Firefly: Embracing Future Web Technologies

W. Roby*[a], X Wu[a], T. Goldina[a], E. Joliet[a], L. Ly[a], W. Mi[a], C. Wang[a],
Lijun Zhang[a], D. Ciardi[a], G. Dubois-Felsmann[a]
[a]IPAC, Caltech , Pasadena, CA, USA 91125

ABSTRACT

At IPAC/Caltech, we have developed the Firefly web archive and visualization system. Used in production for the last eight years in many missions, Firefly gives the scientist significant capabilities to study data. Firefly provided the first completely web based FITS viewer as well as a growing set of tabular and plotting visualizers. Further, it will be used for the science user interface of the LSST telescope which goes online in 2021. Firefly must meet the needs of archive access and visualization for the 2021 LSST telescope and must serve astronomers beyond the year 2030.

Recently, our team has faced the fact that the technology behind Firefly software was becoming obsolete. We were searching for ways to utilize the current breakthroughs in maintaining stability, testability, speed, and reliability of large web applications, which Firefly exemplifies.

In the last year, we have ported the Firefly to cutting edge web technologies. Embarking on this massive overhaul is no small feat to say the least. Choosing the technologies that will maintain a forward trajectory in a future development project is always hard and often overwhelming. When a team must port 150,000 lines of code for a production-level product there is little room to make poor choices.

This paper will give an overview of the most modern web technologies and lessons learned in our conversion from GWT based system to React/Redux based system.

Keywords: Firefly, JavaScript, Visualization, LSST, IPAC, GWT, React, Redux

1. INTRODUCTION

We are currently in the "golden-age" of JavaScript web technology. We now can build web applications on a scale that would have never been considered a few years ago. With its unprecedented sophistication, web development is constantly changing and improving in speed, capabilities and ease of development.

We began writing the Firefly archive front end system in 2008 using the GWT framework (called Google Web Toolkit at the time, now it is just GWT[2†]). GWT provided the key elements that we felt Firefly needed. Unfortunately over time, GWT has become outdated, inflexible and is losing support.

The Large Synoptic Survey Telescope[1] (LSST) will be an 8.4 m optical survey telescope sited in Chile and capable of imaging the entire sky twice a week. Because the LSST project will go into 2030, we realized that Firefly needed updating. To meet the demands of ever-changing technology, the Firefly team of developers had to make the critical decision to go with the most current, robust combination of technology that catapults Firefly into the next decade of functionality. With much learning and labor, our team has made the transformation to JavaScript ES6 using a React/ Redux framework.

The journey to embracing these technologies is best understood with an overview of the JavaScript Ecosystem.

* roby@ipac.caltech.edu; 626-395-8681; http://www.ipac.caltech.edu/; http://lsst.org/
†http://www.gwtproject.org/

Software and Cyberinfrastructure for Astronomy IV, edited by Gianluca Chiozzi,
Juan C. Guzman, Proc. of SPIE Vol. 9913, 99130Y · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2233042

2. JAVASCRIPT

The JavaScript ecosystem is exciting. The emphasis on new programing techniques and paradigms is the most we have seen in years. Not a "toy language" or just limited to the browser anymore, JavaScript has come into a unique time when more and more developers are writing large applications, building tools, solving problems, and implementing computer science concepts like never before. While the popularity of a language is hard to gauge, it is worth noting that, at the time of this writing RedMonk and Stack Overflow have it as the most popular language. Tiobe has it as the seventh and PYPL has it at fifth in popularity. See Table 1.

Table 1. Language Rankings.

