sprite pointing right for closed category */
  background-position: 0 0;
}
```

- Sort up/down arrows defined in TouchTreeGrid.css
  - .x-touchtreegrid-list .x-grid-sort-desc {…}
  - .x-touchtreegrid-list .x-grid-sort-asc {…}
  - background-position: right top;
    - This can be change to locate to different areas within column header cell
    - I found that best appearance achieved by adding "padding-right: .5em !important;" to headerStyle in column[].
- Defaults for footer toolbar (TouchTreeGrid uses Sencha defaults)

```
.x-touchtreegrid-list .touchtreegrid-footer {  <= no changes to Sencha defaults
}
```

Example2 overrides this:

```
.x-touchtreegrid-list-example2 .touchtreegrid-footer {
   background: #5e6266;  <= gray background instead of Sencha default
}
```

- Default layout for expand/collapse buttons

```
.x-touchtreegrid-list .touchtreegrid-expand-collapse-buttons {
   margin-top: .4em;
}
```

- Default layout for label reading "Rotate for Menu" when phone is rotated from portrait to landscape

```
.x-touchtreegrid-list .touchtreegrid-landscape-label {
   margin: .7em .5em 0 0;
}
```

- Default layout for Recycle icon displayed when phone is rotated from portrait to landscape

```
.x-touchtreegrid-list .touchtreegrid-landscape-icon {
   height: 1.5em !important;
   margin: .5em .5em 0 0;
   width: 1.5em !important;
}
```

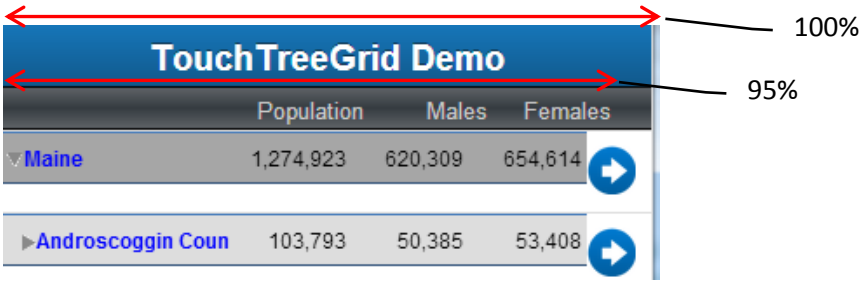
- Custom IPhone style Disclose Icon (treegriddemo.css)

.x-touchtreegrid-list-example2 .x-list-disclosure{
width: 12px;
height: 1.5em;
margin: .4em 0 0 .5em;
-webkit-mask: none;
-webkit-mask-box-image: (see file for embedded image)
}
Note: you may or may not want to use this icon depending on your experience with sensitivity when touching icon on your particular device. Project #1 example also implements itemtaphold event as secondary means to react to user's attempt to navigate.

- [Touch 2.2+] Added .x-list-disclosure:before selector to treegriddemo.css to disable arrow picto from overlaying Example #2 I-Phone style icon:

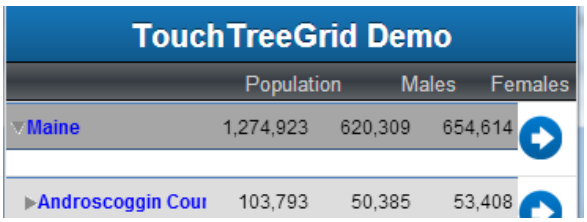
```
.x-touchtreegrid-list-example2 .x-list-disclosure:before{
  content: '';
  font-family: '';
 }
```

- [Following not applicable for Touch 2.2+ as disclosure icons no longer squeeze list item row] Define spacer width when using disclose icons such that header columns line up with category/detail columns:

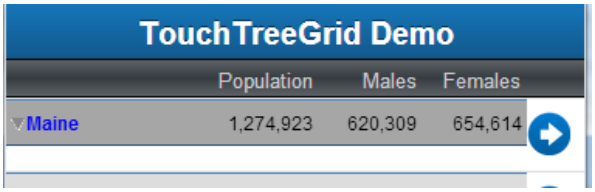


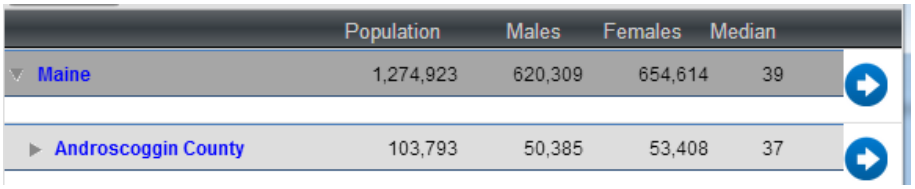

```
.touchtreegrid-disclose-spacer {
  max-width: 95%;
}
```

100% example reveals the problem with no spacer:



90% example looks great on phone portrait, but slightly off for phone landscape:





This is just an approximation and will look slightly different on different devices and orientation, but it is better than no attempt to adjust the display (will spend more time on this later). Default was left at 95% for phone landscape and tablet configs. Note we implemented a similar change to Mitchell Simeon's Ext.ux.touch.grid when using disclosure icons.

