
3.7. Managing history

History in an Ajax application is a series of states. In a normal website, clicking the Forward or Back button takes you to a new page; in an Ajax application, you should move between already visited states. For the chapter example, the state will be whichever of the tabs is shown.

GWT harnesses the event structure that we covered in the last section to support telling your application that the history (state) has changed. You can handle these change events by setting up an appropriate event handler and reacting to it, which is what we'll look into in this section.

3.7.1. Handling history in GWT

Ajax applications can employ many techniques to manage history, each more successful in one browser than in another. Some use an iFrame, some react to changes in the URL, others have a timer that checks to see if the URL has changed, and more still use HTML 5's `onhashchange` event. JavaScript libraries, such as DOJO or JQuery, can manage history and hide the various implementation details from you, and GWT is no exception.[5]

[5] If you're interested in how history is implemented in GWT (as of GWT 2.1), IE8 uses HTML5's `onhashchange` event, IE6/7 uses an iFrame, and Mozilla/Opera/Safari are based on a timer checking for hash changes on the URL.

If you run a GWT application that manages history, you'll see tokens appearing in the hash part of the URL. Try running the `BasicProject` application and clicking the tabs for Home, Products, and Contact, and look at the URL; it will look something like the following, depending on the tab selected:

```
http://www.somename.se/Historyexample.html#Home
http://www.somename.se/Historyexample.html#Products
http://www.somename.se/Historyexample.html#Contact
```

Definition

A *history token* is the hash part of a URL (the text from the # symbol). For example, in the following URL, http://www.manning.com/index.html#books, the token is books.

Changing the hash part in a URL is the only change that can happen that doesn't make the browser request a new page from the server (in a normal HTML page, the hash part of a URL is used to jump to an anchor link within the page). In the GWT world, we refer to the hash part of the URL as the history token.

That's the theory; now let's look at how to handle history in GWT in practice.

3.7.2. Implementing history management in your application

To implement history management in a GWT application, you need to do the following:

1. Add a `ValueChangeHandler` to the `History` object; this is the same principle as adding a `ClickHandler` to a `Button` object.
2. When a notable state change happens, you need to add a new history item to the `History` object using `History.newItem(token)`; this will update the browser's URL with a new history token.

3. Implement the `ValueChangeHandler` 's `onValueChange` method to change your application's state based on a presented history token in the URL.

In the `BasicProject` example, history tokens represents which tab content is visible, so it makes sense to use token values: `home`, `products` , and `contact` .

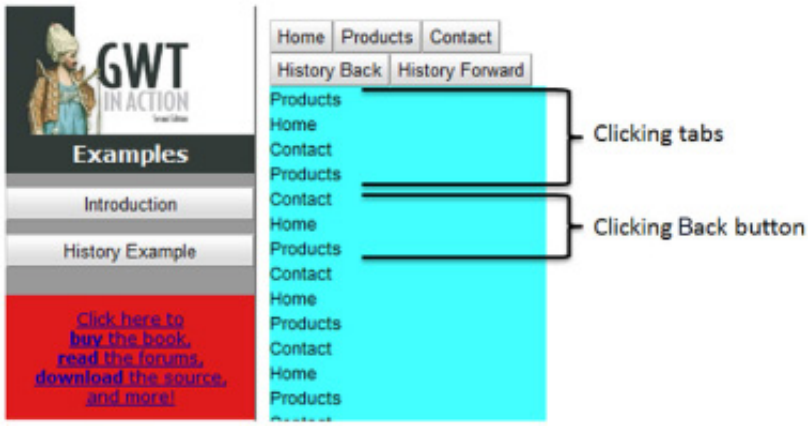
I can't get history working in IE6/7

If you're using GWT history support, you must have an iFrame in your application's HTML page with an `id` of `__gwt_historyFrame` to support IE6/7 (it's usually hidden from view by specifying its style as `position:absolute;width:0;height:0;border:0`).

This is usually added by the creation tools, but you may have deleted it, or perhaps you're adding GWT to an existing application and so have to add this manually.

