Using AJAX in Web Applications

A Practical Guide for ASP.NET Developers

January 16, 2005

1. Overview

2005 will definitely be remembered as the rise of AJAX – the new development technique that many believe will blur the line between web-based and desktop applications. This mystical acronym, authored by Adaptive Path in mid February, is a label for the rich, highly responsive and interactive interfaces of AJAX-enabled applications. It stands for “Asynchronous JavaScript + XML”.

Although we are just beginning to realize its full potential, the proven success of famous AJAX-based projects like Google Maps signifies that this is not the just another media hype, but rather a promising technology that may change web-applications as we know them.

The purpose of this article is to help developers understand the core concept of AJAX, realize its benefits and suitable application scenarios, and of course, become aware of its drawbacks.

You will learn about some of the most important AJAX implementations as well as about some of the leading development tools and components, which can help you jump-start your AJAX-enabled applications.

This document is aimed at ASP.NET developers, although anyone with a good knowledge of how the Internet works will gain a solid understanding of AJAX and its benefits.

2. What is AJAX – a Mere Buzzword or the Key to High-Performance Web Applications?

The pursuit of a development technique like AJAX came from the need for making web applications much more usable and eliminating their key disadvantages in comparison with the desktop platform:

- **Poor Interactivity** – web applications require that users wait for full page reloads after each interaction with the server. During the loading time they have to stare at a blank screen, which tremendously disturbs the whole experience. Although broadband internet connections are becoming a standard, web applications are also becoming increasingly complex and "heavy" so the overall waiting time remains relatively the same.

- **Unresponsiveness** – classic web applications transfer the complete form data to the server, which in turn renders and sends back the full HTML markup of the page to the browser. This happens during each postback and in most cases is highly inefficient, since only a small part of the interface is actually changed. However, lots of bandwidth is consumed and the performance is significantly hindered. This leaves users with the idea that web applications are slow by nature. Even worse, the user will often find the page has scrolled to a different position, causing disorientation.

- **Simplistic Interfaces** – the requirement for full page postback whenever the user interface has to be changed imposes hefty limitations on the degree of sophistication of web user interfaces. Rich and smooth interfaces with on-demand update could only be implemented using Flash technology. This approach, however, is impractical for general use since it is very complex and requires a much different set of skills than those possessed by the typical web developer. It can also cause end-user issues as a plug-in is often required.

- **Low Usability** – if a web application reloads the whole page because the user made a new selection on a form, they will get confused. It is often the case that web applications work in a confusing and esoteric way because the web application has been built around the standard, simple view of the Internet protocols. ASP.NET meant we could build applications with more functionality more quickly, usability has a way to go yet.

AJAX was born with the idea to change all this and narrow the functional gap between the desktop and the web. The new generation of AJAX-enabled applications delivers close-toinstantaneous performance, rich interfaces and tremendously improved user experience. It opens new horizons for much closer interaction with the application and demonstrates in practice what was until recently considered impossible:

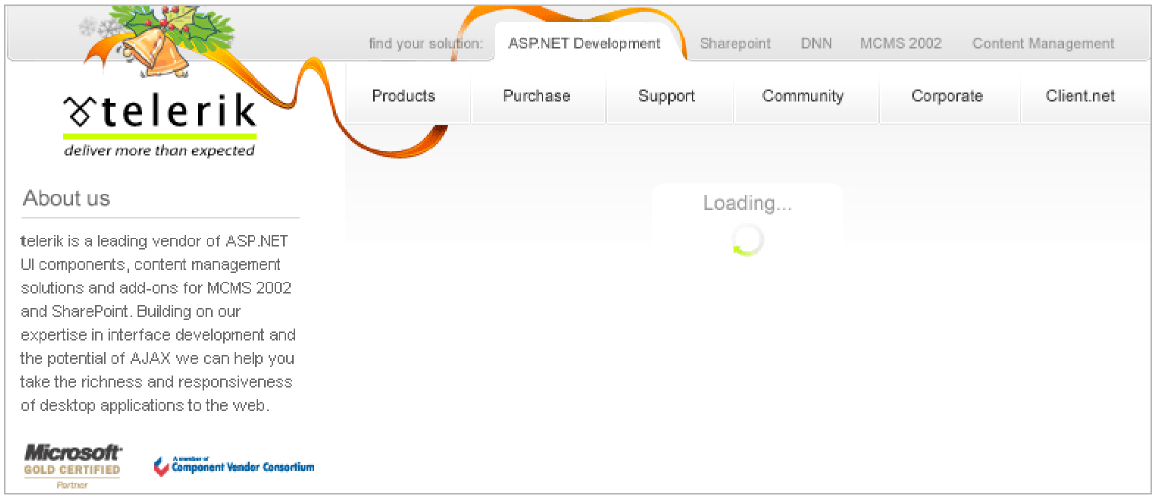
- Real-time map panning in Google Maps and Virtual Earth is just like image panning in Adobe® Photoshop®
- Folder browsing, message previewing, etc. in Microsoft® Outlook® Web Access is identical to that in the desktop version of Outlook.
- Validation checking on complex input fields can be performed by the server, without reloading the page.
- Virtual scrolling of huge tables with telerik r.a.d.**grid** is as fast as in Microsoft Excel®

