Web Atoms

http://webatoms.neurospeech.com
webatoms@neurospeech.com

Getting Started

Please make sure you have enough knowledge about following before continuing this document.

1. ASP.NET MVC or ASP.NET ASPX
2. HTML, JavaScript and CSS

Installation & Configuration

You must include jQuery 1.7 onwards in your page before including any of following resources. Since jQuery is licensed separately and we are not associated with them, you must download and include in your project manually.

If you are using an ASP.NET MVC project, than you can include NeuroSpeech.WebAtoms.dll and call a method @Html.WebAtomsDebug() in the Header section of MVC view. This will include necessary JavaScripts and CSS files.

Otherwise, you have to include following files in your page.

1. Content/WebAtoms.css
2. Content/Flash/FileUploader.swf
3. Content/Flash/playerProductInstall.swf
4. Content/Flash/SingleFileUploader.swf
5. FlashPlayer.js
6. JSON.js
7. linq.min.js

If you wish to use Video Player, then you must include following files from jwPlayer's website

1. jwplayer.js
2. player.swf

Web Atoms for Web Application

Web Atoms is designed to develop Web Applications and not Web Sites.

Difference between Web Application and Web Site

Web Application	Web Site
Less read, more write (add/update/delete etc)	More read, less write
Does not need SEO	Need SEO
Multiple Scrolling Panels – entire application does not scroll, instead we have top panel toolbar, left panel and status bar which are fixed and internal contents are scrollable.	Single Scrollable Page – we often need website with one scrollable page from top to bottom and they appear as a physical paper document.
Contains Fixed Toolbar, Left Panel (Usually a tree or list) and Status bar.	Usually does not contain any fixed/static panels. Scrolling page will remove top/bottom panels out of view.
Resizes and automatically layouts application panels as browser/container resizes.	Web site stays fixed regardless of browser's size, instead visibility scrollable area changes, but most parts of page remain relatively static to each other.
Only relative parts of page refreshes when user does any activity (known as ajax).	Unless you are moved onto different page, mostly page remains static and entire page is refreshed, however most pages do contain javascript effects, but they are not SEO friendly.
Application requires mostly a secure access.	Web sites are generally public facing documents without secure access.
Web Application there for needs more scripts and more interactivity on client side instead of server side.	Web sites are less interactive at client side and needs more logic on server side.
Web Application just needs to work, except end user's functionality; making it perfect HTML5 or any other standard is less important.	Web site must comply to open standards.

Goals

We did identify lots of concepts and have already seen various JavaScript libraries, but none of them fit our requirements. Larger business applications are programmed by many developers and many teams. Other platforms such as Flex and Silverlight offer a lot in terms of managing and maintaining code. Writing code and managing code are two different aspects. Abstractions and rewriting logic in the form of simpler blocks makes everyone's life very easy. Our goal was to bring new concepts into HTML so that visualizing and maintaining code becomes easy.

Service Oriented Architecture

PHP, ASP, Razor are easier to write, but very difficult to maintain and visualize the logic as combinations of inline script operators. MVC architecture is good but when it comes to user interface, the final html code, it becomes messy.

Moreover, it needs more roundtrip if user's view needs changes. I am aware of partial views etc, but they are still messy.

Rich Internet Applications brought concept of delivering UI to the client independently of data and let the client request for the data needed from server side. So by either JSON or SOAP Web Service, data is sent to client and let client display, manage and play with it.

UI Logic at Client

Sorting, Filtering, looking at alterative view, resizing these things should happen at client side without refreshing entire page.

Model View Separation

WebAtoms focuses on truly model view separation, where UI is written strictly with use of binding and templates, and model is sent separately.