Organization	Ranking	URL
RedMonk	1	https://redmonk.com/sogrady/category/programming-languages/
Stack Overflow	1	http://stackoverflow.com/research/developer-survey-2016#most-popular-technologies-per-occupation
Tiobe	7	http://www.tiobe.com/tiobe_index
PYPL	5	http://pypl.github.io/PYPL.html

2.1. Language Growth

Developed in 10 days by Brendan Eich, JavaScript has come a long way since it first came out in the late '90s. However, any developer using it before 2012 might not believe that. JavaScript's official name, ECMAScript, comes from the ECMA International standards organization. Versions of JavaScript are referred to by "ES." JavaScript was stuck at ES3 for a long time. This created an environment that made it fragmented and difficult to use on multiple browsers. It did not start moving forward again as a language until 2011 with ES5. The language started maturing by adding features such as better array manipulation, better specification, JSON object encoding, and a strict mode. The recent standardization to ES6 in 2015 is significant improvement in JavaScript. It went from being a "scripting-language-with-some-interesting-concepts-if-you-used-it-carefully" to a powerful, sophisticated language. Table 2 lists many of the important new features that were added. Table 3 shows a few examples.

Table 2. Some of the new language features in JavaScript ES6 (https://github.com/lukehoban/es6features#readme)

Arrow functions	Classes	Enhanced object literals	Rest, Spread
Template strings	Destructuring	Defaults	Proxies
map, set, weakmap, weakset	Let Const	Iterators	More APIs: Math, Number, String, Array, Object
Promises	Reflect API	Symbols	Generators

Table 3. Some ES examples.

ES3 or ES5	ES6
`function add(a,b) { return a+b; }`	`const add = (a,b) => a+b;`

`function showHelp(text,title) {`
`  title = title || 'This is help';`
`}`
	`function showHelp(text,title= 'This is help') {`
	`  …`
	`}`

`function showNumbers(a,b,c) {`
`  console.log('a= ' + a + ',`
`        b= '+ b + ', c=' + c);`
`}`
	``function showNumbers(a,b,c) {``
	``  console.log(`a= ${a}, b= ${b}, c= ${c}`);``
	``}``

`function objValues(obj) {`
`  var a= obj.a;`
`  var b= obj.b;`
`  var c= obj.c;`
`}`
	`function objValues(obj) {`
	`  let {a,b,c}= obj;`
	`}`

The ECMAScript standard process is continuing to improve. Because the ES6 release took several years to complete, the committee has adopted a new process called TC39 and will now release yearly updates with incremental improvements. The releases will be named according to their year (ES2016, ES2017, etc)[*]. Therefore, ES6 is now officially referred to as ES2015. However, we will continue to refer to it by the original, more common name of ES6.

2.2. Environment

While JavaScript ES6 is very powerful, it would not be near as helpful if it was not for other breakthroughs in the environment. To really understand this, we have to look at Node.js. People who have passingly heard of Node.js know it as a way to have a JavaScript-based web server. However, it is much more general-purpose than that. Node.js is a JavaScript runtime environment that exists outside of the browser. It has made JavaScript a general purpose scripting language such as Perl or Python. This has made for a host of JavaScript based tools that run in the Node.js environment.

The number of development tools to come out of the Node.js environment has grown significantly. They fall into several categories:

Transpilers[†]. ES6 has been out for a year, but is only running on the newest browsers. A JavaScript transpiler converts JavaScript written in ES6 to ES5. This allows us to take advantage of the ES6 features immediately. The most popular transpiler is Babel.

Module bundlers. In the past, all JavaScript for a web page was written in one or two very, very large files. There was no concept of namespacing or modules. That has all changed, ES6 has modules as part of the language. However, these applications still need to load into a web browser in an efficient way. Module bundlers work similar to a linker in C, they take all the JavaScript files and roll them into one file for the browser to load. Webpack and Browserify are the two most common ones.

Linters. Just as with any other language, a linter will improve code quality. JavaScript has many. JSLint and JSHint are good and have been around for a while. We found ESLint to be the best because it is far more configurable than the others. You can add rules specific to coding style, React, JSX[‡] and even write your own. There is a whole community writing ESLint plugins.[§]

Package Managers. The JavaScript world holds a high value for DRY (Don't Repeat Yourself). Reusable modules are key to this. NPM is, by far, the most popular package manager, hosting over 250,000 modules that any developer can use to build upon. Taking advantage of existing work is extremely easy in the JavaScript environment. Firefly currently uses 27 runtime modules (ui, utilities, network, etc) and 47 development modules (testing, linting, transpiling, etc) from NPM. Reusable modules represent a critical piece of JavaScript development. For building, NPM has ways to set up local mirrors. Also, because of recent events, NPM has put policies in place to ensure the stability of modules so that they are not arbitrarily removed from the system[**].