What's interesting to know is that AJAX is not actually that new as a technology. It has been first used after Microsoft implemented `Microsoft.XMLHTTP` COM object that was part of The Microsoft® XML Parser distributive. As an ActiveX object in Internet Explorer 5, it was used to create the famous Outlook Web Access. You have probably seen AJAX in action for quite long in the MSDN Documentation treeview navigation. What is new actually is the name AJAX, which was widely accepted in 2005. Other labels for the same technology are Load on Demand, Asynchronous Requests, Callbacks, Out-of-band Calls, etc.

What's even more interesting is that AJAX is actually not a technology. It is more like a development technique that utilizes in a unique way a number of already mature technologies: HTML/XHTML, XML, DHTML, the XmlHttpRequest object, and JavaScript. For the purposes of simplicity we will refer to it as technology as it is widely accepted as such and provides a useful language to discuss the characteristics of the significant trend it represents.

3. How Does AJAX Work?

The core idea behind AJAX is to make the communication with the server asynchronous, so that data is transferred and processed in the background. As a result the user can continue working on the other parts of the page without interruption. In an AJAX-enabled application only the relevant page elements are updated, only when this is necessary.



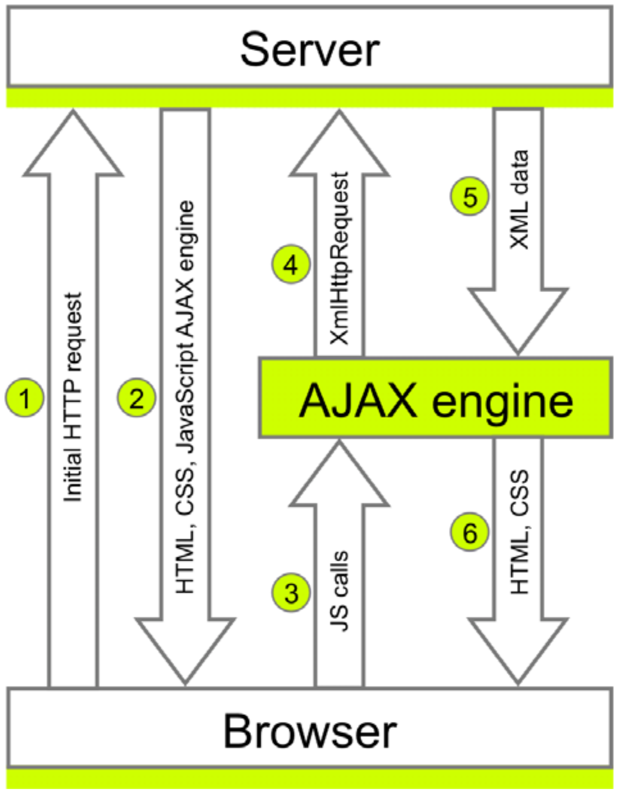
telerik is a leading vendor of ASP.NET UI components, content management solutions and add-ons for MCMS 2002 and SharePoint. Building on our expertise in interface development and the potential of AJAX we can help you take the richness and responsiveness of desktop applications to the web.

In contrast, the traditional synchronous (postback-based) communication would require a full page reload every time data has to be transferred to/from the server. This leads to the following negative effects:

- The user interaction with the application is interrupted every time a server call is needed, since a postback has to be made.
- The user has to wait and look at blank screen during each postback.
- The full page is being rendered and transferred to the client after each postback, which is time consuming and traffic intensive.
- Any information entered by the user will be submitted to the server, perhaps prematurely.