Following creates a scoped variables model and fullName method, which are bound by the UI.

```
<!-- Model (Data) -->
<script type="text/javascript">
	({
		model: { firstName: 'Peter', lastName: 'Parker' },
		fullName: function (fName, lName) {
			return fName + ' ' + lName;
		}
	})
</script>

<!-- View (User Interface) -->
<div
	atom-dock="Fill"
	atom-type="AtomControl"
	atom-data="[$scope.model]"
	>
	<div atom-text="[$data.firstName]"></div>
	<div atom-text="[$data.firstName + ' ' + $data.lastName]"></div>
	<div atom-text="[$scope.fullName($data.firstName,$data.lastName)]"></div>
</div>
```

Loading model from server in the form of JSON data is also very easy. Following code will load a JSON serialized array, which will be fetched from the given URL and list box will display items based on the label field. $URL binding will invoke AJAX request to target URL and it will apply resultant javascript object to items property of the AtomListBox.

```
    <div
		atom-type="AtomListBox"
		atom-items="$URL['/Demo/PageList']"
		>
		<div atom-template="itemTemplate" atom-text="[$data.label]"></div>
    </div>
```

More Markup – Less Code

By separating Model and View, it becomes easy to maintain the code. Most JavaScript frameworks involve lots of coding, lots of ID mapping, which makes it difficult in visualizing the logic and UI at time of development. But WebAtoms is inspired from languages like Flex, Silverlight where XML markup binding makes it easy to visualize and control the UI.

- Markups are easy to collapse and expand in any visual editor.
- Scripts can also be collapsed and expanded but not in hierarchical form.
- We can visualize hierarchy easily by looking at the code.
- Instead of server side code snippets, the layout of application and inline conditional binding of attributes makes it easy to maintain.
- Markup errors are easy to detect.
- Markups are easy to refractor and reuse.

Templates

Ok, we did get the model, that is available in the form JavaScript object, and object contains properties and collections etc. We need templates at client side that can display the data received from server.

Binding

Binding expressions should help us in creating user interface elements display data correctly in the format that we expect and it should let us write complex conditional expressions to display data based on user's choice and bound data contents.

Two Way Binding

UI should update itself automatically as selections change by user or when data is entered by user. This needs concept of two ways binding let UI elements update its own properties or data based on user interaction.

No New Language

Great, we want to incorporate everything, but we do not want to invent a new language, we want to use exiting HTML and add few attributes to make everything work correctly.

Controls

WebAtoms comes with many inbuilt default controls, which lets you easily create application quickly and lets you customize user interface simply by adding your customized CSS.

Application Container

Application container is an UI element, which auto resizes its children when host browser is resized. HTML does not support 100% height for child elements; it does not support docking simply by using CSS. CSS is still missing attributes for conditional resizing and other forms of UI controls that we usually see in any other UI platforms such as flex, Silverlight or any other native UI platform. We have introduced two powerful controls that allow us to create any complex UI.

AtomDockPanel

DockPanel docks the children based on dock attribute. Toolbars are usually docked on the top, status bars are docked on the bottom. And Tree Views are docked on left, and middle space is filled with remaining space and only middle space is scrollable and internal items of Tree View etc are scrollable. As application host browser is resized, DockPanel will rearrange its children to make docked items appear correctly.

AtomViewStack

ViewStack displays only one item at a time, and it stretches its visible child item to acquire entire space by the control. ViewStack's item visibility can be changed by changing selectedIndex property and item at that index will become visible. You can think of ViewStack as the area just below tab buttons in tab control, where based on selected tab, the item is visible hiding all other tab's corresponding items. ViewStack is usually used within templates to control visibility of item group or layout.

Concepts

Element

Any html element such as div, a, ui, li etc are referred here as HTML Elements throughout this document. You can bind any attributes of element only if the element is descendent of any AtomControl. Binding only works if an element is located within a parent AtomControl.

Atom Control

Any html element with attribute `atom-type="_____"` can be initialized as AtomControl. AtomControl attaches more functionality to the applied element. WebAtoms ships with some inbuilt controls such as AtomButton, AtomForm, AtomListBox etc. You can also create your own custom controls and use them in your project.

Initialization

Due to limitations of HTML and JavaScript, creating and initializing controls are little complicated, however we are still trying our best to improve the syntax.

1. Set Control Type

Apply `atom-type="_____"` to desired element and set the attribute value to corresponding control name.

2. Setup Initial Properties

Let's assume that AtomForm control has two properties, next and postUrl. And we want to initialize them.

```
<div atom-type="AtomForm"
        atom-properties="{ next:'/Home', postUrl:'/User/Login' }" >
```

This is a special form of property initialization where you can initialize multiple properties in JavaScript Object Literal form and you can refer a global JavaScript functions and objects. You cannot setup binding in this property initialization attribute.

