Foreword

The success of Ajax is a curious one. It's hard to point to a particular release, product, or article that signaled the arrival of what we now call Ajax. It seemed to have just happened. Even the article by Jesse James Garret which gave us the name Ajax, laid no claim to its invention, but instead pointed to it as a curious phenomenon worthy of a second look. And now that we're all so aware of its presence, we can't really come to any agreement on exactly what it is "Ajax" means. Listen to 20 experts speak on the subject and you'll hear no less than 22 different definitions. And if I had to summarize their opinions, I would be forced to conclude that Ajax is simultaneously the best and worst thing to happen in software in the last 15 years. Yet despite the fact that we don't know where it came from or what it is or whether it's good or not, everyone in the software industry seems eager to launch their next product with a sleek new Ajax interface. From the technologist's perspective, it doesn't seem to make sense. Browsers are limited in their capabilities, difficult to develop with, and plagued with inconsistencies. They rightly point out that from their perspectives, it looks like a pretty bad proposition.

But the beautiful thing about Ajax is that it is not being driven by the technologist alone. There is another force working to temper to the technologist's obsession with architectural beauty—it's users' ever-increasing expectation that software should simplify, not complicate, their lives. So while the technologist bemoans the browser as a crufty place to develop software, the user praises it for familiarity and comfort. It reduces all the complexities of Internet connectivity down to a few key concepts: address, link, forward, back, and search. This is the language of the Web and users are happy with this restrictive view of technology. And why shouldn't they be? In addition to being familiar to them, the added constraints have forced software developers to think more carefully about what users want. The overwhelming theme in successful Ajax development is to do what makes sense for the user, despite the technology available to you. And while this is clearly a very healthy approach that leads to innovation in application development, the price we must pay as software developers is paid in increased pain and frustration.

Increasingly as I hear about new projects there is a common chorus of "and, of course, the UI is going to be very Ajaxy." Generally, I can gauge how much progress they've made by the level of affection they still feel for the project. Those who are just beginning are thrilled with the prospect of being able to work in an area that is getting so much attention. They'll talk about the process of selecting a framework and maybe some rough descriptions of the early UI mockups. At this point, they've built a few small examples without a great deal of fuss. It is very easy to conclude at this point that the frustrations of working in a browser are exaggerated and contented by the technological conquest, they decide to treat themselves to one of those fashionable espresso drinks. A few months later, when I run into them, they're starting to be a bit

more evasive about the project. They'll inevitably tell me how a few tasks that initially seemed so easy proved to be a bit more challenging. "We were planning to have this update in real time, but it just took way too long to load." I can start to detect that their faith weakening and I try to offer them words of support, but I know the test of endurance is just beginning. This is when I usually advise that they give up the espresso drinks in favor of tea. Tea provides a more sustaining and gentle dose of caffeine; coffee will betray you in your time of need. Putting the finishing touches on a good Ajax application is, most definitely, a time of need. Often though they remain optimistic that their nearing the end of the real challenges they usually laugh off the suggestion.

The real meltdown starts when projects start to answer the question: What do our users think? Most people wisely answer this question with user testing or early beta releases. Others, perhaps the same crowd that doesn't wash fruit before eating it, charge ahead with a full release confident they've anticipated the exact needs of their users. It is these people that I pity most. At this point, both sets of developers realize a few things. First, some of the decisions they made to avoid harder problems were actually bad decisions for users. Secondly, they realize that there is no testing like actual use. Now these realizations are not unique to UI's built-in Ajax. I don't know of a single successful project that has avoided this particular stop along the way. What is unique to Ajax applications is that it now becomes increasingly hard to resolve these issues because there are so many elements that seem to conspire against you.

This is usually where the browsers behavioral differences start to show up. Users are reporting that on one browser, their menus are showing up in the wrong location. On another browser the text is wrapping. On yet another browser, it all works great except after about 15 minutes of use the whole thing begins to flicker annoyingly when anything changes. Second, you find that use patterns are not exactly what you had expected and parts of the UI must be changed—which would be fine, except the flexible expressiveness of Javascript that had once been so charming now seems downright offensive and rude. I have tried on numerous occasions to seamlessly refactor large JavaScript code bases and have never been pleased with the results. On top of all of these complications, one fact remains: your application must be good for users. So beyond the immediate frustrations I'm describing, the primary goal of maximizing user experience still remains. And when you do launch your application, this is the only thing anyone will ever see. Did you make an application that serves the needs of your users?

This is Google Web Toolkit's mission in a nutshell; make it much easier for developers to confidently answer "Yes" to that question. We grew tired of attacking the headaches of Ajax development in Sisyphean manner tirelessly shoving the browsers around without ever really gaining any momentum or mechanical advantage. That approach inevitably leads to a situation where you eventually know what is good for the user, but can never quite reach it because you're effectively building a house without the

luxury of a hammer. GWT makes the most of existing tools. There are some good hammers in software engineering, so it was a little bewildering to us why none could be used effectively in Ajax development. We're pretty adamant that the way to ensure that web applications continue to improve is to leverage the good engineering tools and practices that already exist. So rather than bemoaning the status quo, GWT allows you to write your Ajax code in Java, leveraging concepts and patterns that have become very familiar to UI developers; develop using proven development environments that include good code completion and refactoring tools like Eclipse; debug your apps by running them in a real browser, using a solid debugger; then use a compiler to translate all that Java code to tiny, high-performance JavaScript that automatically works around most browser quirks without so much as a nod from the developer. And of course, make it possible to slip seamlessly into JavaScript when the need arises to do things we never even anticipated. GWT is not about trends and language wars, it's about pragmatism and sound solutions.

This is why it pleases me greatly to see that David Geary's GWT Solutions holds true to its name and focuses on concrete and practical solutions. This is very much in keeping with the spirit of GWT. It is not enough to talk about design patterns and elegance of code without following through with why such things are relevant to your users. David does a very nice job here of giving us something that goes beyond the contrived example. Each of the solutions work well on two levels. First they take us through the process of building good user interfaces in GWT making apparent the common patterns and even calling attention to many of the likely pitfalls. But secondly, each of the solutions is actually reusable in the form that he presents it. I will not be surprised to see many of his examples showing up in future GWT applications. I think readers will agree with the effectiveness of this approach. In fact, to hold the solutions to the same crucial metric that I apply to GWT itself: Will *Google Web Toolkit Solutions* aid you in creating applications that serve the needs of your users? I would have answer, Yes.

Kelly Norton, Google