# Pager API

## Pager Overview

The Pager component allows users to break up long lists of items into separate pages. Users can decide if they want paging and how many items should be presented per page.

### Renderer Use

While the Pager *can* be used in a 100% markup-driven mode, it is recommended to use the *data-driven mode*. This mode uses the Pager's built-in integration with Fluid's Renderer, and supports rendering of dynamic multi-page data-driven tables.

NOTE: At this time, the Renderer-based data-driven mode is NOT the default mode for the Pager. To use the data-driven mode, you must configure certain options. For specific information on how to do this, see #Options for the Data-driven Pager below. In future releases, the data-driven mode will be the default mode, and less of this configuration will be required.

### Subcomponents

The Fluid Pager, like many Fluid components, is really one interface to a collection of Subcomponents that work together to provide a unified user experience. In particular, the Pager uses the following subcomponents:

- PagerBar
  - PageList
  - PreviousNext
- Summary
- PageSize
- RangeAnnotator
- BodyRenderer

(For a description of these subcomponents, and an example showing how they might fit together, see Pager Subcomponents.)

These subcomponents are instantiated and managed by the Pager, but each has options that can be used to configure and customize it. These subcomponents, and their options, are described individually below.

---

## Creation

```
fluid.pager(container, options);
```

| Status |
| --- |
| This component is in Preview status |

**On This Page**



**See Also**

- Pager Component description
- Pager Tutorial - Markup-driven
- Fluid Component API

**Still need help?**

Join the infusion-users mailing list and ask your questions there.

## Parameters

### container

The `container` parameter is is a selector, a single-element jQuery, or a DOM element specifying the root DOM node of the Pager markup.

### options

The `options` parameter is an optional collection of name-value pairs that configure the Pager and its subcomponents, as described below in the fluid: Options section.

---

# Supported Events

The Pager component fires the following event:

| Event | Type | Description | Parameters | Parameter Description |
|---|---|---|---|---|
| `onModelChange` | default | Fired whenever the pager's model changes - that is, whenever there is a change to the displayed range of data for the pager, caused by a change to the page index, page size or underlying data | `(newModel, oldModel, that)` | `newModel` is a structure of type `PagerModel` and represents the updated state of the model. `oldModel` is a snapshot of this model before the event was fired. `that` is the overall component `that` |

| | | | | |
|---|---|---|---|---|
| `onRendererPageLinks` | default | Fired when the pager's `pagerBar.pageList` control is preparing to render the page links - this requires that the pageList control is of type `fluid.pager.renderedPageList`. This event supplies the component tree that will be sent to the Renderer which may be manipulated or decorated to adjust the rendering behaviour | (Links Component Tree)<br>New in v1.1:(Links Component Tree, newModel) | The component tree for the links - this will be an array of components with ids beginning with the prefix `page-link:`<br>New in v1.1: A copy of the new `PagerModel` structure |
| New in v1.4: `afterRender` | default | Fired after any rendering of the Pager triggered by a model change, such as a page change, change in number if items per page, etc. This event fires after the initial rendering of the Pager. | `(that)` | The overall Pager component object |

In addition, in this release, the Pager also supports the following events, which are however implementation-specific, and part of a transitional implementation strategy before the adoption of the Data Binder and BeanInvalidationModel:

| Event | Type | Description | Parameters | Parameter Description |
|---|---|---|---|---|
| `initiatePageChange` | default | Fired when the implementation wishes to initiate the change of state corresponding to the selection of a new page | `(pageRequest)` | A structure which includes either the member `pageIndex` representing the new required page index, or `relativePage` representing an offset from the current page position.<br>New in v1.1: The `pageRequest` parameter may also include a member `forceUpdate` which, if `true`, causes the `onModelChange` event to be fired on page change even if the `dataModel` has not actually changed. |
| `initiatePageSizeChange` | default | Fired when the implementation wishes to initiate the change of state corresponding to an updated page size for the visible range | `(newPageSize)` | an integer representing the desired new page size |

For more information about events, see Events for Component Users.