Table 4 shows a list of all these components with their URLs. Putting it all together, we assemble the development flow, shown in Figure 1.

[*] http://www.2ality.com/2015/11/tc39-process.html

[†] https://www.stevefenton.co.uk/2012/11/compiling-vs-transpiling/

[‡] https://facebook.github.io/react/docs/jsx-in-depth.html

[§] https://www.npmjs.com/search?q=eslintplugin

[**] http://blog.npmjs.org/post/141905368000/changes-to-npms-unpublish-policy

Table 4. Components for building Web Applications

Description	Name	URL
JavaScript runtime		https://nodejs.org/en/
Package Manger	NPM	https://www.npmjs.com/
JaveScript Linters	JSHint JSLint ESLint	http://jshint.com/ http://www.jslint.com/ http://eslint.org/
Tranpiler	Babel	https://babeljs.io/
Module Bundlers	Webpack Browserify	https://webpack.github.io/ browserify.org/

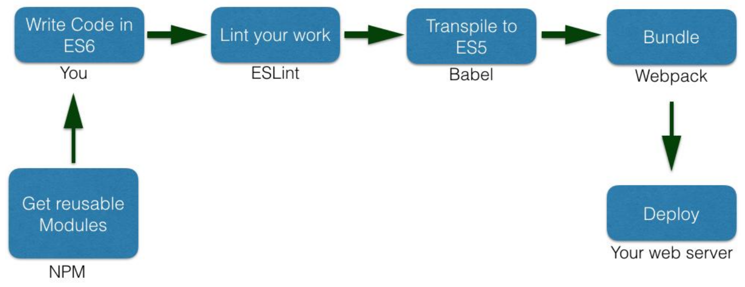

Figure 1. The development flow for Firefly using NPM, ESLint, Babel, and Webpack.

3. WEB DEVELOPMENT FRAMEWORKS

3.1. The Challenge

Probably the biggest challenge to a starting web developer is choosing what tools or framework to use. The problem is that there are just so many. Some have been around since the mid-2000's (like GWT) and are fading. Others are used but not that popular. Some were considered "state-of-the-art" as recently as three years ago but now that is debatable. Some are opinionated defining what is running on the server side, network protocol, data structures, etc. while others are very flexible. New ones are popping up every month or two. Some are improvements on some popular package but not very popular themselves. They might be open source or very tied to a company. There is a website (http://todomvc.com/) that demonstrates this by implementing the same simple todo applications in over 60 different environments or combinations. It can be rather overwhelming.

3.2. Evaluating Frameworks

The various frameworks can be parsed down to a few categories (see table 4). While these categories are helpful to begin thinking about frameworks we further considered several other factors. How hard is it to understand? How popular is it? Who is using it? How flexible? How fast? What is the data paradigm? Does it scale to a large application? Are the key concepts easily learnable?

Looking at both popularity and speed, we were able to cut the options down to Angular, React, Backbone, and JQuery. We soon eliminated JQuery. Although it is very popular, its original purpose was to be a DOM manipulation library for web pages, rather than a framework for large-scale web apps. A JQuery-centric design would not scale well to our application needs. It lacks the big design concepts of the others. Further, some of the original purposes of JQuery, such as cross-browser support and easy DOM API, is becoming less necessary on modern browsers that have a more standardized, advanced API .

Backbone is also popular, and its MVC-based frameworks has the right concept of separation of concerns. However, MVC and HTML templates can get hard to manage when the application gets big[*]. When multiple models begin talking to multiple views, it becomes difficult to understand the internals of the application[†].

