KRAD - Table semantics - uplift to code standards (including accessibility)

Purpose

Primary purpose is for KRAD table structures to meet the W3C Web Content Accessibility Guidelines (WCAG 2.0) - see usage scenario below. This would make it easier for all applications constructed with KRAD to then inherit this benefit without having to invest additional coding time in each application. (Note: assuming the applications populate the small subset of content tags that will require populating with text, for example, to populate the table header tags and column header tags.)

Detailed Description

There are 3 types of table structures in KNS and KRAD that today lack some of the standard tag structures required to meet W3C and government standards (in the U.S., for example, section 508 and 504 of the Rehabilitation Act, which are being harmonized with WCAG 2.0 now that there is a W3C world-wide technology standard - for additional information, see references section below). These tag structures define the semantic relationships between and among the inter-related elements, for example between header rows and columns and their associated cells.

This is not a problem for sighted users, who can infer the relationships from the relative visual placement and other visual treatments. For example, sighted users can infer that a shaded row placed at the top of a table is a column header row. However, tables that lack the required standard tag structures, don't convey their table inter-relationships programmatically. In real usage terms, this means they can't be accurately interpreted by end users who use standard assistive technologies in order to access applications (applications include websites for enrollment, bill payment or other financial handling, curriculum planning, online library systems, online education, in classroom technologies, or other).

KRAD will provide improved support to applications to enable them to more easily comply with code standards, by adding the missing tag structures into the code-base and templates for all to use.

Usage Scenarios

Users who can't see the user interface, who rely on screen-readers or other technologies to interpret and convey the table structures and content, are given the (audible/spoken) cues that relate the data in each cell with its appropriate meaning (labels/titles or column/row headers).

(Note: assistive technologies rely upon standard code structures in order to interpret and convey the content. Additional information is provided in the references section below.)

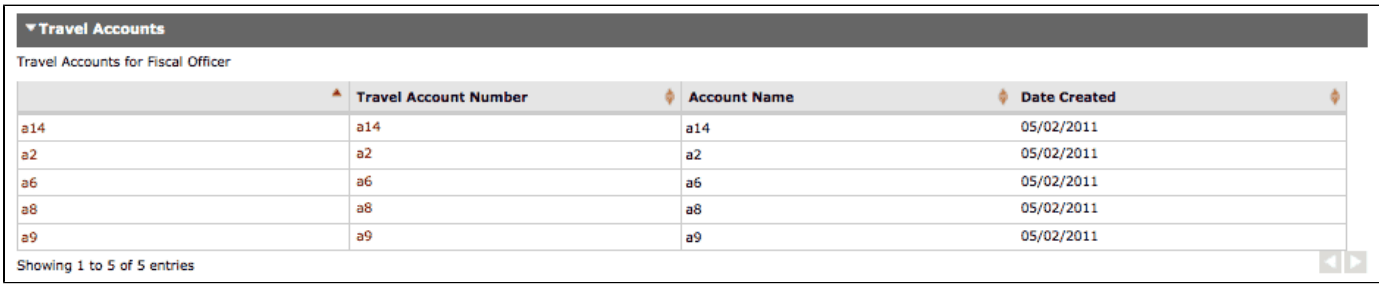
Mocks and Diagrams

Below are descriptions of the 3 targeted types of table structures and how they will be brought up to code standards:

Table Example One:

We currently have a table structure that is programmatically encoded as two separate tables, one for the column heading row and one for the table content (the data itself).

See example below, from the bottom of an inquiry page.



▼ Travel Accounts

Travel Accounts for Fiscal Officer

	Travel Account Number	Account Name	Date Created
a14	a14	a14	05/02/2011
a2	a2	a2	05/02/2011
a6	a6	a6	05/02/2011
a8	a8	a8	05/02/2011
a9	a9	a9	05/02/2011

Showing 1 to 5 of 5 entries

There were reasons why this was done in 2 tables, based on constraints in the widgets used, for example the requirement that we be able to support sortable columns, etc.. But for non-sighted users, the data in the 2nd table is not associated with any conceptual labels or groupings.