---

## Options

The following options to the creator function can be used to customize the behaviour of the Pager component:

| Name | Description | Values | Default |
|---|---|---|---|
| `selectors` | Javascript object containing selectors for various fragments of the Pager component. | The object may contain any subset of the following keys:<br><br>`pagerBar`<br>`pagerBarSecondary`<br>`summary`<br>`pageSize`<br>`headerSortStylisticOffset`<br><br>Any values not provided will revert to the default. | `selectors: { pagerBar: ".flc-pager-top", pagerBarSecondary: ".flc-pager-bottom", summary: ".flc-pager-summary", pageSize: ".flc-pager-page-size", headerSortStylisticOffset: ".flc-pager-sort-header" }` |
| `styles` | Javascript object containing CSS style names that will be applied to the Pager component. | The object may contain any subset of the following keys:<br><br>`tooltip`<br>`ascendingHeader`<br>`descendingHeader`<br><br>Any values not provided will revert to the default. | `styles: { tooltip: "fl-pager-tooltip", ascendingHeader: "fl-pager-asc", descendingHeader: "fl-pager-desc" }` |
| `strings` | Configuration of short messages and strings which the component uses in its UI | key-value structure with `string` values | `strings: { last: " (last)" }` |
| `sorter` | A function used to sort the table's rows with respect to a column as requested via the table's UI. | `function(overallThat, model)`<br><br>Return: permutation | `sorter: fluid.pager.basicSorter` |
| `listeners` | JavaScript object containing listeners to be attached to the supported events. | Keys in the object are event names, values are functions or arrays of functions. | See Supported Events above |
| New in v1.1:<br><br>`decorators` | Information used to modify columns in the results listing to be sortable or unsortable | One or two arrays of renderer decorator specifications, adding any desired decoration to the headers (e.g. special styling through the a `ddClass` decorator). | `decorators: { sortableHeader: [], unsortableHeader: [] }` |

There are also options available to select and configure each of the subcomponents used by the Pager. These options are listed here, and their contents are described in detail in the relevant sections below.

| Name | Description | Values | Default |
|---|---|---|---|
| `pagerBar` | The overall "Pager Bar" subcomponent which is responsible for the previous/next, page size and page links controls | A subcomponent specification<br>`{type: functionName, options: pagerBarOptions}`<br>Typically the `functionName` should remain as the default of `"fluid.pager.pagerBar"` Configuration of the options structure for `PagerBar` itself is described in its own section below. | `pagerBar: { type: "fluid.pager.pagerBar", options: null }` |
| `summary` | A small textual control representing an overall "summary" of the paging state. Typically holds a message reading similarly to "30-31 of 139 items" | A subcomponent specification<br>`{type: functionName, options: summaryOptions}`<br>Configuration of the options structure for `Summary` itself is described in its own section below. | `summary: { type: "fluid.pager.summary", options: { message: "%first-%last of %total items" } }` |
| `pageSize` | Configuration for a control allowing the user to select the number of items shown per page | A subcomponent specification<br>`{type: functionName, options: pageSizeOptions}`<br>Configuration of the options structure for `PageSize` itself is described in its own section below. | `pageSize: { type: "fluid.pager.directPageSize" }` |
| `rangeAnnotator` | A tooltip that indicates the range of data included covered by each page link. | A subcomponent specification<br>`{type: functionName}`<br>Configuration of the options structure for `RangeAnnotator` itself is described in its own section below. | `rangeAnnotator: { type: "fluid.pager.rangeAnnotator" }` |
| `bodyRenderer` | Used only if the pager is being run in "data-driven" mode. Contains a subcomponent specification for a component capable of responding to model update events in order to render the visible page segment. | A subcomponent specification<br>`{type: functionName, options: bodyRendererOptions}`<br>Configuration of the options structure for `BodyRenderer` itself is described in its own section below. | `bodyRenderer: { type: "fluid.emptySubcomponent" }`<br>This is a dummy implementation that does not render anything. It is used by the default, markup-driven mode of the Pager. |

## Options for the Data-driven Pager

While the Pager can be used in a 100% markup-driven mode, it is recommended to use the Pager's built-in integration with Fluid's Renderer, which supports rendering of dynamic multi-page data-driven tables. If the Renderer is used, various configuration information **must** be provided for it.