APPENDIX C – Upgrading TouchTreeGrid component

The following steps can be used to upgrade to a new version of the TouchTreeGrid component within Architect:

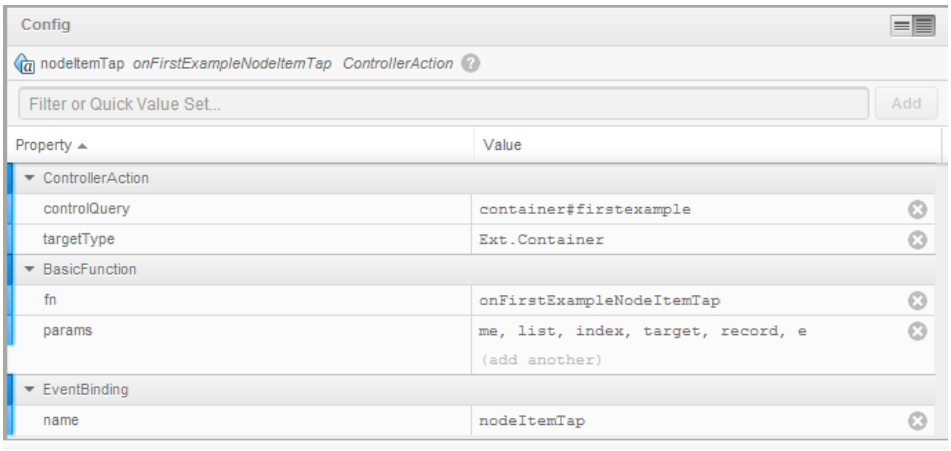
1. **Backup your project!**

2. Update **createAlias='touchtreegrid'** on all linked instances of TouchTreeGrid within Architect's configuration panel (Note: this is an Architect-only setting … you will not immediately see a code change). You should not update createAlias until you are ready to do step #3 to updating linked instances as "Inline Class" as you could observe issues running your code (i.e. perform steps #2 and #3 in unison)

3. Right-click each linked instance and convert to "Inline Class".
    a. This removes the link and creates a copy of all previously defaulted configurations with the default values. If you want to keep you code size small you can go back and delete the added ones, but this is not necessary.
    b. xtype = 'touchtreegrid' should remain after removing
        i. If it doesn't it's because you didn't update createAlias above.
    c. Be sure to add any new configurations supported by the upgraded component if you do not want to accept the default values
    d. Prior upgrades that are already Inline do not need to be modified unless you need to add newly supported configurations.

4. **Repeat step 2 and 3 for ALL linked instances (else they will be deleted !!)**

5. Make sure there are no references to 'touchtreegrid' in controller config->"control" section within Architect's config panel. I specifically updated onFirstExampleLeafItemTap() and onFirstExampleNodeItemTap() functions as follows:
    a. "**targetType**" should be defined as "Ext.Container" instead of "TouchTreeGrid".
    b. "**control query**" should read "container#firstexample" instead of "touchtreegrid#firstexample"



c. If you have references to **touchtreegrid**, any custom parameters **will be lost** when you load the new component !