In order to examine history without having a lot of other distracting code around, let's look at the `historyhelper` example, in particular the class `com.manning.gwtia.ch03.client.history.HistoryExample` . This example is similar to `Basic-Project` but has only the code relating to history management in it (and that code is simplified, for example, defining the tokens as `String` constants rather than as part of an enumeration as in `BasicProject`). Figure 3.7 shows the helper application `HistoryExample` in action.

Figure 3.7. The `HistoryExample` helper application in action—the first four labels are from starting the application and clicking the Products, Home, Contact, and then Products buttons in that order. The next three labels are from clicking the Back button three times; notice that the order is the reverse for these three labels, as you would expect going backward through history.



State is changed in the application by clicking one of the buttons. Click the Products button, and the application reacts by creating a new history token via the `History.newItem` method, with the `PRODUCT String` constant as the parameter.

History is treated as a logical event in GWT. By adding a `ValueChangeHandler` to GWT's static `History` object, GWT will call the handler's `onValueChange` method when the token changes in the URL. The token can be changed either by the application creating new tokens or the user typing in a new URL (or more likely the user clicking Back/Forward in the browser or arriving at a bookmark).

In `HistoryExample` a simple `setUpHistoryManagement` method is called from the constructor:

```
public class HistoryExample extends Composite
                          implements ValueChangeHandler<String>{
   :
  public void setUpHistoryManagement(){
```

```
        History.addValueChangeHandler(this);
        History.fireCurrentHistoryState();
      }
      :
```

Usually the `EntryPoint` class will implement the `ValueChangeHandler` interface for the whole application, and so it will implement the `onValueChange` method (although in the helper-apps example, it's the example composite that implements the necessary interface).

After adding the `ValueChangeHandler` handler to GWT's `History` object, you directly call the `fireCurrentHistoryState` method. This allows the application to react to any token already present on the URL when the application starts—this will be the case if, for example, the user has bookmarked your application at a particular history point and is returning.

Here's the implementation of `onValueChange` that's used in the helper-apps project:

```
public void onValueChange(ValueChangeEvent<String> event){
   String token = null;
   if (event.getValue()!=null) token = event.getValue().trim();
   if ((token==null)||(token.equals(""))) showHomePage();
   else if (token.equals(PRODUCTS)) showProducts();
   else if (token.equals(CONTACT)) showContact();
   else showHomePage();
}
```

The first thing to do is use `event.getValue` to find out the new token (you trim it to remove any hidden white spaces). If there's no token, then you change the state of the application to show the home page; otherwise, if the token matches one of the constants, you show the appropriate page.

Showing the page in the helper application means adding a label to the screen. In a real application it would do more—for the `BasicProject` example, it changes the tab to the relevant content.

Have you run out of history?

Not many things are more annoying than clicking the Back button one too many times and being unexpectedly thrown out of an application. To prevent that from happening, you could add a `ClosingHandler` such as the following:

```
Window.addWindowClosingHandler(new ClosingHandler(){
   public void onWindowClosing(ClosingEvent event) {
      event.setMessage("Ran out of history. Now leaving application, is
            that OK?");
   }
});
```

Now the user gets a warning message from the browser before the application exits.

The history token is simple, merely a reference to the page you want to see. You could envision it encoding more information. For example, a future products page could show many different type of products, perhaps 10 at a time. You could encode the history token to say which subpage of products to show—products 1–10 could have the token `products`, and products 21–30 could have the token `products?page=3`. Showing the right state requires a little parsing of the token, but that's not impossible.

You can also fire history backward and forward programmatically. In the `history-helper` project, clicking the History Back button is the same as clicking the browser's Back button. It's as simple as writing the following in the History Back button's `Click-Handler`:

```
History.back();
```

With GWT's approach to history, your imagination (well, that and Internet Explorer's max URL length of 2,038 characters) is your only limit.

From GWT 2.1, the concept of *places* was introduced. `Place`s can, in some sense, be seen as `History` plus and automate some of the encoding we previously discussed; we won't worry about it for now because `History` is more than adequate for our needs here (we'll cover `Places` in detail in chapter 15 on MVP).

All of the functionality for `BasicProject` is now in place. The only thing left is to make it look pretty and get some of the components in the right place. You do that with styling.