

# SaaS USER INTERFACE





## TABLE OF CONTENTS

> 1. Introduction	1
> 2. UI Definition	2
> 3. Navigation	2
3.1 The Early Web	2
3.2 Interactivity Emerges	3
3.3 The Enriched Experience	3
3.4 Thin Clients Emerge	3
3.5 Keep It Simple	4
> 4. Web UI	4
4.1 HTML UI Controls	4
4.2 Tabs	4
4.3 Controlling Page Flow	5
4.4 Browser Compatibility	5
> 5. Presentation Options	6
5.1 Tree Versus Table	6
5.2 Paginated Table	7
5.3 Limiting Result Sets	8
5.4 Context Switching	8
> 6. Mobile Device Support	9
> 7. Other UI Options	10
> 8. Summary	10
> 9. References	10
> 10. Glossary of Terms	11



## 1. INTRODUCTION

This paper is one of a series of papers that explore and discuss the technical architectural components of the Software-as-a-Service (SaaS) model.

SaaS shares the distinction of being both a business model and an application delivery model. SaaS enables customers to utilize an application on a pay-as-you-go basis and eliminates the need to install and run the application on the customer's own hardware. Customers generally access the application via a web browser or thin client over the Internet. SaaS is most often subscription based and all ongoing support, maintenance, and upgrades are provided by the software vendor as part of the service. Application customization capabilities, if available at all, are generally provided to all customers in a consistent manner. From the perspective of the software vendor, the SaaS model provides stronger protection of its intellectual property, operational control of the environment running the software, and generally a repeatable revenue stream from the service subscription fees. Software vendors have varying capabilities and applications can come in varying flavors but SaaS applications most typically support many unique customers using a single instance of that application - also known as multi-tenancy.

Since SaaS by definition generally involves users accessing the service via a web browser, the topic of SaaS user interface (UI) is an important one to explore. The web browser is not the only possible way to access a SaaS application, and we will discuss mobile devices as one alternative, but there are many issues to be considered in delivering an application to users via a web browser.

In this paper, we will cover some of the most prominent high level issues. Related issues are covered in a separate paper on SaaS Customization and Personalization. We have attempted to address what we consider to be the major issues between these two papers. However, the topics of UI design, usability, web-specific UI issues and UIs for alternative devices have spawned countless books covering the details, trends and latest technological advances. This paper provides the SaaS vendor with a place to start.

We will discuss different aspects of application navigation and how the evolution of web based UI has led to certain conventions that today's users have become comfortable with. We will then review different aspects of a web UI and how these have changed somewhat radically in the last few years. Our discussion will then cover some different presentation options that will provide the reader with an idea of the types of things that might be thought about when designing the UI for a SaaS application. Finally, we will look at some of the issues for consideration in providing UI support for mobile devices.



## 2. UI DEFINITION

According to Wikipedia, user interface is defined as follows<sup>1</sup>:

The user interface (or Human Machine Interface) is the aggregate of means by which people—the users—interact with the system—a particular machine, device, computer program or other complex tools. The user interface provides means of:

- > Input, allowing the users to manipulate a system
- > Output, allowing the system to produce the effects of the users' manipulation.


We can see from this definition that we are talking about input and output, what a user does and sees, as a means of interaction with an application. Note that a user interface is distinct from an application programming interface (API), which provides essentially the same function but for another program or system to interact with the application rather than a person.

## 3. NAVIGATION

Thinking about how a user will navigate around an application to access different functionality presents more challenges than one might think. As technology professionals, application developers sometimes take for granted how they have intuitively adapted to certain UI conventions and find it hard to believe that the users of the applications they write may not understand those conventions at all and end up confused. When it comes to navigation in a web based application it can be helpful to look back at how things have evolved since Tim Berners-Lee invented the World Wide Web in 1989<sup>2</sup>.

### 3.1 The Early Web

The initial specifications for the web included a markup language (HTML – HyperText Markup Language) to format text, a protocol (HTTP – HyperText Transport Protocol) for transmitting these pages across the internet from a web server to a client browser, and a link mechanism (URIs – Universal Resource Identifier) to define the location of a page and therefore connect one page to another. Basically, this setup was put in place to make it easy for scientists at CERN, the European Particle Physics Laboratory where Tim Berner-Lee worked at the time, to easily share information with each other and with other scientists around the world.



The early web had very few web sites (web servers) containing content and this content was mostly textual in nature. Links (or hyperlinks, the HTML notation that turned ordinary text into click-able text that directed a user to the referenced page via its URI) were used as simple means to follow a referenced concept to its source document, and thus became the genesis for today's navigation paradigms.