This is useful to setup and initialize text strings which may not require escaping. Since other atom-* properties initializers may contain binding expressions, you might need to escape your strings. This initializes properties faster and all at once.

3. Initialize Individual Property

```
atom-file-types="{ [ { description:'Photos', extensions:'*.jpg;*.png;*.bmp; ' } ] }"
```

If you set individual property starting with curly braces, it will be considered as a JavaScript expression and it will be evaluated at initialization and result will be set to the associated property of the control.

Control Properties

Every AtomControl contains following properties which behave little differently than normal other properties. These properties are useful in data binding, where inheritance and scope are automatically mapped. Since JavaScript does not contain HTML DOM level scoping, we introduced these special properties which are only accessible under an AtomControl.

Data Property (Inhertied)

AtomControl has a property called “data”, and it can hold any object/value in it. And it refreshes all other controls that are bound to this data property. Data property is not set then inherited in DOM from the parent, and so on. You can set data property in code and in markup. By default, AtomItemsControl will create new AtomControl and set the data property corresponding to the index in items array. Thus data in child element of an AtomItemsControl will be different from the data of AtomItemsControl itself.

Scope Property (Inhertied, Readonly)

Although, scope is read-only, but you can still set a value, and it will merge the values in existing scope. Scope is inherited same as data, but scope is setup automatically by the WebAtoms framework. You can never change the scope, but you can change the properties within the scope, and it will be merged and property in same scope will be accessible to every controls in the same scope.

Scope also allows you to create JavaScript functions with same names in different scopes. For example, child controls of AtomItemsControl can contain same named JavaScript functions but each of them will be different for every scope and within that scope they will be unique.

Name Property (ReadWrite, Unique)

Every control can have atom-name attribute and this control will become an instance within the scope. So you can refer as $scope.name as instance of control within current scope. atom-name is different than id or html’s name. atom-name is way to identify current control in current scope. So you can have two control with same name in different scopes. This is important in case of AtomItemsControl with multiple items with exactly same item template and same logic. But since each item is in its own scope, every item refers correct control by having same name in different scope.

Owner Property (Readonly)

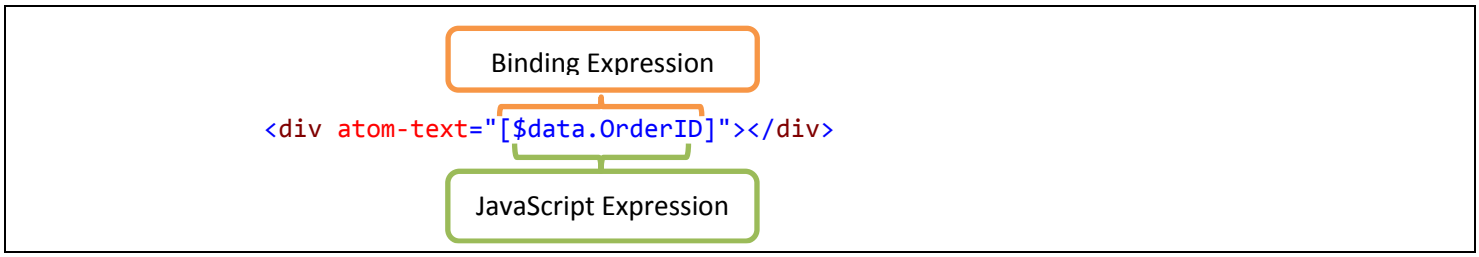
Owner refers to nearest parent AtomControl (Self in same AtomControl markup).

Binding

WebAtoms framework will identify binding expressions only within square brackets ([]), and we have chosen square brackets because, they are very rarely used in any text literals in html and curly braces ({}) represents JavaScript Object Literal Notations.

Binding will be applied only if a property expression begins with either of following words.

1. Data
2. Scope
3. Owner



Binding expression can contain any complex JavaScript expression in square brackets as shown above. You can also do some arithmetic operations or call some methods within binding expression. Expression will only be evaluated to set an initial value and whenever the target binding properties will change. In above case, text of div will change whenever OrderID property of data will change.

