AT&T Developer Program

Getting Started with Web Applications

Development Brief

Revision	**1.5**
Revision Date	**03/16/2010**

Legal Disclaimer

Revision History

Date	Revision	Description
02/26/10	1.0	Original
03/16-10	1.5	Copyedit

Table of Contents

1. Introduction

This development brief considers how Web applications can be used on mobile phones today and discusses some limitations and opportunities for using content from Web application servers as well as recommendations for effective development of mobile Web applications.

1.1 Audience

This document is intended for freshman developers of mobile content. Experienced AT&T Content Providers may also find this paper to be a handy reference for basic mobile Web application best practices.

Do not forward or share this document without prior authorization from AT&T.

1.2 Contact Information

E-mail any comments or questions regarding this document to developer.program@att.com. Please reference the title of this document in your e-mail.

1.3 Resources

The official AT&T Developer Program Web site offers tools and resources to help you develop great mobile applications. Highlights of the AT&T Developer Program Web site include:

- **Documents and specifications.** Download style guides, code samples, white papers, and other resources.
- **Go to market.** The AT&T Developer Program shows you how to take your consumer application from concept to successful product in five clear steps.
- **Platform and operating system support.** Get the tools and information you need to port your application to any phone platform or operating system.
- **Browse and download application information.** Learn what you need to know to make the most of browse and download technologies.
- **Security.** Get the latest on AT&T security policies to keep you and your customers safe.
- **Other resources.** The AT&T Developer Program maintains and updates links to device manufacturers, mobile industry resources, and other useful Web sites for the wireless developer community.

All Content Providers for AT&T are required to join the AT&T Developer Program. Participation in the program facilitates communication between AT&T technical teams and the Content Providers. For additional information, go to the AT&T Developer Program.

2. What are Web applications?

Web applications ("webapps") are just applications designed using Web technologies; e.g., HTML, CSS (Cascading Style Sheets), JavaScript, and the DOM (Document Object Model). That sounds pretty simple, because it is.

But since these Web technologies are used in many Web-based services, even in simple Web pages, webapps include some distinguishing characteristics.

- Webapps are based around HTML documents, which provide a presentation framework that is dynamically updated by JavaScript.
- Webapps use the development technique called “Ajax” for programmed interaction between client and server, via server application programming interfaces (APIs).
- Webapps process data for presentation or application-internal use.
- Webapps use a range of advanced client-side Web runtime features; e.g., persistent storage and client APIs for device features.
- Webapps use any of the above in ways that are automated; thus they do not require user input or control for ongoing execution of the application.

Also, to help distinguish webapps from “native” applications, the following distinctions can be made. Native applications are those developed using technologies for a specific mobile device, e.g. a programming language and development environment which results in a compiled application that is downloaded and installed on a mobile device. Webapps, in contrast, are developed using Web technologies; e.g., the semantic (processable) languages defined by the Worldwide Web Consortium (W3C). Native applications may use Web technologies (e.g., XML and other languages) to interact with application servers, but the native application executes directly within the runtime environment of the device and is responsible for all application support functions not provided by the device operating system. In contrast, webapps execute as interpreted applications within a Web runtime environment (browser or widget engine), and they benefit from the built-in application support provided by those Web-focused application environments. Webapps are thus *enabled* by the Web runtime environment they execute within, and this provides a key aspect of the advantage of developing Web applications.

Webapps typically have both client and server components. This paper focuses on the client side; i.e., webapps developed for use on mobile devices. As the server side of webapps is also a key area for developers to understand, its influences upon the choices in client side webapp design are described here, so developers can better appreciate the relationship between webapp clients and servers. Details of Web server technologies and application design will be addressed in future AT&T Developer Program briefs.

3. Why should I develop Web applications?

Webapps represent a *convergence* of the features of native applications and Web sites. Webapps won't replace native applications or Web browsing, but they can bring together the best characteristics of these environments and thereby provide a new, and, in many ways, simpler environment in which developers can access a broader array of content and APIs and target their applications for a broader set of devices.