### ***3.2 Interactivity Emerges***

As the technology quickly caught on in the early 1990s and started being picked up by the business community (as opposed to academia), functionality began to grow and people started to develop more interactive applications. Rather than simply clicking from one static HTML page to another, people began to use web based forms as a means to collect user input and submit it for processing on the web server, often for storage in a database. The concept of dynamic pages began to catch on. These were web pages created on the fly at the web server based on user interactivity and data being input or output from a database. This led to the emergence of web application servers in the mid-1990s.

As this technology began to develop and more people began to use it, full blown business applications began to appear. Mechanisms for navigating around these applications began to follow certain design standards that static web site designers were deploying. In generalized terms this included some kind of main or top navigation and associated sub or side navigation that enabled users to move around a site or application without getting lost.

### ***3.3 The Enriched Experience***

In the years that followed, use of the web exploded worldwide and all sorts of sophisticated applications were deployed using web technology. The web application paradigm began to supplant and surpass the popular client-server model and web developers found themselves creating technology to more closely mimic the tight GUI (graphical user interface) interactivity of desktop applications within the browser. Along with this enriched UI technology, came more sophisticated navigation elements including functionality such as rollover behavior, drop down menus, etc.

### ***3.4 Thin Clients Emerge***

As browser-based applications gained more sophistication, other thin-client approaches to web application development were explored. A thin client is a non-browser GUI that generally runs either via a browser plug-in or via some other technology that allows download at runtime. Some approaches even involved the download and installation of the thin client locally, so it was run subsequently directly from the desktop.



### **3.5 Keep It Simple**

That brings us to where we are today. With widespread acceptance of web based applications, users are now seeing the benefits of hosted solutions or SaaS and this model is now seeing its own widespread adoption phase. Navigation schemes have run the gamut from very simple text based links to more sophisticated graphical elements with enriched functionality. The key to remember is to keep it as simple as possible for users to find what they need and understand what to do next. The old standard concepts of top navigation to get to the main areas of an application and sub navigation to get to the different functions within a main area has become so well known that virtually anyone with any experience clicking around the web will feel comfortable following this general paradigm.

## **4. WEB UI**


There are certain behaviors that are unique to web based applications as opposed to desktop applications and we will discuss some of these under the general heading of web UI. As a SaaS vendor, it is important to think about these behaviors as the UI for an application is designed.

### **4.1 HTML UI Controls**

Controls are elements of HTML forms including checkboxes, radio buttons, textboxes, etc. Users generally interact with a form by modifying its controls (i.e., entering text, checking off selections, etc.). There are many old standard controls that users have become accustomed to but these generally require a submission back to the application server for processing and a subsequent page refresh. Newer technologies such as AJAX, Adobe Flex and Microsoft .NET provide more sophisticated functionality where user input can be processed back on the server and page elements updated without a complete page refresh. The SaaS vendor must decide what technologies and approaches are most appropriate for their specific application and type of user. Again, the general rule is to keep it simple and only use more sophisticated approaches if they assist the user in accomplishing a task more efficiently or effectively.

### **4.2 Tabs**

The tabbed interface metaphor emerged as a design approach that mimics the old tabbed file folders in a manual filing system. Since this paradigm was already familiar to non-technical users it carried over well to UI design and is used extensively in today's web applications. This metaphor allows a user to navigate through what are actually separate web pages but have the appearance being stacked elements on the same page. By clicking a given tab, the information on that page is displayed and the tab navigation



element appears to be on top of the stack. Use of this metaphor can not only help display a lot of related information in an organized fashion, but it is one that users generally find comfortable and intuitive to use.

### ***4.3 Controlling Page Flow***


One unique aspect of a web application is that it is stateless. What this means is that the page is requested from the server, returned to the client browser and then communication between the server and browser is terminated until another page is requested or a form is submitted for processing. Continuity within an application for a given user is maintained via session information that is stored on the application server. When a request is made by a user, a session identifier of some sort is returned (i.e., via a cookie or some other data element submitted), the server looks up the session information via the identifier and then is able to proceed within the context of the session. An example of such continuity might be knowing from what page a user is coming or what page needs to be displayed next.

Another unique aspect of a web application is that the pages of the application are displayed within another application – the web browser itself! The web browser application has its own menus, navigation elements, etc. that are distinct from those of the web application. This brings up certain challenges regarding the control of page flow. An application may need to bring a user through a series of pages in order to complete a given operation. If the browser navigation elements are used (i.e., the back button), the application may get confused if it was expecting data to be submitted via its own navigation elements. Another potentially confusing aspect to this issue is that multiple browser windows can be opened at the same time and most browsers have now even adopted the tabbed interface paradigm themselves. Users can easily get lost if they confuse some of these browser functional aspects with the functional aspects of the application itself.

These issues must be carefully considered by SaaS UI designers in order to create intuitive and easy to use applications. Thinking through all the potentially unexpected things users may do while using an application is imperative for effective UI design and managing page flow is especially challenging.