This brought it down to Angular or React. Angular is very popular. However, the data model is not very clear. Angular divides the code up between "Angular HTML" and JavaScript. We were concerned that the code would be hard to maintain. Angular is also very extensive with layers of concepts to know. It has a steep learning curve. Angular is probably great for prototyping and smaller applications. We were concerned that it would not work well for a code base as big as Firefly. With these concerns, we eliminated Angular. Eighteen months ago, React was not that popular and very new on the scene. It was touting some radical and interesting development concepts, such as virtual DOM, one-way reactive data flow, JSX, and small API.

We chose React because it was the clear winner in all our categories when we took a closer look at the data model and paradigm for building a UI. We needed something that would lend itself to developing large, stable web-applications. We needed to be able to bring on new developers who would be able to maintain Firefly for years. Beyond that, we needed a solid, large architecture design paradigm. React is flexible and scales well; writing reusable components is core to the system. Also, it is known to be very fast[‡].

Table 5. Types of JavaScript framework

Framework Type	Framework Example	URL
Desktop Like	GWT ExtJS	http://www.gwtproject.org/ https://www.sencha.com/products/extjs/
MVC with HTML templates, including data binding	Backbone Ampersand KnockoutJS CanJS Ember	http://backbonejs.org/ https://ampersandjs.com/ http://knockoutjs.com/ https://canjs.com/ http://emberjs.com/
HTML extension, data binding	Angular	https://angularjs.org/
Web Component like	Polymer	https://www.polymer-project.org
Plain library	JQuery	https://jquery.com/
Virtual DOM renderer (Including Flux architecture)	React Redux	https://facebook.github.io/react/ http://redux.js.org/
CSS and JavaScript utilities for UI	Bootstrap	http://getbootstrap.com/

3.3. React/Redux

React, when combined with Redux as the flux[§] library, creates a different way of thinking about UI development. The idea of React is that it will always render the state of your component. Redux manages that state of your application. The data in the state is immutable. It cannot change directly. When the program fires an action describing the change to the data, a new version of the state is created with the changes. Then React re-renders the parts that are different. Because React uses a virtual DOM, it can re-render very efficiently, leaving the code to concentrate on the presentation of the state. Therefore, that application goes from one state change to the next. As the state changes, the view is recreated based on the new state. Writing the application becomes more about managing and rendering the state. Figure 2 shows Redux/ Flux paradigm.

[*] https://code-cartoons.com/a-cartoon-guide-to-flux-6157355ab207#.tvoajeok3

[†]https://www.infoq.com/news/2014/05/facebook-mvc-flux

[‡] https://www.codementor.io/reactjs/tutorial/reactjs-vs-angular-js-performance-comparison-knockout

[§] http://blog.andrewray.me/flux-for-stupid-people/ or https://medium.com/brigade-engineering/what-is-the-flux-application-architecture-b57ebca85b9e#.h1qeqj7oj

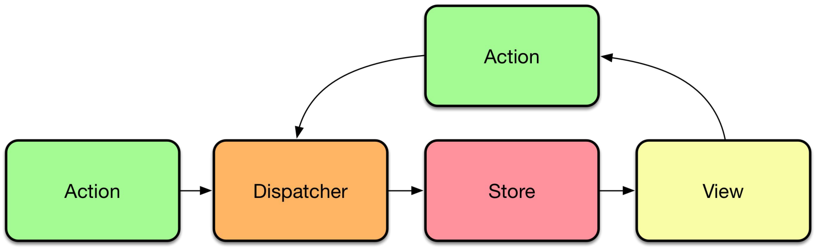

Figure 2. Redux defines actions, dispatchers, and the store. React defines the view.