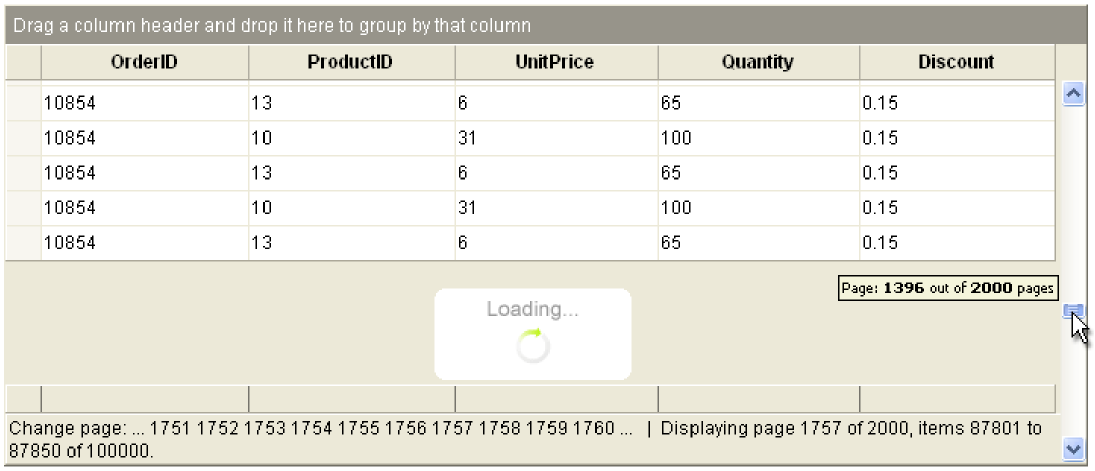
The AJAX-enabled applications, on the other hand, rely on a new asynchronous method of communication between the client and the server. It is implemented as a JavaScript engine that is loaded on the client during the initial page load. From there on, this engine serves as a mediator that sends only relevant data to the server as XML and subsequently processes server response to update the relevant page elements.

Below is a diagram of the complete lifecycle of an AJAX-enabled web form.



1. *Initial request by the browser – the user requests the particular URL.*
2. *The complete page is rendered by the server (along with the JavaScript AJAX engine) and sent to the client (HTML, CSS, JavaScript AJAX engine).*
3. *All subsequent requests to the server are initiated as function calls to the JavaScript engine.*
4. *The JavaScript engine then makes an XmlHttpRequest to the server.*
5. *The server processes the request and sends a response in XML format to the client (XML document). It contains the data only of the page elements that need to be changed. In most cases this data comprises just a fraction of the total page markup.*
6. *The AJAX engine processes the server response, updates the relevant page content or performs another operation with the new data received from the server. (HTML + CSS)*

Example: The AJAX-based VirtualScrolling feature of telerik r.a.d.**grid** loads on demand only the visible page of the grid, which makes possible to browse of 100,000+ records almost in real-time. (See live demo)



Drag a column header and drop it here to group by that column

OrderID	ProductID	UnitPrice	Quantity	Discount
10854	13	6	65	0.15
10854	10	31	100	0.15
10854	13	6	65	0.15
10854	10	31	100	0.15
10854	13	6	65	0.15

Page: **1396** out of **2000** pages

Loading...

Change page: ... 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 ... | Displaying page 1757 of 2000, items 87801 to 87850 of 100000.

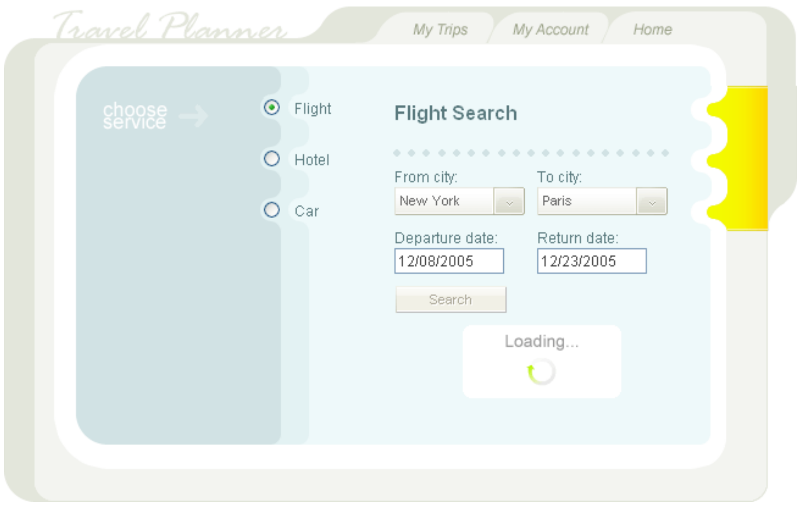
4. Benefits of AJAX

The advantages of AJAX-enabled applications to classic ones can be summarized as follows:

- **Better Performance and Efficiency** – the key advantage of AJAX applications is the significantly higher performance, which is a result of the small amount of data transferred from the server. This makes the AJAX technique especially beneficial for data-intensive applications (e.g. displaying rich reports, browsing through large data structures) as well as for low-bandwidth networks.

- **More Responsive Interfaces** – the improved performance leads to much more responsive interfaces, which create the illusion that updates are happening instantly. As a result the AJAX web applications appear to behave much like their desktop counterparts.

- **Reduced or Eliminated "Waiting" Time** – in AJAX-based applications only the relevant page elements are updates, with the rest of the page remaining unchanged. This approach eliminates the white screen (or page flicker on faster connections) and significantly decreases the idle waiting time.



From city: New York
To city: Paris
Departure date: 12/08/2005
Return date: 12/23/2005

- **Increased Usability –** as the web application is more efficient through the use of AJAX, and the client can communicate with the server without page-loads, it becomes possible to build some wonderful user interfaces that fit much better with users' needs and expectations.

- **Users Can Work with the Rest of the Page** – many AJAX-based applications allow you to continue working with the rest of the page, while data is being transferred in the background. This further adds to the uninterrupted manner of the end-user experience. Some commercial components (like **t**elerik r.a.d.**callback**) allow you to call a number of AJAX callbacks simultaneously.

5. Problems and Challenges when Using AJAX

The benefits of the AJAX development technique may look very attractive, but its practical implementation from scratch is usually a complex task, which only advanced developers can undertake. Among the pitfalls you will face are:

- **Writing and Maintaining Complex JavaScript –** building AJAX-enabled applications requires substantial JavaScript skills, which may turn to be a problem a large number of .Net developers. Furthermore, the lack of good debugging tools for client-side script makes the process even more complicated.

- **ViewState Management –** ASP.NET web controls properly maintain their ViewState between postbacks. The same, however, does not apply for AJAX callbacks. As a result, developers need to figure a way for the proper management of the page ViewState.

- **Breaking the Page Paradigm –** AJAX requires a different way of thinking about a web-site, since the concept of a "Page" is no longer valid. In fact, AJAX applications may be considered as closer to the desktop-applications development approach. The fact that a Page no longer holds constant data leads to two important consequences – the Back button and bookmarking will no longer work as expected. Therefore, developers need to implement specific mechanisms for overcoming these two issues.

- **Accessibility** – the AJAX development technique fundamentally violates the requirements for accessibility. Since the page content is being updated dynamically, the changes may not be detected by accessibility tools like screen readers. Furthermore, some accessibility standards prohibit the use of JavaScript altogether, which practically eliminates the possibility for using AJAX.

- **New UI Interactivity Requires Learning** – the UI richness of AJAX-enabled application introduces presents users with new and unexpected functionality. Although this is the main reason for using AJAX in the first place, it may require some learning.

6. When to Use AJAX

The following are scenarios where it's more likely to be beneficial to use AJAX functionality:

- **Highly interactive applications** – where an application is highly interactive, it may provide benefits to use AJAX to allow the interactivity without the time cost of page reloads. A good example of a highly interactive web site is Google Maps.

- **Parts of pages** – AJAX is best used in key places throughout a web site where it adds significant value. There is little or no value in using AJAX to provide the majority of your page content – you may lose browser compatibility, make the application overly complex and cause problems with the site being indexed in search engines.