To configure the Pager in data-driven mode, you must request the `fluid.pager.selfRender` for the `bodyRenderer` option, as follows:

```
bodyRenderer: {
    type: "fluid.pager.selfRender",
    options: selfRenderOptions
}
```

The `selfRenderOptions` are described below, in the #BodyRenderer subcomponent section.

When Pager is configured for data-driven paging by setting the `bodyRenderer` option to `fluid.pager.selfRender`, several other top-level Pager options become available. NOTE that when the self-rendering is *not* used, these options are ignored.

| Name | Description | Values | Default |
|---|---|---|---|
| `dataModel` | A "pure data" data model which the pager control will operate on. EL paths expressed in component trees and configuration are expressed relative to this model | [data] | `dataModel: undefined` |
| `columnDefs` | Configuration for the rules for extracting and presenting data from the data model into columns. One entry for each column which is to be rendered in the table. This data structure is described in its own documentation section below. | Array of `columnDef` objects, or the string `"explode"`.<br>If `"explode"` is used, the Renderer assumes that the keys in the data model map directly to the `rsf:ids` attached to the associated HTML elements in the table. | `columnDefs: "explode"`<br><br>In version 1.3.1 only:<br><br>`columnDefs: [ { key: "column1", valuebinding: "*.value1", sortable: true } ]` |
| `dataOffset` | An EL path, relative to the root of `dataModel` to which "standard columns" in `columnDefs` are to be referred. These standard columns have EL paths beginning with `*.` - the value held in `dataOffset`will be prepended in place of `*.` (together with the column index) | `string` | `dataOffset: ""` |
| `modelFilter` | Specifies a function signature which will be used to perform the "filtering" portion of the data preparation work required by the `bodyRenderer` with respect to preparing the visible page view. The `modelFilter` is a `function (model, pagerModel) -> data`, where `model` is the value held in `options.dataModel`, `pagerModel` is the pager's model, and the return value is a segment of the data model in a form suitable to be supplied directly to the `bodyRenderer`. A basic implementation is supplied in `fluid.pager.directModelFilter` | function or function name | `modelFilter: fluid.pager.directModelFilter` |

## The ColumnDef structure

The configuration option `columnDefs` takes the form of an array of `ColumnDef` objects, one configuring each required column of the rendered table (a renderer must be in use).

NOTE: Understanding the `ColumnDef` object will be greatly facilitated by looking at the Renderer documentation.

The following table explains the layout and function of the fields in the `ColumnDef` structure:

| Name | Description | Values |
|---|---|---|
| `key` | The core field of the columnDefs structure. Represents a unique key which is used to identify the column | `string` |
| value binding | The EL path through the `dataModel` structure of the data this column is to be taken from. If this is of the form `*.endPath`, the `dataOffset` path will be applied as well as the row number. If it is of the form `startPath.*.endPath`, the `*` will be replaced by only the row number | `string` (EL path) |
| `components` | A cloneable represention of the section of component tree to be used when rendering cells from this column with the [Renderer\|Fluid Renderer - Background]. Typically this will take the form of just a single component, maybe `UIBound`, `UISelect` or `UILink`. See further comments on component expansion below | component tree or `function->component tree` |
| `sortable` | Boolean flag representing whether this column should be made sortable, by clicking on the column header toggle | `boolean` |

### Component expansion

As this component is cloned, once for each cell in its column, it will undergo various forms of expansion - the most important of these is with respect to the special placeholder value `fluid.VALUE`. This may be placed where any string or `UIBound` component is expected, and will be expanded to hold either the value or binding of the correct column cell, with respect to the wildcard path specified in `valuebinding`. A further form of expansion also allows interpolation of read-only values in the middle of literal string, by use of a Smarty-like variable syntax. The string `${VALUE}` will expand to the same value as `fluid.VALUE`, whereas general EL expressions of the form which are supported for `valuebinding` are also supported.