6. **Save the Project !**

7. Once you are sure you converted all linked instances, delete the TouchTreeGrid component from you Project Inspector.

8. Right-click in Toolbox and import new version of TouchTreeGrid.xdc (be sure to override default name of 'container' to 'touchtreegrid')

9. Drag the newly imported component ontop of "View" within Project Inspector.

10. Copy/overlay new version of **TouchTreeGrid.css** if updated.

11. **Save the Project !**

12. Confirm that "TouchTreeGrid" is referenced as a view in the Application component within Project Inspector.
    a. Exception: You may be referencing this within specific controllers and would need to manually add the reference within these controllers as appropriate.

APPENDIX D – TouchTreeGrid Config Definitions

Note: minimum required configurations to define in linked instance of TouchTreeGrid:

- xtype : 'touchtreegrid'
- store
- itemId
- columns[] - refer to Appendix A

Configuration	Default	Commonly Updated in Linked Instance	Ignored if simpleList = true?	Purpose
additionalListConfigs	{}			Any config supported by Ext.dataview.List component not already contained in this list can be applied to the auto-generated Grid list component. Configs defined here would override any similar configs defaulted by TouchTreeGrid. Examples: {scrollable: {direction: 'horizontal', directionLock: true}, etc... } For simpleList=true examples: { grouped: true, indexBar: true, etc… }
applyDefaultCollapseLevel	true		Y	Set to false if collapse levels defined on server side.
arrowPctWidth	4		Y	Width of screen to consume for expand/collapse arrow. 4% good for phones and 2% practical for tablets
categColumns[]	[]		Y	Optional means to define categStyle with different column widths than what would otherwise be defined in columns[] categStyle config.
categDepthColorButtons	true	Y	Y	Set to false to not apply categDepthColorsArr colors to "collapse-to-level" buttons in footer (default sencha colors would then apply)
categDepthColors	false	Y	Y	Set to true to apply default color schemes defined by categDepthColorsArr to category rows.
categDepthColorsArr	['#a6a6a6', '#dddddd', 'white']	Y	Y	Array defining colors to apply to category depths 1,2,3, etc… Default color is white for colors not defined for given depth.
categIndentPct	'3'		Y	Each expanded level is indented by 3% of screen by default.
categItemTpl	''		Y	For internal use only.
categItemTplOverride	''		Y	Used by TouchTreeGrid but user could provide own TPL for category rows (see Manual example)

Configuration	Default	Commonly Updated in Linked Instance	Ignored if simpleList = true?	Purpose
cls	'x-touchtreegrid-list'	Y		If specifying custom CSS, always include 'x-touchtreegrid-list' class in array defining any custom class overrides. Example: ['x-touchtreegrid-list', 'x-touchtreegrid-list-example2', 'x-touchtreegrid-list-example2C'] Refer to Project #3 example. 'x-touchtreegrid-list' class applies minimum required CSS defined in TouchTreeGrid.css 'x-touchtreegrid-list-example2' overrides some of the CSS defined by 'x-touchgrid-list'. Refer to treegriddemo.css. Project #1 CSS overrides are copied to to Project #2 and #3. 'x-touchtreegrid-list-example2C' overrides specific configurations from 'x-touchtreegrid-list-example2'. See treegriddemo.css.
colNumberToTruncateForIndents	1		Y	categIndentPct is added for each indented level. To line up columns, we need to subtract this incremental percentage from another column. Column 1 is the default (i.e. the 1st column).
columns	[]	Y		Refer to Appendix A.
columnSorting	false	Y		Typically set to 'true' when SimpleList=true. Each column In the columns array must also have attribute "sortable: true" (refer to Appendix A)
contentItemTpl	''			For internal use only.
contentItemTplOverride	''			Used by TouchTreeGrid but user could provide own TPL for content/detail rows (see Manual example)
customColumnSortEvent	”			Developer-specified event to fire for custom column sorting. TouchTreeGrid provided sorting will be disabled. Refer to logic in handleColumnSort() method for parameters passed with event.
customExpCollapseEvent	''		Y	Refer to Advanced Features discussion
defaultCollapseLevel	99	Y	Y	Tree level to expand to for initial display (99 is fully expanded).
disableExpandCollapse	False		Y	Allows expand/collapse feature for TreeGrids to be disabled and 'frozen' in initial rendered state.