- **Intranets** – on an internal web site the bandwidth is much higher so more AJAX usage can be successful. It's often the case that intranets involve working with data, and this is often a good application for AJAX.
- **Online wizards** – linear user interfaces that collect a large series of data are a well-known challenge in web applications, and can be implemented using AJAX in a way where the user feels more in control and the application is more responsive.
- **Data input & validation** – before AJAX it was hard to build anything resembling rich data input forms that give clear and accurate feedback to users. With AJAX it's possible to validate the data the user enters, while they are entering it. They can then receive feedback (using the server's intelligence) without the page being posted back. For example an email validation field could check the email address is valid using the DNS, without the user knowing.
- **Data visualization** – visualizing large datasets on the web can be difficult because there's been no easy way to pull in more data as the user needs it. With AJAX it's possible to show a view on the data, and simply load more in as needed. Again, Google Maps is a great example.

7. How to use AJAX

Here are some top tips for things you should consider when using AJAX:

- **Give visual feedback** - When a user clicks on something in the AJAX user interface, they need immediate visual feedback, just like when a button presses in on a standard form. Immediate feedback helps users, and makes them more efficient.
- **Keep the Back button** – make sure that the Back button in your application functions on every page of the site.
- **Use links for navigation** – avoid the temptation to use links as an interface on your AJAX application to change the state of your application. Users have been trained over ten years to expect a link to "take" them somewhere, so give them what they expect.
- **Limit the scope of visual changes** – when an AJAX call results in one or more changes to what the user sees, keep the changes local to the place where the user would expect them to be.
- **Use human-readable links** – people like to pass the addresses of useful web pages to each other. Make sure your application supports URLs that people can share easily, so not too long or complex.
- **Don't bloat the code** – make sure that your application uses as little client-side scripting as possible. This reduces download time for the page and also reduces the processor requirements on the client browser, so results in a faster browser experience.
- **Follow UI conventions** – AJAX is a world of possibilities, but when it comes to user interface the best is invariably the familiar. If you're creating a user interface for a specific feature, the place to start is by replicating an existing successful interface and looking at what your clients expect. Also remember that although it may be cool to implement drag-and-drop, few people may realize the interface relies on it.
- **Don't scroll** – users like to feel in control, so if they have moved the scrollbar to a specific place, don't move the page somewhere else.

- **Reduce page loads** – do as much as you can to reduce the number of page loads the user has to do to achieve their goal.

8. What You Should Know When Using Commercial AJAX Components

A common misunderstanding is that you can put an AJAX wrapper/container around your controls and suddenly your whole applications will start working like Outlook Web Access. It is quite easy to simply render an ASP.NET control after a callback request. However, this is where the story begins. When choosing commercial components that add some AJAX features to your web application you have to make sure that they satisfy the following functional requirements:

- **Maintain the Page ViewState Between Callbacks** – this is the biggest obstacle when using AJAX. Since no postbacks are performed, web controls will not maintain their ViewState automatically. It is up to the developer or the commercial AJAX component to ensure that the ViewState of all controls on the page reflects their current state. If this is not true, your application will not work properly.

- **Preserve the Normal Page Lifecycle** – most web applications need to use a mix of AJAX callbacks and regular postbacks. As a result you have to make sure that the AJAX components you choose will properly submit on postback the correct form data to the server after numerous callback updates have been made.

- **Reflect Control State on the Server** – in real-life scenarios there is no use of simply rendering controls on the page. You will also need to get the values of input controls (i.e. get the value of a textbox or the checked state of a checkbox) for server-side processing, as well as to set the values when the response is formed. Therefore, the AJAX components you choose must be able to properly reflect on the server the current state of all web controls, just like this is done during regular postbacks.

- **Update 3rd Party Controls with Complex JavaScript** – due to the fact that many ASP.NET application use commercial 3rd party controls (e.g. powerful datagrids, treeviews, WYSIWYG editors, etc.), your AJAX components must be able to properly update such controls. The biggest problem here is that 3rd party controls often use complex JavaScript, which is damaged by universal AJAX wrappers/containers. This drawback makes such wrappers practically unusable for the majority of modern ASP.NET applications.

- **Cross-Browser Support** – writing cross-browser compatible AJAX components is a big challenge due to the different implementation of the XmlHttpRequest object. Therefore, you should make sure that all AJAX components you choose operate properly on various browsers and platforms.

- **Proper Operation when Cookies are Disabled** – another requirement related to wide application compatibility is the support for cookieless sessions.

AJAX and telerik

1. History of AJAX in Telerik Components

Since 2002 telerik has been recognized as the leading vendor of UI and data components for ASP.NET, providing professional developers with a comprehensive selection of tools for building rich and highly interactive web applications. The key to the success of the telerik r.a.d.**controls** product line is constant innovation and adoption of the newest technologies and development techniques.