Here is a sample of a `ColumnDef` object with a `UILink` type cell showing both forms of substitution:

```
{key: "user-link",
   valuebinding: "*.userDisplayName",
   components: {
     target: "/dev/sn/profile.html?user=${*.userId}",
     linktext: fluid.VALUE},
   sortable:true
 }
```

Here, the `linktext` field will be bound to `*.userDisplayName`, whereas the URL target will be formed by interpolation with a value from `*.userId`. In this case both of these EL paths will be prefixed by any value in `dataOffset` before fetching.

---

## PagerBar subcomponent

The `PagerBar` is the most substantial subcomponent of the pager, and contains controls operating links for page navigation - these may include previous /next, first/last as well as an array of numbered page links. The dropdown control for selecting page size, `PageSize` may be physically nested within the same container as the markup managed by the `PagerBar`, but is not logically part of the same control.

**Subcomponent Name:** `fluid.pager.pagerBar`

Two PagerBars may be configured for a Pager control, under the names `pagerBar` and `pagerBarSecondary` which were mentioned in the top-level `options` structure above. These may appear at arbitrary positions in the overall markup (top/bottom, etc.), however, the `pagerBar` is primary in that if one of these bars is omitted, it must be the `pagerBarSecondary`.

### options

| Name | Description | Values | Default |
|---|---|---|---|
| `previousNext` | A subcomponent, nested within the PagerBar, which operates the pair of previous/next links for navigating by single pages | A subcomponent specification | `"fluid.pager.previousNext"` |
| `pageList` | A subcomponent, nested within the PagerBar which operates a list of links allowing quick random access to individual pages. This listens for changes in the pager's state, and updates the activity and appearance of the link list. It may also additionally take responsibility for rendering the link list dynamically, in response to the state of the user's data model | Standard implementations are `fluid.pager.directPageList`, which simply accepts a pre-rendered set of page index links which are already in the DOM, and simply takes charge of styling and binding, and `fluid.pager.renderedPageList` which can operate a dynamic strategy to render the page index links, in addition to the responsibilities of `directPageList` | `"fluid.pager.directPageList"` |

| | | | |
|---|---|---|---|
| `selectors` | Javascript object containing selectors for various fragments of the PagerBar subcomponent. | The object may contain any subset of the following keys:<br><br>`pageLinks`<br>`previous`<br>`next`<br><br>Any values not provided will revert to the default. Each of these selectors is expected to identify link components, most likely of tag <a> | `selectors: { pageLinks: ".flc-pager-pageLink", pageLinkSkip: ".flc-pager-pageLink-skip", pageLinkDisabled: ".flc-pager-pageLink-disabled", previous: ".flc-pager-previous", next: ".flc-pager-next" }` |
| `styles` | Javascript object containing CSS style names that will be applied to the Page List subcomponent. | The object may contain any subset of the following keys:<br><br>`currentPage`<br>`disabled`<br><br>Any values not provided will revert to the default. | `styles: { currentPage: "fl-pager-currentPage", disabled: "fl-pager-disabled" }` |

### PageList subcomponent

The `PageList` subcomponent is actually a second-level subcomponent of the overall Pager - it is nested within the `PagerBar` component configured at top level. `PageList` is responsible for managing, and optionally rendering, a list of page link controls which allow quick random access to different page views.

