

Bunch Consulting Inc

Developing HTML for Power Users

Is your website protected?

James D. Bunch II

10

Developing HTML for Power Users

Multiple open windows

A power user will often open a webpage in multiple windows to review one set of data while updating another screen. On the older versions of IE, this was as easy as pressing ctrl+n, and getting a new window within the same session. On the newer browsers, opening another instance of the same browser in any way will open the browser within the same session.

Some multiple window issues will be solved in the same way the website deals with multiple users. The difference between multiple users and multiple windows is multiple windows will share the same user information as well as the same data on the server (session information). Without using advanced techniques, it is not possible to tell which window is submitting the information.

Scenario: A user is shopping on a website and places items in the shopping cart. While reviewing the shopping cart, they realize they are missing an item, so they open a second window, perform the search and add the item to the shopping cart. After adding the new item, they close the second window, returning to the shopping cart window. By default their new item is not shown in the shopping cart on the screen, but the item is in the shopping cart on the server. If the user refreshes the screen, the item will generally appear correctly in the shopping cart. The problem occurs if the user proceeds to the checkout without refreshing the screen. For this example the user is allowed to modify quantities in the shopping cart before proceeding to the checkout. If the server does not check to see if the shopping cart has been updated since the client side shopping cart was last refreshed, the quantity of the new item would come across as a blank, which would either be considered a data entry error, or be interpreted as 0, thereby removing it from the shopping cart. One solution to this problem would be to have a version number attached to the server side shopping cart, and have a hidden field on the client. When the user presses checkout in the shopping cart, the versions are checked on the server. If the version information matches, the user is moved to checkout, otherwise they are presented with the checkout screen containing all the information, and a message stating that the shopping cart had been updated.

Scenario: A common business problem that may arise is when a user is entering data into a form which contains a drop-down list. The drop-down list is populated by another screen in the application and does not currently have the value the user needs. The user may open a new browser window, and navigate to the screen where they add new values to the drop-down. Once they have saved the record, they can close the new window. When they get back to the original screen, they can refresh the browser to get the new value to appear in the drop-down. If the application is using AJAX robustly, the refresh may either be automatically or the screen may have a button which refreshes the drop-down list while preserving the rest of the users work.

What happens when two windows modify and save the same record?

1. **Last write wins.** This method says that whatever data is saved last is what will be in the database. This is the simplest solution, and generally happens by default if the situation has not been thought through. The drawbacks to this method are:
 - a. The user is overriding information they have not yet reviewed, and may be clearing out information they were not trying to edit (e.g. The first write may put in a middle name,

- and the second write was meant to add a phone number, but also clears out the middle name)
- b. If the first user deletes the record, the second user may not be informed that their update was discarded. If the database uses active flags to denote deleted records, the second update may also inadvertently restore the record.
2. **First write wins.** This method requires a timestamp or versioning field on the records being updated. If the user attempts to update a record, and the current timestamp or version does not match the value provided in the update request, the update fails. This ensures that the user must have been viewing the same data they are updating. The drawbacks to this method are:
 - a. If the user must enter large amounts of data to update the record, it not only increases the probability the record will be updated while they are working on entering the data, but also means they will have to re-enter all the data once they have refreshed the data on the screen.
 - b. If an automated process is also updating the record, it may be very difficult for a user to update the record. This issue can be solved through the design of the software, but does need to be considered.
 3. **Smart write.** A smart write is a combination of “first write wins” and “last write wins”, but it uses a review / confirmation process to validate the write. Initially the process works like “first write wins”. If the write fails due to being out of sync, the user receives an error message, but the screen is recreated with the data the user had entered. The user then has a chance to review the record, and override the existing record. There are several methods of allowing the user to review the current record:
 - a. If the application is written to allow bookmarking, the user could review the current record by pressing ctrl+n (opening a new instance of the browser), and toggle between the two screens. This is the easiest solution from the programming aspect, but also the most error prone, and least user friendly.
 - b. The application could present side-by-side editors with the original data in one of the editors, and the new data in the other editor. The original data should be presented in read-only mode while the new data should be read-write. The fields containing differences should also be highlighted in some way. This is a method works well if the two editors will fit on the screen side-by-side.
 - c. If the two editors will not fit side-by-side on the screen, another option would be to put the two editors in a tab control, with one tab containing the editor with the original data in read-only mode, and the second tab containing the editor with the modified data. Again, the modified fields should be highlighted.
 - d. The editor could be written to show original values around the data entry field (to the left, right, top, or bottom). A control could also be added to automatically override the modified value with the value from the database for an individual field.
 4. **Record Locking.** This technique allows the user to take ownership of a record to attempt to avoid the overwriting issues. When the user has multiple windows open, this technique no longer solves the problem, although it does limit the issue to a user overwriting their own data. Drawbacks to this technique include:
 - a. Complicated implementation
 - b. Records get “hung” in a locked state until the lock times out
 - c. If the screen is set up to automatically renew the lock, the record may get locked indefinitely.
 - d. If the screen is not set up to automatically renew the lock, the user must manually confirm retaining the lock on a regular basis.