Realizing the potential of AJAX long before it had a name, **t**elerik introduced the load-on-demand functionality in the r.a.d.**treeview** control, back in 2003. Since then other products like the r.a.d.**combobox** and r.a.d.**grid** were developed with internal support for AJAX. With the release of r.a.d.**controls** Q3 2005 and the introduction of the r.a.d.**callback** suite (part of r.a.d.**controls**) AJAX support has been implemented across the full product line of ASP.NET components.

2. Telerik's unique offering

Historically betting on technological innovations, **t**elerik is offering a multi-level solution what will help you to implement various AJAX based functionality, depending on the specifics and level of sophistication of your project.

The main purpose of the telerik solution is to take the complexity of AJAX development off the shoulders of developers, and handle automatically all common issues mentioned before. Developers do not need to have extensive JavaScript experience, nor need to comprehend the AJAX callback lifecycle in details.

In the same time, by using only the **t**elerik r.a.d.**controls** suite you will be able to build even highly complex AJAX applications with rich interface like Outlook Web Access.

The telerik offering is comprised of three different ways of providing AJAX support and interoperability:

a) Built-in AJAX support in data-intensive components

Description: All **t**elerik components that work with large data structures use AJAX internally to load relevant data and update their own interface on demand.

Applies for: r.a.d.**grid**, r.a.d.**treeview**, r.a.d.**combobox**, r.a.d.**calendar**.

Benefits: Tremendous performance optimization, highly interactive interfaces that closely mimic the desktop counterparts of these components.

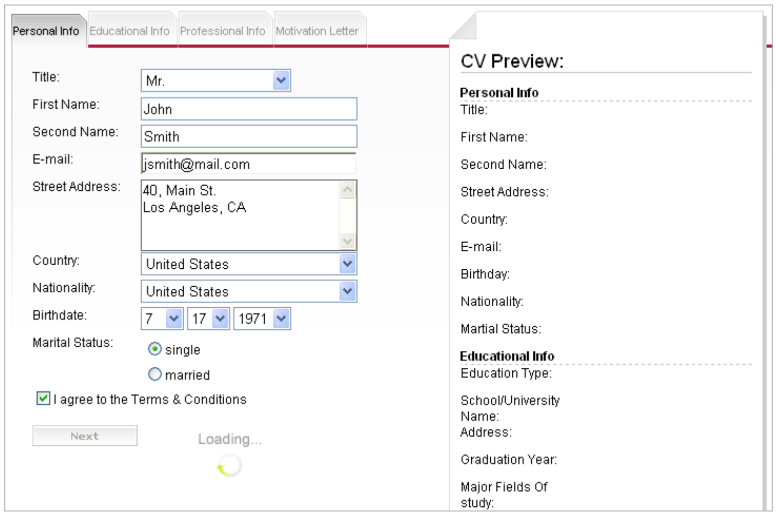
Examples: when performing paging or virtual scrolling in r.a.d.grid, the component loads in the background only the section of the records that need to be currently displayed. This amount of data is relatively small so the paging/scrolling appears to be almost instant. Furthermore, record editing, expanding, and all other actions that typically require a postback, can be performed with AJAX, giving the speed of a client-side operation without the cost of long loading times. See this in action.

How to use: simply set a single property and the respective component will start working in AJAX mode.

Notes: It is important to note that the respective component is able to update only itself, i.e. this internal AJAX engine cannot initiate a callback request and update other controls on the page. This scenario has to be approached differently ***(see point C below).***

b) A suite of 15 AJAX-enabled versions of the standard ASP.NET controls (r.a.d.callback suite)

Description: When you need to initiate a callback request with a standard ASP.NET control (i.e. Button, CheckBox, etc.) you can use the telerik AJAX-enabled version of this control. No extra code or other settings are needed – just replace the control with the new



Title:	Mr.
First Name:	John
Second Name:	Smith
E-mail:	jsmith@mail.com
Street Address:	40, Main St. Los Angeles, CA
Country:	United States
Nationality:	United States
Birthdate:	7 / 17 / 1971
Marital Status:	[✓ single] married

[✓] I agree to the Terms & Conditions

CV Preview:

Personal Info
Title:
First Name:
Second Name:
Street Address:
Country:
E-mail:
Birthday:
Nationality:
Martial Status:

Educational Info
Education Type:
School/University Name:
Address:
Graduation Year:
Major Fields Of study:

one and it will start making callbacks instead of postbacks, in order to update the rest of the page elements behind the scene.