Text Binding

```
<div atom-text="[$data.OrderID]"></div>
```

Every element can contain inner text, which is why atom-text represents inbuilt binding, where it applies to inner text of the element. In runtime this div will contain the text corresponding to evaluating expression.

Value Binding

```
<input atom-value="$[data.FirstName]" type="text" />
```

Value binding can only be applied to form input element or any AtomControl that has value property, however in order to create two way binding $ sign should be used before binding.

Style Property Binding

Instead of managing complicated css classes and assigning them, sometimes styles are dependent upon data and which is easier to set in binding as shown below. Style property bindings start with “atom-style-“ or “style-“, we prefer writing “style-“ only.

Example below is the busyTemplate of AtomApplication which is displayed every time there is some network activity. The first binding for style-width automatically sets width to owner.appWidth property whenever window is resized. Visibility changes to ‘inherited’ or ‘hidden’ as busy status changes.

The inner div also positions itself based on appWidth and appHeight and tries to position itself exactly in center.

```
<div
	style="position:absolute;"
	style-width="[$owner.appWidth + 'px']"
	style-height="[$owner.appHeight + 'px']"
	style-visibility="[$owner.isBusy ? 'inherit' : 'hidden']"
	>
	<div
		style="position:absolute;visibility:inherit; width:200px; height:50px; background-
color:#fff; border-style:solid; border-width:1px;padding:5px; text-align:center; vertical-align:
middle"
		style-left="[(($owner.appWidth/2)-100) + 'px']"
		style-top="[(($owner.appHeight/2)-25) + 'px']"
		>Loading...</div>
</div>
```

In order to comply with html5, we might change syntax to data-bind-style-, I know it’s quite tedious, but what to do, creators of html5 just don’t allow any extensibility points.

Atom Control Property Binding

As element is attached to atom control, on the same element, you can set binding for atom properties which are defined in atom control. For example, selectedIndex of AtomListBox is an atom property, and you can easily set binding on same control as shown below.

```
	<div
		atom-presenter="popup"
		atom-type="AtomListBox"
		atom-items="[$appScope.items]"
		>
	</div>
```

As we know, items property do not exist in div, but since div is converted to AtomListBox, atom-items refers to items property of AtomListBox.

Post Binders

Post Binders are binding function prefix that allows us to process data before binding. This is important in scenario where we want to specify a URL as binding, but we want to actually bind data received from the URL. For example, we want to load list of countries, received in json format from the URL "/Config/Countries"

$URL

Url binding allows us to bind data received from the URL, whenever any component in binding changes, a new AJAX request is sent to resultant URL and data received is bound to target property.

```
        <select
            atom-label="Country:"
            atom-type="AtomComboBox"
            atom-required = "true"
            atom-value="$[data.Country]"
            atom-items="$URL['/Config/Countries']" >
        </select>
```

URL binding for items will fetch json data (list of countries) and it will populate the combo box with it. Please note, every item must have a property label and value. Following is an example of complex binding expression, where URL will be evaluated when data.Country1 will change. For example , in the first combo box, list of countries are displayed and only on basis of selected country data.Country1, URL will become "/Config/Countries/US" (if US is selected). And combo box will fetch new list and display.

```
<select
       atom-type="AtomComboBox"
       atom-label="State / Province :"
       atom-items="$URL['/Config/Countries/' + $data.Country1]"
       atom-value="$[data.State1]">
</select>
```

URL evaluation is complicated when more query string parameters are involved. So we have provided a method called Atom.url which will help you in concatenate URL correctly including URL encoding. For example, "/Config/Countries?CountryCode=US" should be bound as shown below.

```
<select
       atom-type="AtomComboBox"
       atom-label="State / Province :"
       atom-items="$URL[Atom.url('/Config/Countries', { CountryCode: $data.Country1 })]"
       atom-value="$[data.State1]">
</select>
```

Atom.url will expect first parameter to be of string, and second parameter as an anonymous object with properties. These properties will be evaluated at runtime and query string items will be populated correctly.