Two implementations are provided:

- `fluid.pager.directPageList` (the default) which simply accepts a pre-rendered and unchanging list of page links present in the DOM at startup, and
- `fluid.pager.renderedPageList` which treats the links in the DOM as a template, and accepts further configuration in order to generate a link list dynamically as the number of view pages alters.

The configuration for `fluid.pager.directPageList` is taken entirely from the `pageLinks` selector configured into the top-level components.

### Options for fluid.pager.renderedPageList

| Name | Description | Values | Default |
|---|---|---|---|
| `selectors` | Allows configuration of a selector name `root`, which defines the root of the document section which is to form the HTML template for the rendered page links. This cooperates with the top-level selector `pageLinks` which is intended to identify individual nodes under this root which correspond to the actual links | Selector value for `root` | `selectors: { root: ".flc-pager-links" }` |
| `linkBody` | An optional selector which allows an "offset" to be specified between the nodes identified by `pageLinks` and the actual <a> tag to be the peer of the page index and `UILink` control. | A selector string | `"a"` |
| `pageStrategy` | A configurable strategy for generating the list of page indices which should have links generated for them, given a particular range of available page indices | A function that returns an array of the indices which should be supplied with page links | `fluid.pager.everyPageStrategy`, which simply generates an entry, and hence a link, for every page in the range |

### pageStrategy implementations

The framework includes three `pageStrategy` implementations:

- `fluid.pager.everyPageStrategy`, which generates an entry for every page in the range
- `fluid.pager.gappedPageStrategy`, which is a factory function which generates implementations which will collapse pages which are not near either the ends of the range or the currently visible page. For example, `fluid.pager.gappedPageStrategy(3, 1)` will return a `pageStrategy` implementation which will return page numbers that are either within 3 pages of the end of the range, or within 1 of the current page position.
- `fluid.pager.fluid.pager.consistentGappedPageStrategy`, will always display same number of page links (including skip place holders)

Integrators can choose to use one of these strategies by overriding the `pageStrategy` options of the `fluid.pager.renderedPageList` subcomponent.

### PreviousNext subcomponent

The `PagerBar` also has a `PreviousNext` subcomponent, which displays controls for navigating to the 'next' or 'previous' page. This subcomponent has no options of its own.

**Subcomponent Name:** `fluid.pager.previousNext`

---

## Summary subcomponent

The purpose of the summary component is to manage a small textual control representing an overall "summary" of the paging state. This typically holds a message reading similarly to "30-31 of 139 items"

**Subcomponent Name:** `fluid.pager.summayr`

### options

| Name | Description | Values | Default |
|---|---|---|---|
| `message` | A string template for formatting the summary message. This may make use of substitution variables `%first`, representing the index position of the first visible item, `%last`, representing the index position of the last visible item, and `%total`, being the total length of the user's data model | A string template | `"%first-%last of %total items"` |

---

## PageSize subcomponent

The `PageSize` component operates a control which allows the user to select the number of items shown at one time on a visible page of the control.

**Subcomponent Name:** `fluid.pager.pageSize`

The default implementation of the `PageSize` component has no options of its own. Its assumption is that the tag identified by the `pageSize` selector configured at the top-level component represents a standard HTML `<select>` control which has been populated with a list of the desired range of page sizes. By default as listed above, the type of the `PageSize` subcomponent is initialised to `"fluid.pager.directPageSize"`.

---

## RangeAnnotator subcomponent

The `RangeAnnotator` subcomponent decorates the displayed page links with a tooltip displaying the range of data values held on that page. To use the `RangeAnnotator`, you must specify the `rangeAnnotator` option, as follows:

```
rangeAnnotator: {
    type: "fluid.pager.rangeAnnotator"
}
```

Note that if the `RangeAnnotator` is used you must provide top-level the `annotateColumnRange` options. When the `RangeAnnotator` is used, several other top-level Pager options become available. NOTE that when the `RangeAnnotator` is not used, these options are ignored.