One way to fix this is to change this back into a single table, with the appropriate column header markup, but if the sorting capability is lost in that case, that is not a good solution. Given the assumption that this must remain 2 separate tables until we have a table or grid widget that enables column sorting, there is another way to fix this problem. We can use ARIA tags to “fix” the incorrect table structure:

- Wrap a single around both tables.
- Add row = ”presentation” to each individual table element.
- Add back the appropriate role to the elements in each table.

Following is a code snippet example:

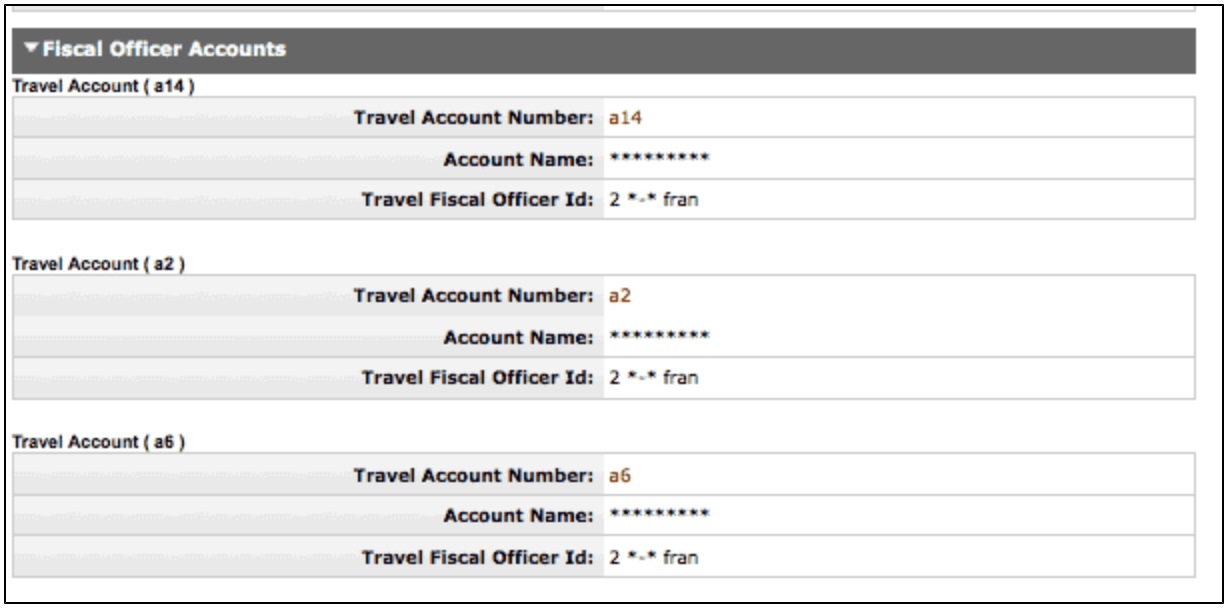
```
 <div role="grid" aria-readonly="true”>
 <table role="presentation">
  <tr role="row">
   <th role="columnheader”>Account Number</th>
   <th role="columnheader">Account Name</th>
   <th role="columnheader”>Fiscal Officer ID</th>
  </tr>
 </table>
 <table role="presentation">
  <tr role="row">
   <th role="rowheader">234235245</th>
   <td role="gridcell">Expenses</td>
   <td role="gridcell">235</td>
  </tr>
  <tr  role="row">
   <th role="rowheader">769856859</th>
   <td role="gridcell">Transport</td>
   <td role="gridcell">237</td>
  </tr>
 </table>
</div>
```

This will look exactly the same to the sighted users, but includes the correct semantics so that the relationships can be programmatically interpreted!

Table Example Two:

The second table structure covered in this requirement, is one that lacks column heading tags and link title tags, and doesn't need them for the sighted users, who can infer the relations between the first and second columns. These are located in read-only inquiry forms, where the user’s focus will jump from link to link when actively navigating, for example, with the arrow keys (rather than passively waiting for the screen reader to read through the entire page).

See example below, from the middle of an inquiry page.