While they should not be expected to be the best choice for every type of application, for both the native application developer and the Web site developer, webapps provide some distinct advantages:

- For native application developers, the advantages include:
  - Reach more devices: Webapps minimize dependencies upon specific devices/platforms, so developers can reach many more devices and avoid dependency upon a specific application store.
  - Easier development: The high-level application programming environment provided by JavaScript, and the presentation framework provided by HTML, CSS, and the DOM, enables developers to focus on application logic rather than detailed user interface programming and window/screen management.
  - Simplified APIs: As compared to native platform APIs and custom client/server protocols, the APIs provided to Web applications tend to be simpler, and thus easier to integrate into applications.
- For Web site developers, the advantages include:
  - Simpler user experience: As compared to Web sites that require user action for each change in the presented data, webapps offer users applications that can automatically update. This is especially important in mobile applications due to the typical user input limitations of mobile devices.
  - Specialized applications: Webapps offer users much more than static presentation of content, and this enables dynamic and interactive applications that can be targeted to fulfill specific purposes and provide specific user experiences.
  - Access to device APIs: Webapps can access device data and native applications outside the browser sandbox, which enables client-side mashup applications and, overall, a much more integrated and personalized user experience.

Webapp design tools are well known, easy to use, and freely available from various open-source communities and vendors. Thus, getting started with Web application development is easy and inexpensive. Some examples of freely available Web application design tools include:

- Eclipse
- NetBeans
- Aptana Studio
- Major browser vendors commonly provide JavaScript design/debug tools as part of their products, e.g. Firebug for use with Firefox
- For those who like to go simple and use just a plain text editor, JSLint provides a JavaScript syntax checker

There are also active Web sites providing helpful information for the webapp developer. These sites can help you avoid interoperability issues and decrease the time between your initial idea and deployment of a working webapp. Some examples of webapp developer sites include:

- W3 Schools
- JavaScript Kit

With well-chosen techniques and use of available developer resources, development of sophisticated, first-class applications (i.e., desktop/native comparable) is possible. With HTML5, Web technologies are advancing toward support of first-class client applications that can run on almost any device. With HTML5 support, Web applications will support many of the key features users have come to expect from desktop applications, e.g. document editing. HTML5 will also provide a much more functional Web runtime environment for applications, including various application programming interfaces (APIs) for client-side application support and networking. Although it may be a while before full HTML5 support exists in most mobile browsers, there is already support for some key features and APIs in the most advanced browsers, e.g. in smartphones.

There is a rich and continually expanding “programmable Web” environment of Web service APIs and client APIs. Programmability evolves the Web beyond the set of static Web pages that typified the Web’s first decade. Client and server APIs enable distributed Web applications that can leverage network-based services and client data outside the browser “sandbox.” The rapid expansion and evolution of server APIs and Web application frameworks is illustrated by sites such as Programmable Web and Wikipedia’s “Comparison of web application frameworks.” Through these Web service APIs and frameworks, very sophisticated webapps can be developed that utilize client-server and server-server interaction to integrate diverse content sources and services.

Development of device APIs for Web runtime environments is also greatly increasing the richness of client-side webapps. The OMTP BONDI project is a leading example of collaborative, open-source development of APIs for browser/widget-based webapp access to a variety of useful device features. AT&T, through its support of the OMTP and the W3C's Device API and Policy (DAP) standards, is helping to standardize these APIs for use across any device type, while enabling developers to address key considerations of their use in particular devices, e.g. mobile phones.

4. What do I need to know to develop Web Applications?

To decide what you can and might want to do through webapps, you need to know what type of basic design decisions you would need to make and how those decisions affect your development effort. Here are some key points, explored further in the following sections.

You should have a good working understanding of the basics of webapp design, including HTML, CSS (cascading style sheets), JavaScript, and the DOM (Document Object Model).

You should understand the pros and cons of depending solely upon your own code or using code that is freely available in JavaScript frameworks.

If you want to target a variety of devices and Web runtime environments, you should be prepared to address the same multi-platform design issues as Web site developers face.

You need to understand why webapps, based upon the distributed resource concepts of the Web, require an asynchronous, event-driven design model.