- e. The less often the user has to confirm retaining the lock the longer the lock will take to time out.
5. **Smart Write + Soft Record Locking.** This technique combines the functionality of a “smart write” with the ability to see who else is working on a record. Some implementations may allow a user to see the in-process changes of another user.

Proper use of the Session object

The session object resides on the server and is associated with an instance of a browser. Generally this association is equivalent to a user on a computer. The exceptions to this rule are when multiple types of browsers are open on the same computer (e.g. Safari and IE), and some older versions of IE will start separate sessions when not started from an existing instance. Since the connection is not always persevered between page requests, it is not possible for the web server to associate the session with the connection, therefore a temporary cookie is used as an identifier when communicating with the server to identify the session. This means that there may be multiple windows associated with a single session object on the server. The session object is a good place to put information such as login credentials, and user specific settings. Using custom classes to hold the data will make code much more readable, efficient, and will avoid many run-time errors. This technique will also allow you to make multithreaded calls more efficiently.

The session object should NOT be used to directly store page data or page state information.

Scenario 1: A user is editing record X, so it is retrieved from the database and stored in the session object. After the user modifies the record they click save, and the record is retrieved from the session object, populated with the new data and saved back to the database. This is a very efficient way of updating a record, except it has a major bug as demonstrated in Scenario 2.

Scenario 2: A user is editing record X so it is retrieved and stored in the session object. They then get a call and a change needs to be made immediately to record Y. The user does not want to lose their work on record X so they start a new window, navigate to record Y, which loads record Y into the session object (replacing record X). The user then modifies record Y and saves their changes to the database. Now they close out the new screen and return to editing what they believe to be record X in the original window (all of record X's data is still in that window). When the user makes their changes and saves the record, the record they pull from the session object is record Y, which gets populated with all the data from record X, and saved back to the database. The result of all this is record X is never updated, and record Y ends up getting updated with the data meant for record X, and record Y's data is discarded.

*NOTE: There are techniques which cache data effectively within the session object, this statement is meant for data object cached directly on the session object.

Delay writing to the database

Sometimes it may be necessary to spread data entry over several screens. When there is too much information to display on a single page, it is common practice to use a tab group to display the information. Data entry wizards are also examples of multiple screens being used for data entry. When using multiple screens to gather data for a record, it is good practice to gather all the data before writing the record to the database. A user may change information in the first tab, change to the second tab, and change some more information, and then change their mind about updating the record, and close out the screen. If the application is writing to the database before all the information is gathered, they would have updated the information in the first tab even though they never saved the record. When

using a wizard, the record may not be valid until the wizard is complete, so writing to the database is not possible. To save the data between the screens, the application can:

1. Store the information in hidden variables
 - a. Each of the fields on the non-shown screens can have their own hidden input, but lists within the wizard must be handled carefully
 - b. The view state (C# and VB.net) can be used to store an object which contains the data from the wizard.
2. Store the information in the session object.
 - a. An object is created to store the information from the wizard
 - b. A random unique identifier is generated to identify the object within the session object
 - c. The random unique number is placed in a hidden field on the page
 - d. When the page is posted, the object is looked up by the identifier, and updated based on the posted data.
 - e. *NOTE: This technique does not save the information if the session times out
 - f. *NOTE: A large number of users on the server using this functionality may put a strain on the server due to memory constraints.

Multi-Threaded Application

A thread is a process which can run without blocking the main execution of a program. Imagine a scenario at home. Two kids need to be taken to practice, one to football, the other to baseball. One solution would be loading both kids into the car, dropping the first kid off at football practice and then taking the second kid to baseball practice. This is the single threaded method of solving the problem (you are the main thread doing all the work). The next approach would be to ask your spouse to take one of the kids to practice. This is the multi-threaded approach. In this scenario you are still the main thread doing part of the work, and your spouse is the new thread which takes over part of the work. There is another scenario where you ask your spouse to take one of the kids to practice and ask your neighbor who has a kid on the same team to take the other kid to practice because you have to go to work. While this may not be the ideal parenting style, it is very efficient. This is the optimized multi-thread approach.

Even if a web based application never creates and executes a thread, every web-based application is multi-threaded. When a web-browser is communicating with a server, it creates multiple threads in order to connect to the server multiple times so it can download the main html page, images, style sheets, etc. simultaneously. This allows the browser to show the page as efficiently as possible since no single resource (except possibly the HTML) can block the rest of the page from being processed. Multiple browsers, whether they are on the same machine, or on different machines, will create separate threads on the web server. If the application is not designed as a multi-threaded application, it may have serious consequences which will be VERY difficult to diagnose.

Do not assume the user is using a browser

This goes beyond not assuming your users are using a particular browser. A power user may not be using a browser at all. A power user may write an application which reads your website, and posts data back. If written correctly, it is impossible to determine the difference between this type of user and someone using a browser. The reason is, the only things your server knows about a client machine (other than networking information such as IP address) is what the client decides to share. Don't be too alarmed by this, because most of the time it is not malicious hacking. Plus, if someone has taken the time to write a program to automate your website, it means they see value within your website.

What this means is, even though your website may validate all the users input on the screen before letting the data post back to the server, it does not mean that all the data being posted back to your server is valid. Something as simple as the user disabling JavaScript on their browser could also cause your server to received invalid data.

Data validation is important on the webpage before the page is allowed to post to the server because it is more efficient. The user does not have to wait for a page to reload to see if they missed a field, or typed something incorrectly. It also keeps some bad requests off your server. The fewer requests the server must handle per user, the more users the server can support. This level of validation makes the website “pretty”, but does not make it secure.

Validation on the server is what makes your website secure, and helps ensure valid data in the database. If validation fails on the server side, the page should be presented back to the user the same as if the validation had occurred on the client side through JavaScript. You cannot assume that a page that gets past the client-side validation (JavaScript), but fails server-side validation is an attack. A user may have JavaScript disabled by default on their browser until they believe they can trust a site. If your website requires JavaScript in order to show up, you are requesting your customers trust you before giving them any information about you. The .Net framework currently defaults to disallowing certain characters to be submitted from a web-page. The idea is to disallow characters which could be used to hack into a database driven website. This option should NOT be enabled. An application does need to be guarded from hacking attempts, but this option is a way of avoiding instead of solving the problem. If this option is enabled and your user tries to send you a message that says “My server is down and It’s costing me money” your website will respond with an error message because it sees the single tick (') as a hacking attempt. If you do not know how to guard against hacking attempts, contact a developer or development company such as Bunch Consulting who can help. There are also several books and websites available if you prefer to research the subject.