4. FIREFLY HISTORY, FUTURE CONVERSION

4.1. What is Firefly?

Firefly is a front-end system designed for astronomical data archives (see Figure 3). It provides much of the UI/UX capabilities necessary to browse an astronomy data archive. It provides data viewing capabilities that are fairly advanced for a web-based system. It includes a full FITS[3] viewer, scatter plots, density plots, histograms plots, and high performance table browser. These capabilities can be customized for various archives (see Figure 4)

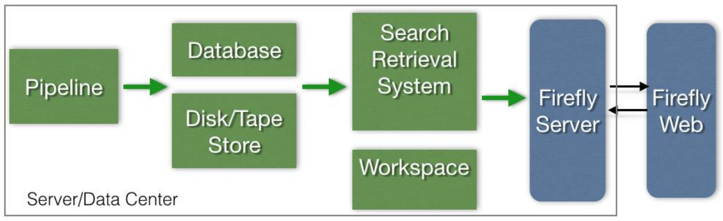

Figure 3. Firefly is a Web based Archive Front End system for the user to interact, evaluate and possibly download data

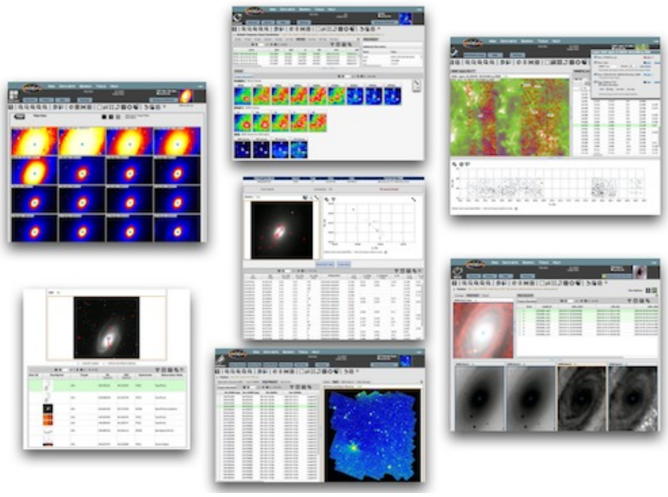

Figure 4. Firefly can have many different looks depending on the Archive system (http://irsa.ipac.caltech.edu).

4.2. Firefly History 2008 - 2014

Firefly development[4-10] was started in 2008 with the goal of providing advanced archive tools, more than was typical at the time. Plain JavaScript made it very difficult to write such a large, powerful web application. With the arrival of Google Maps in 2005 and Gmail soon after, developers quickly began to understand just how difficult is was to write at this scale. Every browser was different. JavaScript code had to be filled with conditions for each browser type. The understanding at the time was that JavaScript (ES3) was just not easily able to handle big applications.

In this large undertaking, we choose GWT. This allowed us to write in Java and have the code cross-compiled into multiple JavaScript instances, one for each of the browser types. This solved many problems and allowed us to eventually write 150,000 lines of client code. It effectively made JavaScript the assembly language for GWT (See Figure 5).

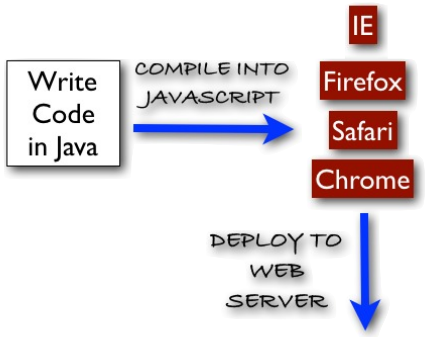

Figure 5. Basic concept behind GWT. Each compile of the Java code makes 4 versions of JavaScript code. This was great before the coming of 'Evergreen browsers' but now unnecessary. An 'Evergreen browser' is one that keeps itself updated. Chrome was the first.

4.3. Future Plan

Firefly should be flexible enough for web applications, JavaScript API or a remote API driven by a Python network interface. As a web application, Firefly will be a primary web source to query and visualize LSST data. However, as an API, Firefly might sit inside a Jupyter[*] notebook as a FITS or plotting visualizer. Firefly might also be used for an archive-in-a-box: the idea that given some data on a disk and a simple database, a Firefly web application could be up and running to serve it in a matter of hours. All these considerations require the design to be very flexible.