You need to be aware of the relationship between webapp clients and servers and the various data exchange approaches that you may encounter in using server APIs.

Since webapps are based around the concept of a dynamic HTML presentation environment and often involve various types of data processing, you should be familiar with the issues and approaches to handling HTML, XML, and JSON (JavaScript Object Notation) data.

As one of the key device features you can take advantage of is persistent data storage, you should be aware of the various APIs supporting data storage and the related approaches for handling the opportunities and issues of persistently storing webapp data.

You should be aware of the security risks to webapps and be prepared to take appropriate measures to protect the security and privacy of your webapp users.

You should be aware of the advanced features and APIs that are just becoming available on desktops and mobile devices, e.g. through HTML5 and OMTP's BONDI. You can be prepared for the widespread availability of these features if you design webapps that use them today, which you can do by using developer tools and devices that already support these features and APIs.

4.1 Webapps are Applications

It may seem obvious, but webapps are fundamentally applications, and they are quite different from Web sites. You should approach the design of a webapp, as you would any application, using structured design techniques that enable you to keep your application simple, modular, and efficient. A design methodology often used with Web applications is called “model-view-controller,” from the approach of structuring applications into modules for working with data resources, user interfaces, and the application logic that controls the overall flow of the application.

You can address some of the design complexities of designing advanced webapps through the use of JavaScript frameworks, which can simplify your design by “abstracting away” some of the application structure considerations.

4.2 JavaScript Frameworks

One of the key decisions to make is whether to use a JavaScript framework to simplify webapp programming tasks, which can be complex. A variety of freely available JavaScript frameworks are in wide use, e.g. jQuery, Prototype, dojo, YUI, MooTools, etc. These frameworks can simplify a wide range of webapp programming tasks. However there are caveats to their use, including the need to package the framework libraries with webapps, and the need to recode webapps if changing frameworks.

Because the libraries can be significant in size (even when “minified”), they can result in significant network usage, as they are re-downloaded every time the webapp is used. Widgets can avoid that caveat, as the framework can be packaged with the widget for installation. If you are just getting started with webapps, you may benefit from the experience of building your own library of JavaScript utility functions. This will enable you to learn the reasons why frameworks exist, and, perhaps, help you minimize dependence upon the frameworks.

4.3 Multi-platform Design

Two key decisions you will need to make are how many devices you want to target and whether you want to deploy your webapp for use in browsers or as a downloadable widget.

One of the key motivators for developing webapps is the opportunity to develop applications that can run on a very broad set of devices, with little or no device-specific customization. In contrast, native application developers typically don’t expect their applications to be usable on different devices, without at least recompilation and, most likely, some porting effort. The Web, however, is based upon the inherent assumption that Web content standards (including the processing model for that content) should minimize the variation in different devices for Web based services. Nonetheless, Web developers are familiar with the realistic limitations of browser interoperability.

Multi-platform design, meaning designing for consistent support by different device types and Web runtime environments, is just as much a challenge for webapp developers as Web site developers. This is because webapps are based upon the same Web runtime environments that support browsers and widgets. So while the goal of interoperability is clear, and in many cases achieved, webapp developers will benefit by adopting a variety of best practices, and, where necessary or advantageous, by seeking the support of JavaScript frameworks to simplify webapp design.

Some basic best practices for improving multi-platform support include:

- Keep your webapp simple, well-structured, and refactor it often to improve structure and performance
- Focus on consistently supported features, especially for the HTML DOM and in user interface elements
- Prepare to support alternate methods for APIs you intend to use
- Use exceptions where necessary to implicitly detect incompatibilities and trigger alternate feature handling
- Avoid depending upon explicit detection of the browser type, except as a last resort
- As a fallback approach when no alternate is available for a feature, gracefully degrade the user experience if necessary and acceptable
- Keep user interface elements simple, or use JavaScript frameworks for user interface aspects (this is one area where they provide a lot of value, especially for the desktop, but their advantages for mobile devices may vary)
- Use development tools and actual devices to test
- Test early and often, across as many different platforms as you can
- Build design-for-testability into your webapp, including internal event/exception logging (JavaScript online debugging can be very difficult and impossible on some platforms)