disableSelection	true			Recommended to disable selection when using onItemDisclosure. Reason is to avoid confusion where user could select one row and disclose from another. Refer to TouchTreeGrid.css selectors ".touchtreegrid-item-selected", ".touchtreegrid-item-pressed"

Configuration	Default	Commonly Updated in Linked Instance	Ignored if simpleList = true?	Purpose
				and ".touchtreegrid-simplelist-cell" on how to specify selection and pressing colors. Also refer to TouchTreeGrid configs: pressedCls and selectedCls on how to define custom selectors if needed.
disclosureProperty	'disclose'			Only applicable for onItemDisclosure=true. 'disclose' used to add disclosure icon for all rows. 'leaf' used to only add disclosure icon on leaf rows.
footer	{ xtype: 'toolbar', docked: 'bottom', ui: 'light', cls: 'touchtreegrid-footer' }		Y	Used to auto-add toolbar to bottom of grid for expand/collapse buttons. Additional toolbar configurations such as height, color, etc.. can be defined by overriding this.
header	{ xtype: 'toolbar', docked: 'top', cls: 'touchtreegrid-header', maxHeight: '1.8em', minHeight: '1.8em' }			Used to auto-add toolbar to top of grid for column headers. Additional toolbar configurations such as height, color, etc.. can be defined by overriding this.
headerTpl	''			For internal use only.
headerTplOverride	''			Used by TouchTreeGrid but user could provide own TPL for header rows (see Manual example)
helpHtml	''			Used for demo only.
includeFooter	true		Y	Set false to prevent auto-creation of footer. Not applicable for simple grids (simpleList=true).
includeFooterLevels	true		Y	False to not include 'collapse-to-level' buttons on footer.
includeHeader	true			Set false to prevent auto-creation of column header.
infinite	True			[Touch 2.2x] Refer to Sencha Touch documentation.
itemHeight	47			47 pixel height for list items is standard sencha default
landscapeIcon	''		Y	Refer to Advanced Features discussion.
layout	{type: 'vbox'}			For internal use only. Parent container to linked instance should have Layout = 'card' or 'fit' typically. 'hbox' for horizontal scrolling.
list	{}			For internal use only.
listItemId	'touchtreegridlist'	Y		To add any logic in your controller you should assign a unique itemId for each auto-generated grid.

Configuration	Default	Commonly Updated in Linked Instance	Ignored if simpleList = true?	Purpose
listPlugins	{}			Refer to Advanced Features discussion
listScrollable	true			Generally all lists would be scrollable unless you were implementing something like a dashboard and only wanted the first 4 items for example to appear.
mode	SINGLE			Refer to Sencha Touch documentation for Ext.dataview.List
onItemDisclosure	false	Y		Refer to Sencha Touch documentation for Ext.dataview.List
pressedCls	'touchtreegrid-item-pressed'			TouchTreeGrid.js includes logic to match pressed color to display color to avoid unwanted color flickering for onDisclosure. For cases where row selection is desired (simpleList=true), TouchTreeGrid.css contains 'touchtreegrid-item-pressed' selectors to specify color while pressing an item.
renderers	{}			Refer to Advanced Features discussion
selectedCls	'touchtreegrid-item-selected'			For cases where row selection is desired (simpleList=true), TouchTreeGrid.css contains 'touchtreegrid-item-selected' selectors to specify color for selected items.
simpleList	false	Y		True for basic non-tree grids. Refer to "DOW Hist" example.
singleExpand	false	Y	Y	Typically used for Accordions where only one sibling branch expanded at a time.
store	''	Y		Must be Ext.data.TreeStore for Tree Grids and Accordions (simpleList = false)
styleCategRow	'display: -webkit-box;-webkit-box-orient: horizontal;'		Y	Refer to Advanced Features discussion
styleContentRow	'display: -webkit-box;-webkit-box-orient: horizontal;'			Refer to Advanced Features discussion
styleHeaderRow	'display: -webkit-box;-webkit-box-orient: horizontal;'			Refer to Advanced Features discussion
useSimpleItems	True			[Touch 2.2x] Refer to Sencha Touch documentation.
variableHeights	true			Refer to Sencha Touch documentation.