### Top-level options for the RangeAnnotator

Please note that these options, while used to configure the `RangeAnnotator` subcomponent, must be provided as top-level options to the Pager itself. In future releases, they will become options of the `RangeAnnotator` itself.

| Name | Description | Values | Default |
|---|---|---|---|
| `annotateColumnRange` | One of the values of the `key` fields in `columnDefs`, which will be used to highlight the range of values that a particular column will take, usually by displaying a tooltip as the mouse hovers over a page link. Requires a `rangeAnnotator` component to be configured. | `string` | `annotateColumnRange: undefined` In version 1.3.1 only: `annotateColumnRange: "column1"` |
| `tooltipDelay` | The delay, ms, in hovering over a control to displaying its tooltip | `number` | `tooltipDelay: 300` |
| `tooltipId` | The document id of the displayed tooltip (this markup is generated internally) | `string` | `tooltipIt: "tooltip"` |

## BodyRenderer subcomponent

The component configured as `bodyRenderer`, to the knowledge of the overall Page component, simply has the role of a standard listener to `onModelChanegd`. However, the special function of this subcomponent, if configured, is to perform the work of locating an HTML template suitable for rendering the actual paged view contents, and responding to changes in the pager's model for the purpose of keeping the rendered View updated.

The work of rendering the paged body is split into two parts - firstly, the part of preparing the direct (JSON) representation of the data to be renderered. This is performed by the top-level component configured as `modelFilter`. A `ModelFilter` is simply a function, whose signature and purpose is documented as part of the top-level component options - it need not actually make use its argument `model` to filter the data View, but there is a standard implementation `fluid.pager.directModelFilter` that simply takes the `model` (configured as the top-level `that.options.dataModel`) and extracts just those rows identified by the `pagerModel`.

The standard implementation of the `BodyRenderer` in `fluid.pager.selfRender` locates and makes use of the configured top-level `ModelFilter` in its listener implementation. The `ModelFilter` is invoked to do its work of preparing the data suitable for this event cycle, which is then handed to the Fluid Renderer - Background for rendering. Other implementations of `BodyRenderer` might use other rendering schemes.

### options

| Name | Description | Values | Default |
|---|---|---|---|
| selectors | | | selectors: {<br>root: ".flc-pager-body-template"<br>} |
| keyStrategy | | | keyStrategy: "id" |
| keyPrefix | | | keyPrefix: "" |
| row | | | row: "row:" |
| header | | | header: "header:" |
| renderOptions | | | renderOptions: {} |

### Options for fluid.pager.selfRender

| Name | Description | Values | Default |
|---|---|---|---|
| selectors | Javascript object containing selectors for various fragments of the selfRender subcomponent. | | selectors: {<br>root: ".flc-pager-body-template"<br>} |
| New in v1.1 styles | Javascript object containing CSS style names that will be applied to the selfRender subcomponent. | | styles: {<br>root: ".fl-pager"<br>} |
| rendererOptions | An options structure, to be passed through to the Fluid Renderer when rendering the body of the table | An options structure | {} |
| row | The rsf:id to be used for the container, when generating each successive row of the table | string (an rsf:id) | "row:" |
| header | The rsf:id to be used for the container, when generating the table header row | string (an rsf:id) | "header:" |

---

## The Pager Model

Where the component `that` is constructed by a line such as

```
var that = fluid.pager(component, options);
```

the returned object will have a data member called `model`:

```
that.model
```

This object is the "model" of the pager - note that this is not the same as the data model of the user:

- The Pager's model consists of the state of the paging component, which specifies the page index position, overall data range and page size.
- The user's data model consists of the actual data being paged, which is stored in `that.options.dataModel`.

This, the Pager's model, should be treated as **read-only** via this interface, and should only be manipulated by use of the component's event system.