4.3.1 Know What to Expect

With just a little bit of research, you can learn what to expect to be supported by the major browsers. Sites such as the following provide very useful tutorials, technical implementation details, and interoperability information:

- W3 Schools
- JavaScript Kit
- Quirksmode

Key areas to watch out for interoperability issues include:

- DOM XML and HTML document parsing (“DOM traversal”)
- Support for standard DOM objects and methods
- Browser-specific DOM objects and methods
- HTML5 features: while some HTML5 features are currently supported, it will probably take until 2012 for there to be widespread and consistent support for most HTML5 features

4.3.2 The Widget Alternative

Designing your webapp for distribution as a widget can simplify some of these interoperability issues. Widgets can be targeted for particular widget environments, such as AT&T’s Plusmo Widget Engine, which can provide a consistent user experience regardless of the host device. Even so, you can still use the same webapp standards to develop your application for use in either browsers or widget engines and thereby avoid the need to maintain multiple versions of a webapp.

Let’s look a little closer at the differences between webapps in the browser and as widgets. The major difference between the browser and widget context is that, for use as widgets, webapps are packaged, downloaded, and installed on your device, rather than being accessed like normal Web pages.

Standards-based widgets will soon be supported in most mobile devices. The W3C has completed the definition of the widget packaging standards, and it continues to develop APIs and other standards that support widgets and webapps in general. The BONDI standard APIs have greatly expanded the set of device features available to webapps and have contributed in large part to the ongoing standardization work in W3C.

You gain various advantages by developing your webapps to run as widgets. First, as noted before, you can target your widgets for a particular widget player, such as the AT&T Plusmo Widget Engine. This can provide a more consistent environment for your webapps, even in different devices. Widget engines also typically have better support for device features such as BONDI APIs, and overall they typically have a more complete set of APIs similar to native application environments. This API set, and the ability of widgets to go beyond same-origin restrictions, enables a rich environment of content sources for the creation of client-side mashups.

Widgets can avoid the same-origin limitations of the Web security model imposed by browsers by disclosing the server domains to be accessed in their configuration file. This is because, like other downloaded applications, widgets can be tested and pre-approved through a developer program, which enables well-designed webapps to have more freedom in their content and service sources. That same pre-approval process can offer the webapp developer access to device APIs without security prompts, which can result in a better user experience. This is enabled by signing the widget package, similar to how Java applications are assigned various trust levels by AT&T.

4.4 Thinking Asynchronously

You should be prepared to design for the unique characteristics of the webapp programming model, or you may find that as you integrate new device and network resources into your application, and present it through the HTML document framework, the resulting application may be unreliable or difficult to use. One of these key characteristics is asynchronicity, i.e. dependence upon an inherently asynchronous, event-driven design approach. For native application programmers experienced with a synchronous function call design model, this takes some getting used to.

The key reason for the asynchronicity of webapps is that many APIs depend upon access to resources outside the Web runtime (e.g. a Web server, device native function, or hardware component response). The response to the API request may take some time, and to avoid blocking the Web runtime (and user interface), the responses to API requests are typically returned as events to some object or explicitly indicated callback function. This results in the need for event-driven application logic that can handle a potentially unordered sequence of events from asynchronous API requests, network data, and user input. This will help ensure that such simultaneous operations and events have a reliable effect upon data and the user interface.

For Web site developers, this can also present a challenge because it differs from the conventional model of Web user interaction (user action, request to server, response from server, content presentation). The conventional model results in a more or less static Web page being presented, with little or nothing occurring until the next user action. In contrast, several things can be occurring at once in webapps: a user can be interacting with the presented user interface elements, a server request can be outstanding, a local device API request can be outstanding, and some other webapp (of the same origin) can be modifying data relevant to the first webapp and triggering events.

A related key consideration is to ensure that regardless of the variety of events that may be occurring, the user should see a sensible, consistent content presentation and never lose control of the webapp. Thus the user interface aspects of the webapp should be mediated by logic (e.g. a “controller” in the model-view-controller design pattern) that determines which events should update the user interface and when.