### ***4.4 Browser Compatibility***

Users have the option of using any of a number of different web browser products to access a given web application. As a SaaS vendor, it is difficult to control which browser product (and which version of a given product) your customers will use. While it is possible to design a UI that will only function properly in a given browser, this can significantly limit your potential customer base. As a result it is a good idea to support as



many different browsers as possible. This brings up its own set of challenges because certain vendors tend to follow generally accepted standards while others do not. The best advice we can provide is to establish a clearly defined list of the browser products and versions your application will support and then test, test, test in all of them.

## 5. PRESENTATION OPTIONS

We will now shift our focus to the different approaches for presenting information in a web UI. Aside from some of the issues we have already explored above, it is important to always be cognizant of how much data your application is returning to the client browser for a given request. Users expect fairly prompt response from web applications so it is important to think about how much data can realistically travel from the server to the client browser in a given time period. For example, sending 500K of data may be acceptable whereas sending 500MB might not be acceptable. All this depends on the specific application functionality, general users' connectivity speed, acceptable bandwidth usage, etc. Below we will present some different options for presenting data that may help alleviate this issue.

### 5.1 Tree Versus Table

Sometimes data is well suited for display in a hierarchical format. An example might be a departmental structure and there are various tree controls available today that can be used to display this hierarchical view in a user friendly way. However, while some companies may have a relatively small number of departments that work well in this view, another company may have thousands of departments that would result in too much data being returned to the browser at once. In these instances, a control such as a paginated table (see below) maybe more effective and the same data can be presented in a way that still gives the user the sense of hierarchy. Examples of these different approaches are shown in Figures 1 and 2 below. Note that this is an excellent example of two presentation approaches that can both be offered in an application, but which one is chosen by a specific user can be a configuration option that the user controls.

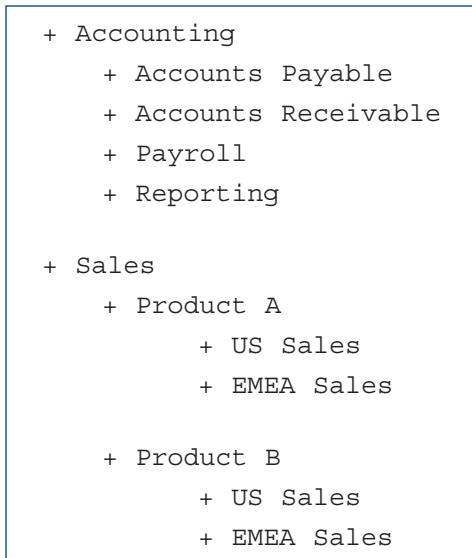


Figure 1 – Tree view of departmental data

Accounting	Accounts Payable	
	Accounts Receivable	
	Payroll	
	Reporting	
Sales	Product A	US Sales
		EMEA Sales
	Product B	US Sales
		EMEA Sales

Figure 2 – Table view of the same data

## 5.2 Paginated Table

A paginated table control is one that limits the amount of rows displayed from a given result set and generally has elements to move forward and backward a page at a time or to go directly to a specific page number. These controls generally tie directly into a database concept called a cursor in which it is possible to navigate through a large result set a certain number of rows at a time. This enables large result sets to be displayed and navigated by users without sending all the data to the browser at once.

It is strongly recommended that this approach be used rather than a straight table. Programmers have a habit of testing with small data sets and a page that may look fine in tests could produce undesired results when a live user tries to return a large result set. Use of a paginated table control will scale from the smallest result set to the very large and provide consistent response in either case.



### **5.3 Limiting Result Sets**

Another approach to handling large result sets is to provide programmatic limitations to how many rows that can be returned by a given query. In order to provide flexibility, users can be given the option of configuring these limits themselves. The result of using such limitations is that response times can be kept more consistent and over taxing the database can be avoided. In many cases users do not want to sift through huge result sets. Instead the application can limit the rows returned but inform the user that the results were limited and suggest further filtering to produce more usable results.

### **5.4 Context Switching**

Due to the nature of web applications generally following a submit/request and response paradigm, it is easy to confuse or overwhelm users by jumping around from screen to screen all the time. For example, a user maybe looking at a result set of employees for a given department. This departmental view will therefore be the current context with which the user is looking at the data. Now suppose the user needs to edit the detail data for one of the employees. One UI approach might be to display the employee name as a link and if the user clicks this link they can be presented with a new screen to edit the employee data. At the bottom of the edit screen one might expect to find save and cancel buttons, either of which might return the user to the previous screen. This is an example of a context switch – the user’s context has switched from a departmental view to an individual employee view, and then back again – and a lot of these can be jarring to the user.