In read-only inquiry forms, user's focus will jump from link to link when actively navigating. So it is important that the live links (the "a#" in the table example above) are associated semantically with their meaning (what are they a link to?). We could fix this in one of three ways:

- Add off-screen column headings, e.g., "Property name" and "Travel Account Property value" (not visible to the sighted user). This is what we're recommending, though either of the below would also work.
  or
- Change this to an ARIA list.
  or
- Add a title tag to each link ("Travel Account Number"), though that would make for redundancy when more passively reading the entire screen.

Table example 2a: Semantics fixed with Offscreen column headers

Below is a code snipped example of how we could fix this with offscreen column headers (not visible onscreen):

```
 CSS:
  .offScreen {
  position:absolute;
  left:-10000px;
  top:auto;
  width:1px;
  height:1px;
  overflow:hidden;
}

HTML:
<h4 id="tableLbl">Travel Account (a14)</h4>
    <table aria-labelledby="tableLbl">
           <tr class="offScreen">
                 <th>Property Name</th>
                 <th>Property Value</th>
          </tr>
          <tr>
                  <td>Travel Account Number</td>
                  <td><a href="#">a14</a></td>
         </tr>
```

Table example 2b: Semantics fixed with an ARIA list:

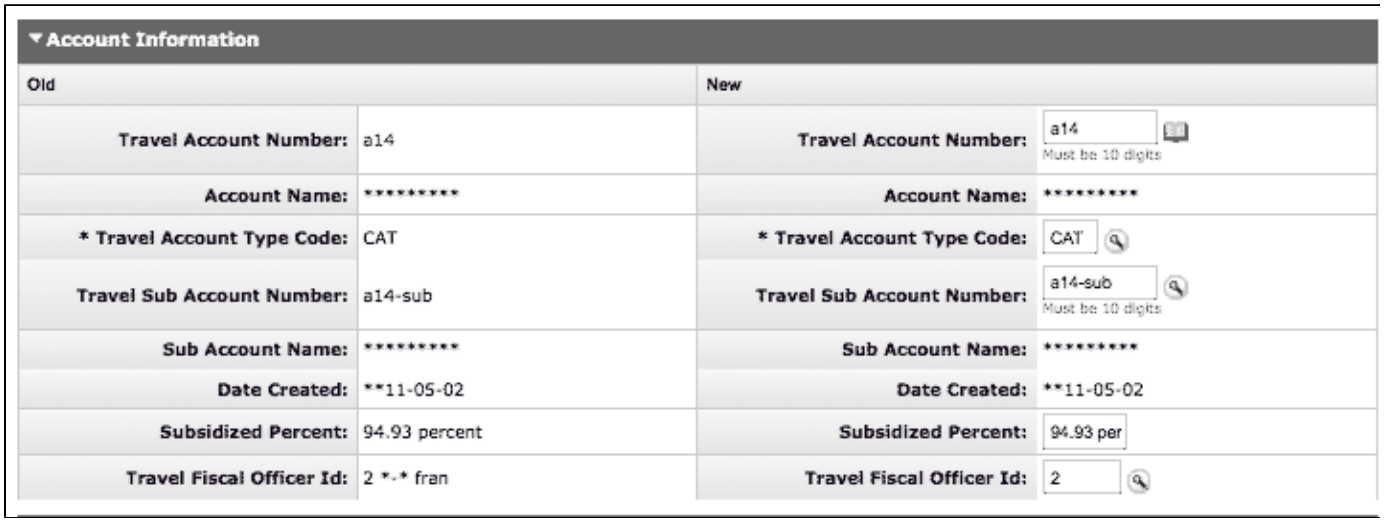
Below is a code snipped example of how we could fix this with an ARIA list:

```
<h4 id="tableLbl">Travel Account Information</h4>
<table aria-labelledby="tableLbl" role="list">
   <tr role="listitem">
            <td role="presentation">Travel Account Number</td>
            <td role="presentation"><a href="#">a14</a></td>
    </tr>
    <tr role="listitem">
             <td role="presentation">Account Name</td>
             <td role="presentation">*********</td>
    </tr>
    <tr role="listitem">
             <td role="presentation">Travel Fiscal Officer Id</td>
             <td role="presentation">2 *.* fran</td>
     </tr>
</table>
```

Table Example Three:

The final example we'll cover is a table structure we have that is currently a multiple column-pair table that lacks column span and column group tags. **The example below comes from a Maintenance (Edit) page.**

In this example below, the problem is that the “Old” and “New” semantics are not included in the markup. For sighted users these semantics can be inferred from the relative placement and visual treatments. But for non-sighted users, the semantics can't be interpreted programmatically by assistive technologies, so they can't be conveyed to the end user.