4.4. Web environment changes

Over the last several years there have been significant changes in the web environment (see Section 2 and 3). Support for GWT is fading. The JavaScript language has radically improved. JavaScript frameworks have mushroomed. In the current environment, GWT has become more of an anchor to us than an advantage. After researching the improvements in JavaScript and evaluating our current system it had become clear that we needed to convert Firefly to a more modern web development architecture. This has been a significant effort which hopefully will yield long term benefits.

4.5. Decision

JavaScript has matured into a very powerful language. To reach our decision to use React we considered the factors discussed in Section 3. To do this we wrote a little code in both Angular and React. Drawing on years of UI development experience it became clear that React approach to UI development was some of the most clean and cutting edge that we have seen. While we have a similar opinion about Redux, it took a little longer to reach that decision. Redux was in beta at the time. We ended up going though four different flux architectures before we settled on Redux. Looking back, it was worth it. Redux is the best flux paradigm available. Redux removes much of the unnecessary boilerplate complexity of other flux implementations. It uses the approach of having pure functional reducers that update pieces of a single store instead of having more the Object-Oriented type stores. This functional approach has set Redux apart from all the other implementations. With 18,500 stars on Github and almost universal praise in blog and articles[†], it appears that most of the developer community is converging on that opinion.

[*] https://ipython.org/notebook.html

[†] For example: https://github.com/xgrommx/awesome-redux, http://www.youhavetolearncomputers.com/blog/2015/9/15/a-conceptual-overview-of-redux-or-how-i-fell-in-love-with-a-javascript-state-container

We also chose other libraries which are listed in Table 6:

Table 6. Major JavaScript libraries used in Firefly

Major JavaScript Libraries	URL
React	https://facebook.github.io/react/
Redux	http://redux.js.org/
Babel	https://babeljs.io/
lodash	https://lodash.com/
Redux-saga	https://github.com/yelouafi/redux-saga

4.6. Training

Developers transitioning from Java to JavaScript had a steep learning curve. Though "C-like," the language itself is very different from Java. Java is designed on object oriented-programming. JavaScript brought some welcomed changes and new ideas to approaching problems. It is much more built around the idea of functional programming. Working with immutable data plays a significant role with this type of programming. These paradigms take some adjusting to.

JavaScript is a much tighter syntax, so programmers need to learn more crisp ways to attack problems. JavaScript object literals make for passing data very simple but take some training to use them effectively. Array processing in JavaScript is one of the most powerful features. Good JavaScript developers start seeing almost everything as an array or an array of objects. This a a huge change from Java in which everything is some sort of Class.

GitHub code review significantly helped the training process as more experienced developers were able to point out the best practices in JavaScript development.

5. LESSONS LEARNED

This is a very exciting time to be doing JavaScript development. Although the Firefly conversion has been demanding, we have learned a lot. Here are a few of the lessons:

5.1. OO programming is overused

Many, many developers are trained to do Object Oriented programming. The weakness of this approach becomes very apparent when you move to JavaScript. The language is very flexible and works much better if you are not trying to make everything a class. Beyond that, the functional approach of JavaScript makes for more stateless code. Functional, stateless code is easily testable, refactorable, and maintainable.[*]

5.2. Code reviews add value

Many new functional techniques are discussed during the code review process. Developers learn from each other and the product becomes better. Github helps facilitate code reviews with line comments and pull request. After the Firefly work started with LSST Data Management (DM)[11], we added code reviews based on the LSST approach. Firefly uses a code review process very similar to that used by LSST DM[†]. However, there are some slight differences because DM is more focused on Python and C++. Firefly does not have a JavaScript coding standard like the LSST project has for Python and C++. We do focus on good practice, understanding of the system, and effective use of library functions.

5.3. React/Redux is a completely different UI paradigm

It will take developer a while to get their head around the concepts of Redux/flux. While the diagram (figure 5) is simple, learning to think in Redux immutability will take a while for programmers to work out.