Another UI approach might be to have the click-able link change the employee data in-line to an editable state and then back to a view-only state when the data is changed and the field exited (i.e., via hitting the tab or enter key). The advantage of this approach is that the data can still be edited but this is accomplished without the jarring context switch. There are many modern web UI controls that can be found to do all sorts of useful functions in a more efficient manner such as this. It is up to the UI designer to think about how to make the interface as user friendly as possible and avoiding context switching wherever possible will help achieve this goal.



## 6. MOBILE DEVICE SUPPORT

More and more business users are moving toward mobile devices with advanced capabilities to access web based applications. However, these devices present certain challenges to users because of numerous limitations<sup>3</sup>, including the following:

- > Limited bandwidth - A wireless device typically has much less bandwidth available for transmitting and receiving data than a wired device.
- > Intermittent connection - The connection to a wireless device is typically unreliable. A persistent point-to-point connection is difficult, if not impossible.
- > Limited battery life - A wireless device is typically (also) a mobile device. Since mobility dictates compactness in size, and since there is no wired power connection, batteries are the only means of power supply. Even the longest lasting batteries offer a very limited amount of power.
- > Limited memory on client device - Once again, because of the mobile nature of wireless devices and their requirements to remain a small size, room for memory is limited. Memory is also limited by the available power source (batteries) on the device.
- > Limited CPU - Because of the size of the devices and the battery life, processing information on the device is very expensive. Very few operations should be performed on the device and they should be only performed where there is strong justification for them.
- > Limited user interface - A keyboard and/or a mouse are normally not available for a wireless or a mobile device. Also, the display is almost always very small. This makes viewing and data entry more difficult.

Understanding these limitations is a critical consideration for a SaaS vendor that is looking to allow users to access their service via mobile devices. Providing UI options that support different mobile devices can have a major impact on acceptance and adoption of a given service. A great example is Twitter, which virtually exploded in popularity via a largely SMS-based capability. You can certainly login via a normal web browser and access the service via this UI, but it was the simple use of SMS that helped lead to their success.



## 7. OTHER UI OPTIONS

As mentioned earlier in section 3.4, other non-browser GUI or thin-clients are also options for a SaaS UI. There are numerous technologies available today to select from but regardless of what technology used, certain issues must be addressed. First, is the issue of how you will distribute the thin client GUI to the user. Options may include running via a browser plug-in, download and install at runtime, or installation and then subsequent running from the desktop. Another issue that needs to be addressed is how the GUI will be updated when development changes are made. This is less relevant for distribution modes at runtime, but if the thin-client is installed on the desktop there needs to be some sort of check for updates. Last, we will address the issue of GUI “richness.” The idea of providing “client-server-like desktop GUI functionality” for a web-based application is often an excuse for needlessly complicated UI design. While there are certainly applications that may call for certain rich UI functionality, the SaaS vendor is advised to always aim toward keeping the UI as simple and intuitive as possible. This generally results in an overall more positive user experience and therefore to happy customers.

## 8. SUMMARY

This paper has touched upon some of the high level aspects of web UI design that might be considered when designing a SaaS solution. Having a thorough understanding of the issues that are unique to a web based interface is helpful. We discussed the evolution of navigation in web applications in order to give the user some perspective on what techniques are being used today. We then proceeded to distinguish some of the unique aspects of web UI elements. Following that we presented some presentation options that can be used to avoid scalability issues. We then finished by presenting some considerations when providing mobile devices support. There are many aspects of SaaS UI design and implementation that were not covered as the scope of this topic could encompass many books. However, we have endeavored to provide a good starting point for addressing this topic within an overall SaaS architecture.

## 9. REFERENCES

<sup>1</sup>Wikipedia, “User Interface,” – [http://en.wikipedia.org/wiki/User\\_interface](http://en.wikipedia.org/wiki/User_interface).

<sup>2</sup>Tim Berners-Lee bio – <http://www.w3.org/People/Berners-Lee/>.

<sup>3</sup>B’Far, Reza, with Richards, Roger and Ditlinger, Stephen, “Designing Effective User Interfaces for Wireless Devices,” – <http://archive.devx.com/wireless/articles/rb0501/rb0501-1.asp>.

## 10. GLOSSARY OF TERMS

Term	Description
CSS	Cascading Style Sheets (CSS) is a style sheet language used to describe the presentation of a document written in a markup language.
Customer/Tenant	Customer/Tenant is a term in this document that refers to the company or business that subscribes to the SaaS offering.



**Worldwide Headquarters**

Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA  
Tel: +1 781 280-4000 Fax: +1 781 280-4095  
[www.progress.com](http://www.progress.com)

**For regional international office locations and contact information, please refer to [www.progress.com/worldwide](http://www.progress.com/worldwide)**

© Copyright 2008 Progress Software Corporation. All rights reserved. Progress is a registered trademark of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. or other countries. Any other trademarks contained herein are the property of their respective owners.