DRAG AND DROP PICTURE ROW EDITING

Picture row editing includes caption editing. For this discussion, refer to the pdf document “dragNdrop.pdf”.

**BASIC DESIGN:**

The implementation is based on the design as depicted in the diagram. It consists of three coordinated parts: 1) A row of insertion points: purple-colored circles with down arrows pointing to the location at which the drop will place the dragged image; 2) a row of images; 3) a row of captions for photos (wherever photos occur) and/or boxes with the text “NO EDIT” indicating that captions are not provided for the corresponding images (e.g. maps, elevation charts, etc.). These elements are constructed as follows:

* There is a “lead” insert point for every row, not attached to any image; This point remains even when the row has been vacated so that images may still be placed there.
* A vertical alignment of the primary components; the image, a corresponding insertion point to its top right, and caption or blank box below.
* Each image has a space to its right
* Each caption is directly below its corresponding photo
* Each “No Edit” box is directly below its corresponding image

Whenever an image is dragged, its corresponding insertion point and caption accompany it.

Whenever an image is dropped, all components are inserted (insertion pt, image, capt./box)

NOTE: iframes are not directly draggable, and so have been placed in a div container with a sufficiently wide border to “grab”. The cursor over this border changes to illustrate that it can be dragged. In the diagram, the second image in the second row is colored to indicate the association of the three parts. The dotted red lines also show grouping: i.e. the insertion point includes a left-margin, and the image & capt/box include a right-margin. When items are moved to new locations, they maintain their id’s during the edit session.

**IMPLEMENTATION:**

There are essentially three major components to consider during implementation: presentation; saving the edits; dragging; dropping. They will be treated below.

< **Presentation** >

*Basic:*

The elements depicted in the diagram are created in the editDB.php routine. First, the row image data is extracted from the database.csv file. Presentation includes only those rows that are not empty in the database. For each row, php creates:

1. A row (div) of insertion points ($loadIcon). The row has a unique identifier consisting of the character “insRow” plus the row number. The row is a div containing the following elements:
   1. The “lead” insertion point
   2. One insertion point for each subsequent image, whose left margin is determined as indicated in the diagram (a function of the icon size, $alpha, the space between images, $beta, and the image width). Each insert point has a unique identifier consisting of the characters “ins” and a sequentially assigned number.
2. A row (div) of images, derived from the database, with a unique identifier of the characters “row” plus the row number.
   1. Each image is assigned an id sequentially based on its “type”: photo (“pic” + seq. number); non-captioned image (“nocap” + seq. number); iframe – see below.
   2. For iframes, the iframe is placed in a div within the row div, and a border provided sufficiently large to “grab” for dragging. The div is provided the id of “map0”, and the iframe an id of “theMap”. At this time only one iframe is expected at most.
   3. Each image has a right-margin of the assigned fixed space between images: $beta.
3. A row (div) of “captions” whose identifier is “caps” plus the row number. The row contains any combination of the following:
   1. A caption (textarea) edit box for any image whose id begins with “pic”. The textarea is given the identifier of “capArea” + the same sequential number as its corresponding insertion point. The box will also have a right-margin of the defined spacing: $beta. Note that there is a width adjustment provided for textareas, as they seem to always consume more space than the assigned width.
   2. A “No Edit” box (div) for any image whose id does not begin with “pic”. The div is given the id, as above, of “capArea” + the corresponding insertion point number.

*Enhancements*:

1. In addition to the database presentation, there is provided a place to load externally sourced images and then drag them into rows (one at a time). The editDB.php code provides a brief text statement on the edit page, a text box in which to paste a url, and a checkbox requesting the image be uploaded. When this happens, the editDB.js file is triggered – i.e. the checkbox (id = loadimg) change is processed. The routine reads the url, forms an <img> which is then appended to the page. After loading, the image is sized to approximately the normal expected size of images in rows. It is defined as draggable, and there will be no accompanying caption uploaded. Nonetheless, the code allows a caption to be added after dropping the image into a row.
2. Adding rows: if the user desires to add a row to the edit page, he/she may do so by checking the checkbox indicated below the rows of images. The essentially adds a ”lead” insertion point to the row after the last appearing row (up to six rows allowed), as well as the parent div’s for row images and caption fields (empty at this point). At the new insertion point, new images may be dropped.

< **Saving the Edits** >

Two things must be saved and passed on to the “saveChanges.php” routine for writing to the database: 1) the edited row data; 2) edited caption data.