4.5 Working with Web Application Servers

True to the Web’s origin as a framework of interconnected, distributed content, most webapps will include a server component. The only exception to this is in the special case of widgets that are designed to use only device-local data and APIs. The requirement for a webapp server need not be complex, however; webapp servers can be as simple as a normal Web server available from any Web hosting service. For example, if you are just providing a Web page with associated scripts and server-hosted data, these can be served upon request to a browser from any Web server. Webapp servers can also provide complex services, e.g. providing Ajax-callable APIs and acting as a bridge to diverse Web service APIs as part of a server-side content mashup. Various webapp server development frameworks can simplify the development of such complex webapp server components, but for these, a more specialized and specifically configured Web server environment may be required.

One of the key reasons that webapp servers are used to provide a bridge to APIs is due to the Web security model. If you plan to use server APIs or data, you should know that except for widgets, which have special security requirements, webapps can only access Ajax-based APIs provided by the same “origin” (combination of host domain, protocol, and port number) from which the webapp was downloaded. Widgets can access other servers only if they are declared in the widget manifest document (“config.xml” file). But browser-based webapps cannot directly interact with any server but the same-origin server. Note that support for “cross-origin resource sharing” is being standardized in W3C, but when this work is done it may still take a considerable amount of time to deploy updated Web server support for these standards.

Your webapp can load content from other servers indirectly, for example by creating an iframe and setting its source to the URL you want to load. In that case, however, your webapp is not directly involved in retrieval or processing of the content (the Web runtime retrieval and presentation, as with any normally referenced content), and it will not be able to access or manipulate the content in that iframe, since JavaScript is still prevented from accessing documents obtained from a different origin. These limitations are being addressed through HTML5, which will provide you with additional methods of securely accessing content from different origins.

4.5.1 Web Application Server API and Data Handling

If you decide a webapp server component is required for your purposes, you will also need to consider the types of data and/or APIs the server should provide. Server APIs are as numerous and distinct as applications themselves, and they can be implemented using a variety of techniques and data types. Some key considerations are given here.

If you are just retrieving and presenting data on the client side, e.g. just embedding server content into your application's HTML, you can create HTML elements that just reference the server content, and you can pass any needed API parameters as URL fields in the "REST" style. For automated retrieval, you can also generate such simple content requests using the XHR API (XmlHttpRequest, the core of the "Ajax" design method) and then apply the returned content to the HTML element's "innerHTML" DOM property.

Use of XHR is one of the basic webapp design techniques. Due to Web runtime variations for the XHR API, a JavaScript framework or user-provided XHR utility function will likely be required. You should also be aware that special techniques (e.g. queuing) may be required due to Web runtime limitations on the number of outstanding requests and the complexities of dealing with multiple outstanding asynchronous requests (e.g. see this article for a good description of the problem and a solution for XHR+XML). JavaScript frameworks often address these issues, e.g. as in YUI's Connection Manager.

If you need to create or process data exchanged with a server, you will need to determine which data format best suits the type of webapp. Commonly used, basic content types include HTML, XML, and JSON.

JSON is often preferred for APIs due to its simplicity of processing, since JSON data is exchanged as JavaScript code strings which can be directly parsed into JavaScript objects and vice versa. A key consideration for use of JSON data is security, because it's possible for malicious code to be executed if untrusted JSON data is non-securely parsed. For that reason, JSON libraries (e.g. json2.js and general JavaScript frameworks) commonly support JSON "stringify" and secure parsing functions. The stringified JSON data can then be sent or received in the body of XHR.

HTML and XML can also be exchanged in the body of XHR. Similar to JSON, exchanged HTML and XML data must be converted between document objects and strings (“serialized” and parsed). Browser support for XML serialization and parsing support is generally good, although there are variations that require special processing (Mozilla recommends Sarissa, a “cross-browser wrapper for native XML API”). A good introduction to the overall use of XML with JavaScript is provided by webreference (an excerpt from “JavaScript: The Definitive Guide, Fifth Edition,” published by O'Reilly Media, Inc.).