The pager model is laid out as following:

| Field | Type | Primary /Computed | Description |
|---|---|---|---|
| `pageIndex` | integer | Primary | The current index of the page, expressed 0-based amongst the range of pages |
| `pageSize` | integer | Primary | The number of "items" which may be shown on a page |
| `totalRange` | integer | Primary | The total count of "items" in the underlying user's data model |
| `pageCount` | integer | Computed | The limit of the range in which pageIndex is expressed |
| `[fluid: pageLimit]` | integer | Computed | The limit of the range of items shown on the current page represented by the model |
| `sortKey` | string | Primary | The `key` the `columnDef` for a column whose data is to be used for sorting the table [Optional - only supported for rendered views] |
| `sortDir` | -1 or 1 | Primary | +1 represents sorting in ascending order by the `sortKey`, -1 represents sorting in descending order |

Note that `pageLimit` is not actually stored within the model at any point, but is supplied a computation function `computePageLimit` to match `computePageCount` by which `pageCount` is derived from the primary fields.

---

## Dependencies

The Pager dependencies can be met by including the minified `InfusionAll.js` file in the header of the HTML file:

```
<script type="text/javascript" src="InfusionAll.
js"></script>
```

Alternatively, the individual file requirements are:

```
<!-- Stylesheets -->
<link rel="stylesheet" type="text/css" href="
framework/fss/css/fss-reset.css" />
<link rel="stylesheet" type="text/css" href="
framework/fss/css/fss-layout.css" />
<link rel="stylesheet" type="text/css" href="
components/pager/css/Pager.css" media="all" />
<link rel="stylesheet" type="text/css" href="lib
/jquery/plugins/tooltip/css/jquery.tooltip.css"
media="all" />
<link rel="stylesheet" type="text/css" href="lib
/jquery/ui/css/jquery.ui.theme.css" />

<!-- Scripts -->
<script type="text/javascript" src="lib/jquery
/core/js/jquery.js"></script>
<script type="text/javascript" src="lib/jquery
/ui/js/jquery.ui.core.js"></script>   <!-- New
in v1.3 -->
<script type="text/javascript" src="lib/jquery
/ui/js/jquery.ui.widget.js"></script>   <!--
New in v1.3 -->
<script type="text/javascript" src="lib/jquery
/ui/js/jquery.ui.position.js"></script>   <!--
New in v1.3 -->
<script type="text/javascript" src="lib/jquery
/plugins/bgiframe/js/jquery.bgiframe.js"><
/script>   <!-- New in v1.3 -->
<script type="text/javascript" src="lib/jquery
/plugins/tooltip/js/jquery.ui.tooltip.js"><
/script>   <!-- New in v1.3 -->
<script type="text/javascript" src="lib/json/js
/json2.js"></script>   <!-- New in v1.3 -->

<script type="text/javascript" src="framework
/core/js/Fluid.js"></script>
<script type="text/javascript" src="framework
/core/js/FluidDOMUtilities.js"></script>   <!--
New in v1.3 -->
<script type="text/javascript" src="framework
/core/js/FluidDocument.js"></script>   <!-- New
in v1.3 -->
<script type="text/javascript" src="framework
/core/js/jquery.keyboard-a11y.js"></script>   <!
-- New in v1.3 -->
<script type="text/javascript" src="framework
/core/js/DataBinding.js"></script>
<script type="text/javascript" src="framework
/core/js/FluidRequests.js"></script>   <!-- New
in v1.3 -->
<script type="text/javascript" src="lib
/fastXmlPull/js/fastXmlPull.js"></script>
<script type="text/javascript" src="framework
/renderer/js/fluidParser.js"></script>
<script type="text/javascript" src="framework
/renderer/js/fluidRenderer.js"></script>

<script type="text/javascript" src="components
/tooltip/js/Tooltip.js"></script>   <!-- New in
v1.3 -->
<script type="text/javascript" src="components
/pager/js/Pager.js"></script>
```