Applies for: all components in the **t**elerik r.a.d.**callback** suite.

Benefits: you can quickly AJAX-enable an application that uses standard ASP.NET controls. The interface will be smoothly updated, the page ViewState will be maintained, and no further modifications to your application will be necessary.

Examples: if you have a long form, whose elements are populated in accordance with the user input you can avoid the numerous disturbing postbacks by using the **t**elerik form controls from the r.a.d.**callback** suite instead. The form will be updated almost instantly and the user may continue entering data while a particular field is being updated with AJAX. See this in action.

How to use: replace each control that initiates a postback with its AJAX-enabled alternative from the **t**elerik r.a.d.**callback** suite. The controls which are merely being updated, but do not initiate the callback request themselves, do not need to be replaced with their AJAX alternatives.

Notes: Besides AJAX-enabled alternatives of the standard ASP.NET controls, the **t**elerik r.a.d.**callback** suite includes 3 very important controls:

- **CallbackPanel** – a wrapper control that allows the AJAX engine to properly update the products from the **t**elerik r.a.d.**controls** suite, without damaging their JavaScript. See it in action.

- **RadCallback** – a generic callback control, which allows the products from the r.a.d.**controls** suite to initiate callback requests and update other page elements.

See it in action.

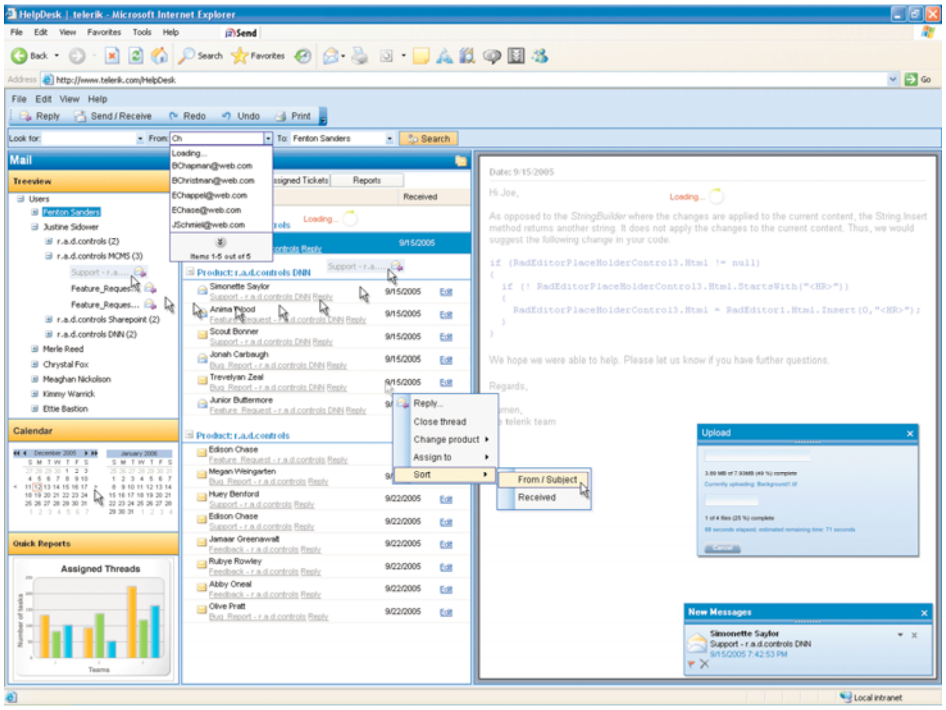
- **CallbackTimer** – a timer control that makes a callback request at a given time period, e.g. every 5 seconds. See it in action on the **t**elerik home page.

c) Full AJAX interoperability of the products in the r.a.d.controls suite

Description: Every product from the r.a.d.**controls** suite has been specifically modified for complete interoperability with the AJAX-enabled telerik r.a.d.**callback** controls. Although r.a.d.**controls** use complex JavaScript, they will be properly updated by any of the r.a.d.**callback** controls. In addition, components that typically cannot make callbacks to update other elements on the page can now perform this function.