While HTML form submission is a basic browser function, direct exchange of HTML content through XHR can be a bit trickier. To create a request body or URL query string, HTML form serialization is supported by various JavaScript frameworks. Parsing server content in HTML form should be as easy as loading the XHR response body into an HTML document element and then using DOM functions to access the HTML elements.

4.5.2 Efficient Use of Device and Network Resources

A key consideration in your webapp design should be the effectiveness and efficiency in the use of device and network resources. Programmatic access to server-based data is one of the key opportunities driving the richness of Web applications, but it needs to be carefully managed. Over-use of XHR or dynamic HTML that results in retrieval of server-based content can put a strain on a variety of resources, including the device battery, data network, and application servers. This can cause a poor user experience and result in the removal of your webapp, e.g. if the user finds that your webapp is causing substantial data cost or is draining the device battery due to excessive data traffic.

The most basic principle to keep in mind is that every resource you use has an impact and needs to be associated with some specific value to the user. You need to consider that value carefully and use effective techniques to manage resource use. In addition, you should allow the user some control in how your webapp uses resources, or at least provide a simple and accessible way for the user to learn what your webapp will do and how it affects device and network resources.

4.6 Working with XML and HTML Documents

As standardized languages, both XML and HTML are well supported by JavaScript and the DOM, although there are differences in browser implementations that limit interoperability of the standard DOM functions. For examples, see the w3schools listings of HTML DOM function support across major browsers. These variations (e.g. namespace support) mean that supporting all or even most browsers with exactly the same code is unlikely. To handle the differences, you will need to either rely upon a JavaScript framework, or build your own utility functions with special code where necessary.

For specialized data schemas or reliably-structured document types of which only a few elements are important, you can use relatively simple/direct methods such as parsing a document using your own code, based upon the basic DOM methods (for HTML and XML).

HTML and XML documents can be created and updated using the same JavaScript DOM functions. Similar to accessing the documents via the DOM, some variations should be expected (e.g. with XML namespaces), which require special handling.

4.7 Data Storage

JavaScript's support for data objects enables webapp developers to process and store significant amounts of string data while the webapp is running. This basic ability to use structured objects, arrays, and variables is a powerful tool for the webapp developer. However, without the ability to store data that has been downloaded and possibly modified by the webapp, the data must be uploaded to a webapp server, or it will be lost when the webapp exits. "Exits," in this case, means, for widgets, that the widget is stopped, or, for browser webapps, that the user (or webapp itself) navigates away from the webapp origin.

4.7.1 Persistent Data Features Available Today

A key near-term opportunity for webapp development is the ability to store data that persists within a session or between sessions, and to share that data between webapps of the same origin. While initially limited to proprietary solutions (e.g. Google Gears), data storage standards are beginning to emerge, and support is widening for webapps running on desktops and mobile devices. Developers can begin to gain experience with persistent data capable webapps today, learn what this powerful new feature can offer to their applications, and discover the new webapp design considerations it brings.

The HTML5 Web Storage API standard, nearing completion in W3C, is already supported by some desktop browsers (Safari, Firefox, IE8) and mobile browsers (Safari on iPhone). This API provides storage of name-value string pairs that live within the context of a browser session (sessionStorage) or that persist between sessions (localStorage). Most desktop browsers should support the Web Storage API in 2010, and more high-end browsers on mobile devices will begin to support it as well. W3C is also working to standardize other persistent storage APIs, such as structured data storage and a file API focused on user-selected file read access.

The Web Storage API focuses on data storage inside the Web runtime, as compared to the BONDI File System API, which allows trusted webapps to access local filesystem data on devices that support a filesystem. The BONDI File System API enables applications to directly read and write filesystem data, much like the support in native programming languages. BONDI has also defined the Application Configuration API, which addresses the same basic objective of HTML5 localStorage and is intended for use with Web runtime environments that do not support the Web Storage API. BONDI-compliant devices are expected to be available in 2010.

4.7.2 Effective Use of Persistent Storage

The storage of data through the HTML5 or BONDI APIs involves similar issues and techniques as server data exchange. JavaScript data and HTML/XML documents must be stringified before being stored, and they must be re-parsed when reloaded.