▼ Account Information			
Old		New	
Travel Account Number:	a14	Travel Account Number:	a14 Must be 10 digits
Account Name:	*********	Account Name:	*********
* Travel Account Type Code:	CAT	* Travel Account Type Code:	CAT
Travel Sub Account Number:	a14-sub	Travel Sub Account Number:	a14-sub Must be 10 digits
Sub Account Name:	*********	Sub Account Name:	*********
Date Created:	**11-05-02	Date Created:	**11-05-02
Subsidized Percent:	94.93 percent	Subsidized Percent:	94.93 per
Travel Fiscal Officer Id:	2 *-* fran	Travel Fiscal Officer Id:	2

We could fix this in one of three ways:

- Use a 3-column table - where the 1st column contains the variable name, the 2nd column contains the old values, and the 3rd column contains the edit controls for the new values. This is the model we recommend, since it is most faithful to the data. The user experience, the user's task, doesn't require and is not aided by the redundant columns of property labels in this case.
- Add column span & scope markup and a hidden row of headers, so the semantics can be interpreted by assistive technologies, and the nested column headers conveyed to the end users
- Use two separate tables, placed side-by-side

Table example 3a: Semantics fixed by changing this to a 3-column table:

This can be fixed by changing this to a 3-column table, where the 1st column contains the variable name, the 2nd column contains the old values, and the 3rd column contains the edit controls for the new values.

Below is an accessible code snippet example that does this:

```
<table aria-labelledby="tableLbl">
        <tr>
            <th>Property</th>
            <th>Old Value</th>
            <th>New Value</th>
        </tr>
        <tr>
            <td>Travel Account Number</td>
            <td><a href="#">a14</a></td>
            <td><input type="text" /></td>
        </tr>
        <tr>
           <td>Account Name</td>
           <td><a href="#">********</a></td>
           <td><input /></td>
       </tr>
       <tr>
           <td>Travel Fiscal Officer ID</td>
           <td><a href="#">2 *.* fran</a></td>
           <td><input type="text" /></td>
       </tr>
   </table>
```

Table example 3b: Semantics fixed in the original single table of 4 rows, with colspan and scope attributes, and a hidden row of headers:

Retaining the structure of the original table, we can fix this by adding the appropriate tags that convey semantic structure.

See code snippet example below. (The example code snippet for table example 2 above includes the css for defining the offscreen header class.)

```
<h4 id="tableLbl">Travel Account (a14)</h4>
<table aria-labelledby="tableLbl">
      <tr>
              <th colspan="2" scope="colgroup">Old</th>
              <th colspan="2" scope="colgroup">New</th>
      </tr>
      <tr>
              <th>Name</th>
              <th>Value</th>
              <th>Name</th>
              <th>Value</th>
      </tr>
      <tr>
              <th scope="row">Travel Account Number</th>
              <td><a href="#">a14</a></td>
              <td>Travel Account Number</td>
              <td><input title ="new travel account number"/></td>
      </tr>
</table>
```

Note that the colspan and scope attributes don't require the ARIA-labelled by tags, but we are recommending including the labelledby tag at this time. ARIA tags will not break the behavior or presentation in browser windows, but they will be useful to browsers that support ARIA (which will

grow over time and which are already considerable - see list of references at bottom below).

