YUI Library: Browser History Manager

2007-12-4 **v2.4**

Getting Started with Browser History Manager

1. Required Markup

The Browser History Manager requires the following in-page markup:

```
<iframe id="yui-history-iframe" src="asset"></iframe>
<input id="yui-history-field" type="hidden">
```

1. The asset loaded in the IFrame must be in the same domain as the page (use a relative path for the src attribute to make sure of that)
2. The asset loaded in the IFrame does not have to be an HTML document. It can be an image for example (if you use an image that you also happen to use in your page, you will avoid an unnecessary round-trip, which is always good for performance)
3. This markup should appear right after the opening `<body` tag.

2. Module Registration and the `register` Method

Use the following code to register a module:

```
YAHOO.util.History.register(str module, str initial
state, fn callback[, obj associated object, b scope])
```

Arguments:

1. ***module***: Arbitrary, non empty string identifying the module.
2. ***Initial state***: Initial state of the module (corresponding to its *earliest* history entry). `YAHOO.util.History.getBookmarkedState` may be used to find out what this initial state is if the application was accessed via a bookmark.
3. ***callback***: Function that will be called whenever the Browser History Manager detects that the state of the specified module has changed. Use this function to update the module's UI accordingly.
4. ***associated object***: Object to which your callback will have access; often the callback's parent object.
5. ***scope***: Boolean – if true, the callback runs in the scope of the associated object.

3. Using the `onReady` Method

Once you've registered at least one module, you should use the Browser History Manager's `onReady` method. In your handler, you should initialize your module(s) based on their current state. Use the function `YAHOO.util.History.getCurrentState` to retrieve the current state of your module(s).

```
YAHOO.util.History.onReady(function () {
  var currentState =
   YAHOO.util.History.getCurrentState("module");
  // Update UI of module to match current state
});
```

4. Initializing the Browser History Manager

Before using the Browser History Manager, you must initialize it, passing in the id of the required HTML elements created in step 1:

```
YAHOO.util.History.initialize("yui-history-field",
   "yui-history-iframe");
```

Storing New History Entries: The `navigate` Method

Any registered module can create a new history entry at any time. Doing so creates a new "stop" to which the user can navigate to via the back/forward buttons and that can be bookmarked in the browser. You can create new history entries in your script using the `navigate` method.

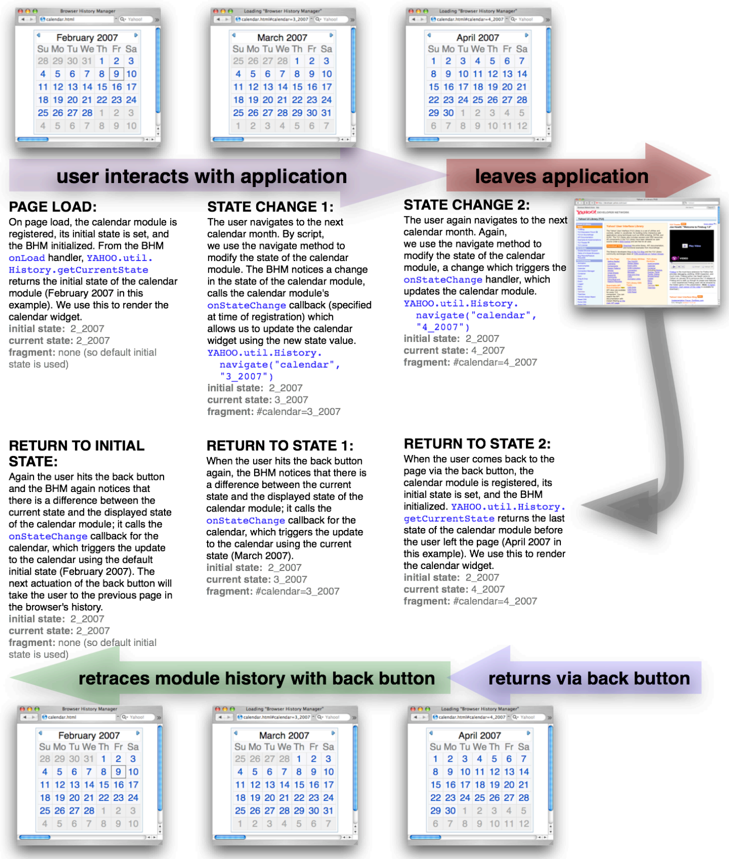
```
YAHOO.util.History.navigate(str module, str new state);
```

Arguments:

1. ***module***: Module identifier you used when you registered the module.
2. ***new state***: String representing the new state of the module.

Note: The navigate method returns a Boolean indicating whether the new state was successfully stored.
Note: The multiNavigate method allows you to change the state of several modules at once, creating a single history entry, whereas several calls to navigate would create several history entries.

A Sample Interaction



PAGE LOAD:
On page load, the calendar module is registered, its initial state is set, and the BHM initialized. From the BHM `onLoad` handler, `YAHOO.util.History.getCurrentState` returns the initial state of the calendar module (February 2007 in this example). We use this to render the calendar widget.
initial state: 2_2007
current state: 2_2007
fragment: none (so default initial state is used)

STATE CHANGE 1:
The user navigates to the next calendar month. By script, we use the navigate method to modify the state of the calendar module. The BHM notices a change in the state of the calendar module, calls the calendar module's `onStateChange` callback (specified at time of registration) which allows us to update the calendar widget using the new state value.
```
YAHOO.util.History.
  navigate("calendar",
  "3_2007")
```
initial state: 2_2007
current state: 3_2007
fragment: #calendar=3_2007

STATE CHANGE 2:
The user again navigates to the next calendar month. Again, we use the navigate method to modify the state of the calendar module, a change which triggers the `onStateChange` handler, which updates the calendar module.
```
YAHOO.util.History.
  navigate("calendar",
  "4_2007")
```
initial state: 2_2007
current state: 4_2007
fragment: #calendar=4_2007

RETURN TO INITIAL STATE:
Again the user hits the back button and the BHM again notices that there is a difference between the current state and the displayed state of the calendar module; it calls the `onStateChange` callback for the calendar, which triggers the update to the calendar using the default initial state (February 2007). The next actuation of the back button will take the user to the previous page in the browser's history.
initial state: 2_2007
current state: 2_2007
fragment: none (so default initial state is used)

RETURN TO STATE 1:
When the user hits the back button again, the BHM notices that there is a difference between the current state and the displayed state of the calendar module; it calls the `onStateChange` callback for the calendar, which triggers the update to the calendar using the current state (March 2007).
initial state: 2_2007
current state: 3_2007
fragment: #calendar=3_2007

RETURN TO STATE 2:
When the user comes back to the page via the back button, the calendar module is registered, its initial state is set, and the BHM initialized. `YAHOO.util.History.getCurrentState` returns the last state of the calendar module before the user left the page (April 2007 in this example). We use this to render the calendar widget.
initial state: 2_2007
current state: 4_2007
fragment: #calendar=4_2007

YAHOO.util.History Methods:

getBookmarkedState(str *module*)
returns str *bookmarked state*
getCurrentState(str *module*) returns str *current state*
getQueryStringParameter(str *param name*[, str *query string*]) returns str *param value*
initialize(str *stateFieldId*, str *histFrameId*)
navigate(str *module*, str *state*) returns Boolean *success*
multiNavigate(arr *states*) returns Boolean *success*
register(str *module*, str *initial state*, fn *callback*[, obj *associated object*, b *scope*])

Dependencies

Browser History Manager requires the YAHOO Global Object and the Event Utility.