$CURL

CURL (Cached URL) is same as URL, but it will not create multiple request for same URL, instead it will reuse existing available result. For example, if you are loading list of countries multiple times in multiple drop down list, each list will fetch same list, and we will have multiple arrays as items source for multiple drop down lists. This just increases memory at client side, in order to reduce memory usage on client side, we have introduced CURL, which will return cached response of same URL.

CURL is not same as client side caching, even if the response from URL enables client side caching, browser will send cached response from browser's text, but JSON parser will create a new fresh copy of entire response. For example first list of countries will be received by browser directly from server and JSON will parse response and create an array of 200 country objects. Second time,

browser will send cached response, but JSON parser will create new array of 200 country objects. CURL will prevent this from happening, this saves parsing time as well.

Lesser JavaScript objects, better performance achieved at client.

Next

Most atom controls has a property called “Next” (written as next). This property can hold reference to any other atom control, or a function or an object. Based on type of object set to next, next will be evaluated and executed after successful completion of logic. For example, when we are posting some information in form, form’s next will be invoked if the form post was successful. Properly chaining control’s next property will allow executing very complex logic in very simple manner. Think of it as a state machine, which executes very easily.

String

If next is a string, current page will be navigated to the string specified.

```
<button atom-type="AtomButton" atom-next="/Home">Email</button>
```

Since we have set next as “/Home” which is interpreted as string, on click of this Button, next will be executed and browser will navigate to the “/Home”.

Script Function

```
    <script type="text/javascript">
        function addToEmail(scope, sender) {
            var list = appScope.resultList;
            list = list.get_selectedItems();
            var results = appScope.emailResults;

            var ae = new AtomEnumerator(list);
            while (ae.next()) {
                results.push(ae.current());
            }
            AtomBinder.refreshItems(results);
            AtomBinder.setValue(appScope.resultList, "selectedIndex", -1);
            appScope.setValue("_viewIndex", 2);
        }
    </script>

<button atom-type="AtomButton" atom-next="{addToEmail}">Email</button>
```

If next is set to a script method (Evaulated between curly braces), the method will be executed when button will be clicked. Function should accept two parameters scope and sender. Sender is an atomControl itself, and scope is sender’s current scope.

#ID

This will set atomControl instance of specified ID. And in next invoke, refresh method of atomControl will be called. Assume we have a control with id “resultList” which will hold results of some URL data. When next will invoke, resultList will refresh the URL Data contents and refresh the list items.

{Object}

When we want to change scope values as well as we want to invoke any method further, we can set object as follow.

{ { scope: { pageID: 1 } , next: addToEmail } } ← double braces, outer braces executes code and inner braces returns an inline object.

This will set scope.pageID to 1 and then it will invoke addToEmail function.

[$scope.someControl]

Every control in current scope can have a name “someControl”, this allows us to refer atomControls directly as $scope.someControl. This is usually just a binding expression, but you can set reference to control by specifying scope.someControl and atom control’s refresh method will be invoked.

Array of items

If I want to execute multiple actions, then I can pass it in an array shown as below.

{ [{ scope: { pageID: 1 } } , addToEmail] }

addToEmail is a global function, and first item in the array is inline object with field scope.

Next Binding

You can bind anything in next as well, following will increment pageID whenever it will be executed.

[{ scope: { pageID: $scope.pageID + 1 } }]

$scope.pageID binds to current value of scope.pageID and it will be incremented everytime it will be executed.

Refresh

Refresh method of an atomControl will invoke some method to refresh its contents or do some more processing as directed. Following controls behaves differently when refresh method is invoked (note, refresh is mostly invoked by atom-next only).

AtomItemsControl

Any control derived from AtomItemsControl, (AtomListBox, AtomToggleButtonBar etc), will refresh “items” binding, which will cause URL binding to refresh and fetch new data from server.

AtomFileUploader

AtomFileUploader will upload the file, if file was specified, otherwise it will execute “next”.

AppScope (written as appScope)

appScope, (Written with first letter small) is scope at application level. Any values contained in appScope are serialized in URL bar with #, so it lets you navigate different scope items. Only properties of appScope that are either string or number are serialized in URL. If you don't want string value to be serialized in URL, then you can prefix property name with "_" underscore.