1. Preparing the Save Data:

As the rows of images are already created by the editDB.php code, what remains is to update them whenever changes are made. In order to save the row data – i.e. pass it on to the saveChanges.php code when the edits are submitted – a hidden input is provided on the edit page for each possible row (6 max), each with a unique id of r0, r1, … etc. The picNplace.js routine updates this (hidden input) information by storing the page’s row html in each corresponding input element. The html is then passed to the saveChanges.php routine. An input’s html value is updated: when an item is dragged; when an item is dropped. Since the caption is also the “alt” data for the image, captions are also saved by this routine by updating the image’s html and saving it to the corresponding hidden input whenever a change to an image textarea is detected.

A ‘special case’ occurs when an externally sourced image is dropped into a row, with its corresponding caption textarea. In this case the $captions object will not include the new textarea, as the object was created prior to its existence. In order to add it to the object, a function ‘capureCaps()’ is utilized, which re-established the object and its change event binding. Prior to this call, in the drop routine (below), the object is nulled and the bindings are turned off.

NOTE: The placement of the code occurs in picNplace.js because that’s where the changes to rows occur and the corresponding hidden inputs are updated. There is a timing consideration to note: since the page’s html elements specify the drag and drop functions to invoke, those functions must be defined BEFORE the page is loaded. However, since the rows don’t exist yet, those defining objects ($rows, $captions) are determined AFTER the page load by providing sufficient time in a setTimeout call. The function calls and setTimeout appear prior to the functions defining drag and drop.

1. Processing the Save Data:

When the submit button is activated on the edit page, the data is submitted to the server and processed by the saveChanges.php routine. With regard to the row html, it is read by the routine, and each row is parsed to extract the image type designator (p,f,or n – photo, iframe, or non-captioned image), the row height, the each image’s width and source, and if a photo, its caption (alt). This information is then used to create the row’s “array string” and saved to the database. As the images are processed, the alt image values are also saved in the array $capts, which is then imploded (along with caption count) to form the caption array string. The parsing code is held in the formRowStrings.php routine and included in the saveChanges.php code.

< **Dragging** >

When an image is dragged, its id is captured by the event, and then the routine “reduceImgCnt” is invoked after a half-second delay. At this point the row is re-displayed without the dragged image, its insert and caption. The invoked routine identifies the image as either an externally loaded source image, or one from an existing row, each are treated differently:

* Externally loaded image: the global ‘dragRow’ is set to -1 to differentiate it from regular existing rows. Its attributes are extracted (width and src). It is assigned a sequential id of ‘pic’ + seq. no., where the sequence begins at 100 so as not to conflict with pre-existing ‘pic’ id assignments. Image, insertion point, and capArea html are created for the image, and placed into the globals draggedImg, draggedInsert, and draggedCap, for use during drop. No other adjustments are required.
* Pre-existing image: the parent row is identified, and its id is extracted so that the row no. can be determined. This is assigned to ‘dragRow’. Next, all the children of the parent row are identified, and the id’s are compared to the id of the dragged image to determine the node position of the target item. This node position can then be used to properly identify the insertion point and capfield that go with the image. This way, the draggedInsert and draggedCap are established. Lastly, the parent row (without the dragged image) has its html extracted and saved to the corresponding input on the page for passing to saveChanges.php.

< **Dropping** >

The drop event calls the function “increaseImgCnt()” after a short timeout. As described in the drag event, the parent row id is captured and dropRow is set to its value. An object is established containing the image row’s children. This is checked to determine whether or not any children exist, setting the variable ‘empty’ accordingly. When a dragged item is being dropped into a new row (always true for externally sourced images), the item must be sized to the current row height before proceeding. The sizing will not occur if the ‘empty’ variable is ‘true’. Sizing includes updating the left-margin for the insertion point, and the capfield width. Next, the node no. of the drop-in location is identified so that proper insertions can be made to all targets: insertion point, image, capfield. Lastly, the width of the row, with the image-to-be-dropped included, is calculated. If the new row width is too large, the routine ‘fitToSize()’ is invoked before proceeding, so that all images will fit into the row maximum. The drop point may occur either: 1) at the “lead” insertion point (beginning of a row), or 2) at an image’s designated insertion point. Each situation is differently addressed:

1. For currently empty rows, the items are simply appended to the respective div’s: insertion, image, and capfield. If the row is not empty, the items are appended as ‘first child’ of those div’s (note that for the insertion point row, it will be the second item, as the first is always the “lead” point.
2. In the case where the drop occurs at an image’s insertion point, the node number is used to establish where to “insertBefore” the correct child.

In order to save any captions introduced by dragging in an externally sourced image, the $captions object and events are nulled and then re-established in all rows, which will of course include the new caption. Externally sourced images have caption id’s > 99.

The final act for the drop routine is to extract the updated row html and save it to the hidden inputs on the page for passing to php.