A new consideration for many Web developers – that of being responsible for handling data schema updates as the webapp evolves – will be a side-effect of using persistent data. If a new version of a webapp includes changes to persistent data structures, the webapp must be designed to take this into account and provide a means to migrate current stored data if possible. In order to detect that such a situation has occurred, a data versioning scheme or other automated detection method must be used.

As a fallback to data migration, a means to clear local persistent data can be provided. This option is also recommended to allow users to release storage resources of the webapp, which may not be easily or automatically removed otherwise. For example, HTML5 localStorage data may not be removable by browser menus (except through clearing all stored data, which may be undesirable), and file system stored data (e.g. files and directories outside the widget home directory) may not be automatically removed when a widget is uninstalled (and for browser-based webapps, there is no “uninstall” procedure). Since HTML5 stored data is associated with the webapp’s source origin, it is also recommended that if removing all HTML5 stored data, the webapp should delete its own data directly, rather than via “localStorage.clear(),” to avoid deleting data of other webapps of the same origin.

4.8 Security

Like most things that are easy, webapps carry risks, and one is security. The nature of the scriptable Web environment is such that malicious code can easily masquerade as data, and, therefore, unsecured APIs may provide easily exploitable vulnerabilities as well as a ready platform for a wide variety of security attacks (such as cross-site scripting, cross-site request forgery, and click-jacking).

While the same-origin policy is intended to minimize the risks from these common attacks, you need to use specific measures to ensure your webapp keeps user data secure and is not exploited as a platform for attacks.

Because your webapp may obtain data from a variety of sources, especially if accessed through a webapp server, you need to be careful of security threats from that data. An example is in how you handle JSON data. Since it can represent any valid JavaScript code, you should always use secure methods of parsing JSON data, and never use the “eval()” operation. There are various JavaScript frameworks that can help you, and native JSON parsing is beginning to be supported by browsers.

Device APIs, as one of the most valuable new features of webapp client environments, further pose risks due to the sensitivity of the information that the APIs can provide. Uncontrolled access of these APIs can result in seriously compromised privacy and security as well as excessive costs. Thus, while AT&T is still developing its policies relating to webapp and API security, you can expect that similar to AT&T’s current policy for Java applications, access to some APIs will be provided only to applications that have been certified as safe, through the AT&T Developer Program. The goal of these security considerations is to improve the user experience by reducing or eliminating security prompts, while allowing application access to the device APIs that enable a personalized user experience.

4.9 Advanced Features

One of the newest areas of Web runtime features is in client-side resources, i.e. data and device APIs. Many useful APIs are already available in desktop and mobile Web runtime environments, and others are available for testing in software development kits (SDKs) and in reference implementations, as described below. You can start taking advantage of these APIs now, not only to learn what they can offer for your webapps, but also to gain experience with effective API use.

4.9.1 HTML5 APIs

If your webapp is focused purely on processing/presenting server data and user interaction, the webapp is probably fine with sticking to what's available within the browser sandbox. Even within the browser, HTML5-capable Web runtimes will support advanced APIs that are very useful in creating webapps, including the Web Storage API described above, and also the following:

- Web Workers: This API "allows Web application authors to spawn background workers running scripts in parallel to their main page. This allows for thread-like operation with message-passing as the coordination mechanism." Webapps can thus be structured, with foreground windows working with distinct or shared background scripts for processing, e.g. data processing or server API handling.

- Server-Sent Events: This API supports "opening an HTTP connection for receiving push notifications from a server in the form of DOM events." This can significantly reduce data traffic for pushed data, e.g. in comparison to polling techniques. Note that the webapp server will need explicit support for the API, which may require special server code or configuration.

The W3C standards for the HTML5 APIs above are nearing finalization, which means they are likely to be supported by Web runtime environments soon. The W3C is additionally working to standardize a variety of other useful APIs, some of which may be available as early as 2010.

4.9.2 OMTP BONDI

As introduced above, the OMTP BONDI project is an open-source development of APIs for browser/widget-based webapp access to a variety of useful device features. AT&T will be launching BONDI-compliant devices in the near future and is actively supporting the OMTP development of these APIs, including the creation of SDKs and reference implementations that developers can use today (see Development and Test Tools).