We use appScope to store global variables accessible throughout the page for binding and we have controls that watches the appScope for changes and updates itself. For example, if you are creating search page, and if you searchQuery to be navigatable (change as back button changes), then you can put it in appScope without underscore.

```
<script type="text/javascript">
    ({
       searchText: '',
       start: 0,
       pageSize: 20,
       _selectedID: 0
    })
</script>
```

This is the script that will set appScope parameters.

```
<input atom-value="$[appScope.searchText]" placeholder="Project" />
```

This input box will store value of text within appScope.

```
<div
       atom-type="AtomListBox"
       atom-dock="Fill"
       atom-items="$URL[Atom.url('/Search/Projects', { text: $appScope.searchText ,
start:$appScope.start, size:$appScope.pageSize})]"
       />
```

This listbox will wait for changes in appScope, and will load list of projects. Now any changes in any variables within appScope will be reflected in URL navigation as below.

If you type "a" in searchText, URL will become "#searchText=a&start=0&pageSize=20", if you type "b" in searchText, URL will become "#searchText=b&size=0&pageSize=20". This will allow you to hit back button and load previous appScope, that will allow you to refresh list based on previous selection. Any property with underscore will be ignored, so if value of _selectedID changes, hash will not change.

Summary of Property Expression

{} Curly Braces

Initial setup, evaluate and apply to individual property. If set to atom-properties, then it should be an associated array and values must be set to individual properties.

[] Square Brackets

Square brackets are used to bind an expression to the target property. Fragments of expression can be bound by prefixing it with the "$" sign as shown below.

[$data.FirstName + ' ' + $data.LastName] will bind to full name of the person. Since we wanted to reduce parsing load on javascript engine, and we wanted to make binding simpler with only regular expression, we decided to prefix binding expressions with $. Here is the reason why.

Binding Parser parses the given string and it looks up for expressions like (data|scope)(.[a-zA-Z0-9_]) , now you can bind to data, or data.FirstName, but when we give an expression like ['data is ' + data.count] , RegEx fails and tries to bind the data within the string. In case like inline object literal [{ data: { FirstName: data.FirstName } }], data on left side is name of property, and what we are interested in binding which is on right side of the colon. In such cases, once again writing RegEx is more complicated. And we didn't want to spend time writing complete JavaScript parser, so we decided to change our binding expression to be of type $(data|scope|appScope|owner)(.[a-zA-Z9-0_])* , so RegEx simply looks for anything after $ to bind.

So in expression [{ data: { FirstName: $data.FirstName } }] , binder recognizes data.FirstName as an expression to bind.

$[] Square Brackets with $ prefixed

Sets up two way binding

$URL[]

Invokes AJAX request and binds to newly available data from the response.

$CURL[]

Same as CURL, but only first time URL will be requested, on subsequent requests, existing response data will be used.

Controls

AtomAutoCompleteBox

This is derived from AtomListBox, and it provides ability to search and select items by typing in the text box.

placeholder

Allows placeholder (watermark) to be set for the textbox.

AtomButton

This control can be applied to image or button.

Default Action

On click on button, atom-next will be executed if specified.

button-icon

Allows you to set icon to button on the left side of button.

AtomCheckBox

is-checked

Boolean property that reflects associated checkbox's checked status.

```
        <input
            atom-type="AtomCheckBox"
            atom-is-checked="$[data.RememberMe]"
            type="checkbox" />
```

This will store "RememberMe" property as true or false based on whether checkbox is checked or not.

AtomCheckBoxList

Derived from AtomItemsControl, it lets you select items by comma separated values displayed in checkbox list.

value

Value property stores comma separated values for selected items.

value-separator

Value Separator is set to "," by default, however you can change this to anything else if you wish.

AtomComboBox

This is derived from AtomItemsControl so all the properties inherit from AtomComboBox, this behaves as typical ComboBox and you can load items from the URL binding on items property. Value property reflects current selected item's valuePath.

AtomDataPager

This control binds to items array, response from server that must contain "total" field that helps in dividing results into pages.