Table example 3c: Semantics fixed by changing this into two tables, displayed side-by-side

See example code snippet below:

```
<table role="presentation">     <tr>
        <td>
            <table>
                <tr>
                    <th>Old Property Name</th>
                    <th>Old Property Value</th>
                </tr>
                <tr>
                    <td>Travel Account Number</td>
                    <td><a href="#">a14</a></td>
                </tr>
                <tr>
                    <td>Account Name</td>
                    <td>*********</td>
                </tr>
                <tr>
                    <td>Travel Fiscal Officer ID</td>
                    <td>2 *.* fran</td>
                </tr>
            </table>
        </td>

         <td>
            <table>
                <tr>
                    <th>New Property Name</th>
                    <th>New Property Value</th>
                </tr>
                <tr>
                    <td>Travel Account Number</td>
                   <td><input /></td>
                </tr>
                <tr>
                    <td>Account Name</td>
                   <td><input /></td>
                </tr>
                <tr>
                    <td>Travel Fiscal Officer ID</td>
                    <td><input /></td>
                </tr>
            </table>
        </td>
    </tr>
</table>
```

For the example table shown above (example 3), see also how and why to wrap the multiple controls within a cell within fieldset/legend tags.

Performance

If applicable, list expectations for performance (optimal and worst cases would be fine, give time in seconds):

References

- Rice Accessibility
- Web Content Accessibility Guide 2.0 (WCAG 2.0)
- Web Accessibility Initiative-Accessible Rich Internet Applications (WAI-ARIA)

(Thanks also to Hans Hillen from The Paciello Group for consultations on ARIA and other code structure.)

Requirements Listing

List all requirements (individual verifiable statements) that indicate whether the work for this item has been complete. If there are requirements that are not essential to the functionality but would be nice to have if time allows, enter those under 'Non-Essential':

Primary:

1. For datatables that are provided via a widget that puts the column header row into one table structure and the table content rows into a second table structure (so they are sortable), do the following *(example = at bottom of all inquiry pages)*:
    - Wrap a single around both tables.
    - Add row = ”presentation” to each individual table element.
    - Add back the appropriate role to the elements in each table.
    - For more details, see code snippet in the table example 1 in the *Mocks and Diagrams* section above.
2. For tables that lack column headings and for which adding those into the visual display of the table would negatively impact their usability by sighted users, do the following (*example = all read-only inquiry pages)*:
    - Add off-screen column headings, e.g., “Property name” and “Travel Account Property value” (not visible to the sighted user).
    - For more details, see code snippet in the table example 2a in the *Mocks and Diagrams* section above.
3. For tables that currently are 4 columns, with some columns being identical data (columns 1 and 3 on *all maintenance edit pages* - old/new), do the following:
    - Change these to a 3-column table - where the 1st column contains the variable name, the 2nd column contains the old values, and the 3rd column contains the edit controls for the new values.
    - For more details, see code snippet in the table example 3a in the *Mocks and Diagrams* section above.
4. For other tables that have need to have headings that pertain to multiple columns (that do NOT have redundant identical label columns), do the following:
    - add column span and column group tags.
    - For more details, see code snippet in the table example 3b in the *Mocks and Diagrams* section above.
5. For all tables that are data tables (not tables that are used only for visual layout), ensure that:
    - all data tables have a heading
    - there is an ID on the heading for each data table
    - there is an "ARIA-labelledby=" tag, pointing to the ID (value) on the appropriate table header
    - For more details, see code snippets in the table examples 2a, 2b, or 3b in the *Mocks and Diagrams* section above.

Secondary:

1. When there are multiple controls within a cell, encapsulate them within the standard fieldset/legend tags. This is listed as secondary in this requirement related to tables, because there is another Rice 2.2 KRAD requirement related to adding fieldset/legends to group multiple controls that relate to the same field (in all places, including within table cells).

Dependencies

List any functional or technical work that must be completed before work on this item can begin:

1. item

Issues

List any issues that need to be resolved before work on this item can begin:

Functional:

1. item
2. item

Technical:

1. item
2. item

QA or Regression Testing Plan

List steps needed to test the basic functionality of this update, enhancement, bug fix

1. test/steps
2. test/steps

Checkoff

Functional Analysis Complete? **No (completed by SME)**

Needs Review by KAI? **No (completed by SME)**

Technical Analysis Complete? **No (completed by DM)**

Needs Review by KTI? **No (completed by DM)**

Estimate: **30 hours (completed by DM)**

Technical Design: Link Here **(completed by DM)**

Jira: Link Here **(completed by SME)**

Final Documentation: Link Here **(completed by D**