The latest version of the BONDI API specifications (BONDI 1.1) includes the following API modules:

- Messaging: sending, receiving, and watching for SMS, e-mail, and MMS messages
- File System: accessing the filesystem to create and read directories and files and to write to files

- Media Gallery: accessing a metadata-enhanced repository of media content
- Geolocation: accessing location services
- Camera: taking a picture or recording a video
- Telephony: accessing the log of incoming calls
- Personal Information Management
    - Contacts: accessing the device’s native address book data
    - Calendar: accessing the device’s native calendar data
    - Tasks: accessing the device’s native task (to-do list) data
- Device Status: accessing various device properties and status, e.g. battery level, memory size/usage, etc.
- User Interaction: adding widget menus and being informed of changes in the window modes (e.g. display orientation)
- Application Launching: launching native applications by their associated URI scheme (e.g. http, https, tel, sms, mailto, file)
- Application Configuration: storing webapp key-value string data persistently

Note that W3C Device API and Policy (DAP) is also working on similar APIs for generic devices and is focused on W3C-typical browser use cases, e.g. in which API invocation is user-directed. For example, the DAP camera API may be associated with an HTML <input> element. The BONDI APIs are expected to continue development as a more programmatic model for API access, i.e. one in which user involvement is not assumed due to the user interface limitations of constrained devices (e.g. mobile). The DAP API specifications are expected to be complete in the 2011 timeframe.

4.10 Development and Testing Tools

In bringing all this together, you should leverage tools that simplify your development tasks and support any special requirements of the webapps you are focusing on.

The BONDI reference implementation (RI) is available for Windows Mobile 6.1 and up devices. Developers with access to touchscreen-based Windows Mobile devices can load this RI, which includes BONDI support as a standalone Widget manager, and as an ActiveX extension to Windows Mobile Internet Explorer. A variety of sample widgets are also available from the BONDI Widget Gallery.

AT&T will also be launching its own Widget SDK, which is based upon the Plusmo widget runtime, by June 2010. As described on the AT&T Developer Program Web site, you can also access test devices through the DeviceAnywhere Service: "*Through DeviceAnywhere's original non-simulated, real-time platform, you can remotely press buttons, view LCD displays, listen to ringtones, and play videos just as if you were holding the device in your hands.*" DeviceAnywhere can be used to test the compatibility of your webapps with various devices, or you can load the BONDI RI to gain experience with BONDI on real devices.

Integrated Development Environments (IDE) can provide a convenient code and test toolkit framework that can help you accelerate your webapp design. Such tools can integrate SDKs with test harness capability, such as the LiMo BONDI SDK (a plug-in for the Eclipse IDE). Using these tools, you can closely approach the target runtime environment and reduce device-specific testing time. However, if you do intend to target a range of browsers or widget engines with your webapps, nothing will replace testing on actual devices or by using device-consistent emulators, which are usually available through the major Web runtime vendor developer programs.

As referenced on the BONDI Developer Tools page:

- LiMo BONDI SDK: *The BONDI SDK project is an effort to develop and maintain an open source Web SDK based on the BONDI specifications. The objective of this project is to create an SDK that enables development of Web applications and widgets that can run across all mobile handsets supporting a BONDI-compliant Web runtime. The BONDI SDK project is being sponsored by the LiMo Foundation.*
- LG SDK 1.03 for Mobile Widgets: *The LG SDK for Mobile Widgets allows developers to create widgets for LG mobile phones quickly and easily. The LG SDK for Mobile Widgets is based on the World Wide Web Consortium (W3C) Widgets 1.0 specifications, supports a subset of the Open Mobile Terminal Platform (OMTP) BONDI 1.0 Candidate Release, and includes some LG specific APIs.*
- Perfecto Mobile testing service: *This service has a number of Windows Mobile devices running the BONDI reference implementation. You can use this service to test your widgets online*

and see how they work on a range of devices on different operators' networks.

For links to these resources, visit the AT&T Developer Program, which provides continually expanding support for webapp developers.