HTML	Binding Expression	JavaScript	
items	items	get_items, set_items	Array of items and total information
current-page	currentPage	get_currentPage, set_currentPage	Current selected page
page-size	pageSize	get_pageSize, set_pageSize	Current page size, default set to 25
total	total	get_total, set_total	Total number of items in the result set (must be sent by server)

AtomDateControl

Typical Year, Month, Day drop down box editor, which sets date as JavaScript date format as value property.

AtomDeleteButton

This is derived from AtomPostButton, and it has "confirm" property set to true.

AtomDockPanel

This control provides docking layout for its children. You have to apply atom-dock property to its children to allow docking to work correctly. AtomDockPanel will only function if it is fixed to size. So AtomApplication derived from AtomDockPanel will set itself to available max size in HTML Body.

style-width is required for docking to Left or Right
style-height is required for docking to Top or Bottom

AtomFileUploader

AJAX file uploading is not available by default on all browsers, so AtomFileUploader wraps functionality around flash player object and provides you AJAX uploading along with progress.

fileTypes

You can set an array of description and extensions to filter the file types as shown below.

```
atom-file-types="{ [ { description:'Photos', extensions:'*.jpg;*.png;*.bmp; ' } ] }"
```

Curly braces will evaluate the expression and it will set it immediately, this is not binding.

value

True if file is selected.

url

Url to where file should be uploaded. Please note, cross site restrictions apply here and you will have to set site policy and options for HTML5 if you are going to upload to different host.

AtomForm

AtomForm control will automatically layout child elements along with labels right justified and displayed correctly to form a proper form shape without adding anymore styles.

mergeResult

By default, true, it will merge result of AJAX POST to “data” of current form. This is usually important when we want to display some errors or turn few options after posting back.

postUrl

Url where the “data” of the form will be posted to, by default “data” will be encapsulated in JSON format within “formModel” field in order to maintain consistency of data sent, since normal POST will result in null value sent as “null” which will result in incorrect string processing on server side.

successMessage

If set, it will be displayed in an alert after the AJAX post was successful.

AtomItemsControl

allowMultipleSelection

Default: false, if set to true, it will allow multiple selections to be made. Value property will contain comma separated (can be customized), text representation of selected items.

allowSelectFirst

Default: false, if set to true, it will automatically select the first item when items array has been set. This property is true for AtomToggleButtonBar, as toggle button bar always has one selection.

AtomLinkBar

This is derived from AtomToggleButtonBar and it highlights current URL automatically.

AtomListBox

AtomListBox is derived from AtomItemsControl and it inherits all public properties and behavior as well.

autoSelectOnClick

It is true by default. If set to false, item will not be selected on event of click. To select item, selectCommand must be set to next for $scope.__owner.

items

This property will hold array of items and it will change its children whenever there is change in source array. AtomListBox will create new child item for every item that exists in source array and recreate/change data of every item whenever there is change.

selectedIndex

This holds index of selected item, this is a bindable property and it is inherited from AtomItemsControl.

selectedItem

This holds current selected Item or first item in selectedItems array. If items array is set, then this will contain the data item of selected ui element, otherwise it will contain reference to selected ui element.