[*] For more on functional programming in JavaScript: https://medium.com/javascript-scene/the-two-pillars-of-javascript-pt-2-functional-programming-a63aa53a41a4#.t4a0aj5u8)

[†] http://developer.lsst.io

5.4. Documentation

JavaScript documentation is hard. The language is more flexible than the tools used to document it. JSDocs (the most common one) has a lot to be desired. There are not many examples. Even so, we find that we do document functions more because the parameters are not as clear as in Java and because of functional programming. Since a pure function is a complete unit, it is more natural to document. We are currently using JSDocs but we are still considering other options.

5.5. The JavaScript world changes fast, you need to keep up

In general, we find that we are reading blogs more and following others on Twitter to stay on top of the developments. This is a strength of the JavaScript world but you have to keep up.

5.6. JavaScript ES6 is a very powerful language

After years of Java, C, and C++ programming, some of us believe that the JavaScript language is exceptional. It is a truly productive and concise programming experience. Early on in the conversion, we were faced with the decision to write in ES6 and transpile. We have no regrets.

6. CONCLUSIONS

The Firefly conversion took the Java lines of code from 150,000 to about 70,000 in JavaScript. The conversion took our team of five to six people about seven hard-working months. The pace got faster in the final three months as developers became more comfortable with JavaScript. The open-source development environment proved very helpful. On the whole, our developers were very pleased with JavaScript ES6.

ACKNOWLEDGMENTS

We thank Tim Jenness for his extensive review and suggestions. Also, Ben Emmons has our thanks for close review of the JavaScript sections on the draft manuscript. This material is based upon work supported in part by the National Science Foundation through Cooperative Support Agreement (CSA) Award No. AST-1227061 under Governing Cooperative Agreement 1258333 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DEAC02-76SF00515 with the SLAC National Accelerator Laboratory. Additional LSST funding comes from private donations, grants to universities, and in-kind support from LSSTC Institutional Members.

This work was supported in part by IRSA, the NASA/IPAC Infrared Science Archive. IRSA curates the science products of NASA's infrared and submillimeter missions, including many large-area and all-sky surveys.

REFERENCES

1. Kahn, S., “Final design of the Large Synoptic Survey Telescope,” in [Ground-Based and Airborne Telescopes IV], Hall, H. J., Gilmozzi, R., and Marshall, H. K., eds., Proc. SPIE 9906, in press (2016).
2. Tacy A., Hanson R., Essington J, Tökke A, [GWT in Action], Manning Publications, Shelter Island, NY, (2013).
3. Pence W. Chiappetti L., Page C., Shaw R, Stobie E., "Definition of the Flexible Image Transport System (FITS), version 3.0”, Astronomy & Astrophysics, Vol 524, A42 (2010).
4. Roby, W. et al., “Archive Web Sites using AJAX & GWT”, ASP Conf Ser. 434, 21, (2009).
5. Roby, W. et al., “Writing Web 2.0 Applications for Science Archives”, Proc. SPIE 7740, (2010).
6. Wu, X. et al., “Spitzer Heritage Archive”, Proc. SPIE 7737, (2010).
7. Roby, W. et al., “Using Firefly Tools to Enhance Archive Web Pages”, ASP Conf Ser 475, 315, (2013).
8. Roby, W. et al., “Next Generation Search Interfaces”, ASP Conf Ser. 495, 417, (2015).
9. Goldina, T. et al., “Web based 2 d visualization with large data sets”, ASP Conf Ser 495, 137, (2015).
10. Wu, X. et al., “IPAC Firefly Development Roadmap”, ASP Conf. Ser. in press, ASP, San Francisco (2016).
11. Juric, M. et al., “The LSST Data Management System,” in [Astronomical Data Analysis Software & Systems XXV], Lorente, N. P. F. and Shortridge, K., eds., ASP Conf. Ser. in press, arXiv:1512.07914, ASP, San Francisco (2016).