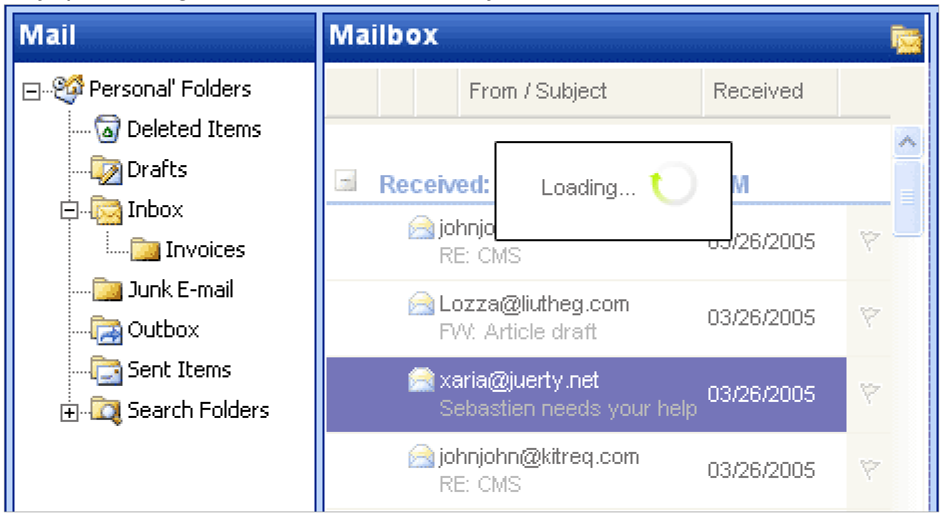
Applies for: all products from the r.a.d.**controls** suite.

Benefits: you can build complex ASP.NET applications with rich UI using telerik r.a.d.**controls** and still enjoy the benefits of the AJAX development technique.
- Every product from the r.a.d.controls suite can be properly updated by a r.a.d.**callback** AJAX request.
- Every product from the r.a.d.**controls** suite can initiate an AJAX callback request and update other controls on the page.
- You can build sophisticated web-applications that do not make a single postback (i.e. Outlook Web Access). An example of such application is the **t**elerik HelpDesk sample.


HelpDesk | telerik - Microsoft Internet Explorer
File Edit View Favorites Tools Help
http://www.telerik.com/HelpDesk
Reply Send / Receive Redo Undo Print
Look for: From: To: Fenton Sanders Search
Mail
Treeview
Users
Fenton Sanders
Justine Sidower
Calendar
Quick Reports
Assigned Threads
Product: r.a.d.controls DNN
Product: r.a.d.controls
Reply...
Close thread
Change product
Assign to
Sort
From / Subject
Received
Upload
New Messages
Simonette Saylor
Support - r.a.d.controls DNN
Local intranet

Examples: Imagine you have an interface similar to that of Outlook, with a treeview on the left and datagrid on the right (r.a.d.**treeview** and r.a.d.**grid**). What you will typically need to do is update the datagrid when you click one of the treeview nodes. This function includes two sub-actions:
- the r.a.d.**treeview** control, which typically uses AJAX only internally, should be able to

initiate a general callback request, identical to the general page postback.
- the r.a.d.**grid** control which uses complex rendering JavaScript should be updated properly, preserving its client-side functionality.



By using r.a.d.**treeview** and r.a.d.**grid** in conjunction with the CallbackPanel and the generic RadCallback controls, developers can easily implement this scenario.
See it in action.

How to use:
- to properly update a product form the r.a.d.controls suite you need to enclose it in a CallbackPanel (included in the telerik r.a.d.callback suite)
- to initiate a callback request by a product from the r.a.d.controls suite you need to use it in conjunction with the generic RadCallback control (included in the telerik r.a.d.callback suite)

Notes: the unique telerik offering of market-leading UI components that are also fully AJAX interoperable (thanks to r.a.d.callback suite) cannot be matched by combining any of the available UI suites with any AJAX-enabling component(s).

3. Telerik and AJAX in the future

We believe that AJAX will fundamentally change the perception of web user interface as we know it. Stepping on our substantial expertise as a leader in ASP.NET UI component development, we will continue our path of innovation, making the AJAX functionality a central aspect of our product offering.

Developers may soon expect to see novelties in the following areas:
- Better internal AJAX support of telerik r.a.d.**controls**
- New AJAX based UI components
- Improved AJAX engine
- Interoperability with ATLAS