label

labelPath

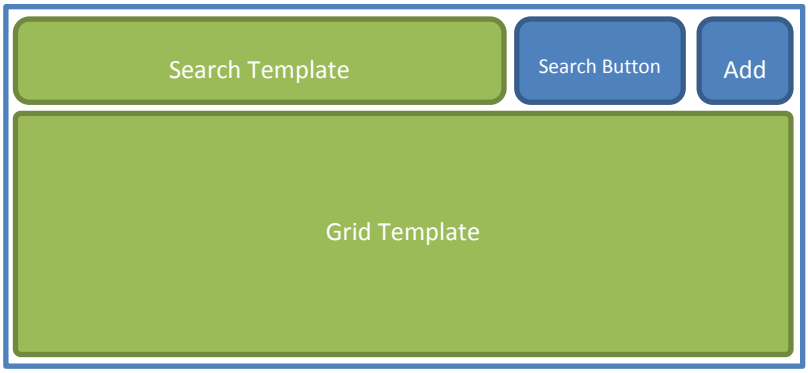
valuePath

value

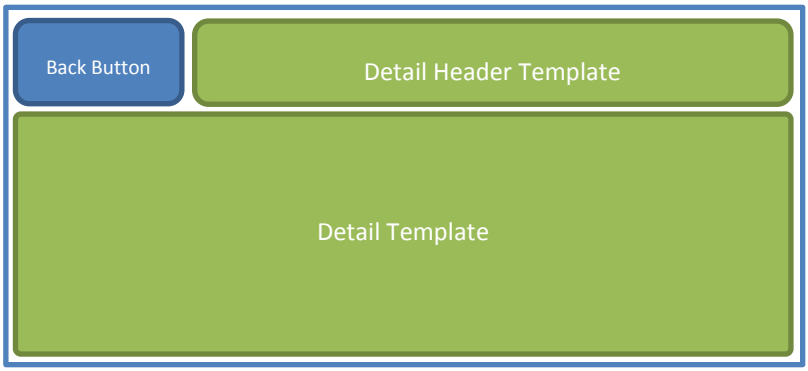
AtomNavigatorList

This control is a complex composite control, comprises entire CRUD (Create, Retrieve, Update and Delete) UI mechanism.

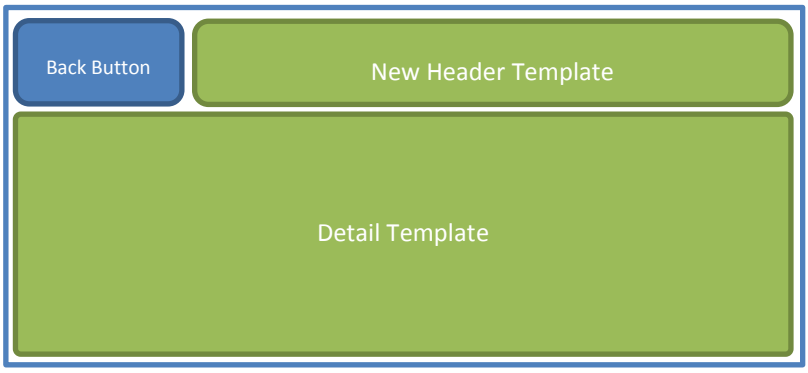
1st View (Grid View)



2nd View (Detail View) opens when item is selected.



3rd View (Add View) opens when Add button is clicked



Following templates are available to customize AtomNavigatorList.

1. GridTemplate (Required), displays table of items
2. SearchTemplate (Optional)
3. DetailTemplate (Required), displays full view of selected item
4. DetailHeaderTemplate (Optional), displays header on the top of selected item's full view
5. NewHeaderTemplate (Optional), displays header on the top of new item's full view

Please note, DetailTemplate is same for both, editing an existing item or adding a new item.

AtomPostButton

AtomPostButton control initiates AJAX POST operation to specified postUrl when clicked. It uses postData or data based on whichever is available to post as AJAX post. And after the successful result, it will invoke next if set.

postData

AtomPostButton will post an object set at postData to specified postUrl. You can set binding on this property. If this is null or undefined, AtomPostButton will post existing data found in parent hierarchy.

postUrl

Url to post data to.

confirm

If set to true, it will ask confirmation from user before posting.

confirmMessage

Confirmation message displayed for confirming before posting.

successMessage

Success message that will be displayed in an alert box after the post was successful.

Library

Atom

url(baseUrl,parameters)

Atom.url method will encode url along with query string from given parameters in string and JavaScript Object literal format.

tableLayout(columns,cellWidth,cellHeight)

Atom.tableLayout will apply tabular layout to element for layout children in newspaper column layout.

AtomDate

toShortDateString

AtomDate.toShortDateString converts string date or JavaScript Date object into DD-MMM-YYYY format as 2-FEB-2012.

```
<div atom-text="[AtomDate.toShortDateString($data.ExpiryDate)]"></div>
```

AtomPhone

toSmallPhoneString(val)

This will format phone number in +1.800.800.800 format.

toPhoneString(val)

This will format phone number in +1.800.800.800 (ext) (msg) format if extension and msg did exist in the provided value.