Web Atoms

CREATING RICH INTERNET WEB APPLICATIONS WITH WEB ATOMS

Akash Kava
HTTP://WEBATOMSJS.COM

1 CONTENTS

2 WEB ATOMS LICENSE

This section is for reference only, for latest updates on License Agreement and Terms, please visit https://webatomsjs.neurospeech.com/

NOTICE: READ THIS LICENSE AGREEMENT CAREFULLY BEFORE INSTALLING, ACCESSING, COPYING OR USING THE SOFTWARE ACCOMPANYING THIS AGREEMENT. CLICKING ON THE "I ACCEPT" BUTTON BELOW, OR IN ANY OTHER WAY INSTALLING, ACCESSING, COPYING OR USING THE SOFTWARE, CREATES A LEGALLY ENFORCEABLE CONTRACT AND CONSTITUTES ACCEPTANCE OF ALL TERMS AND CONDITIONS OF THIS AGREEMENT WITHOUT MODIFICATION.

This License Agreement ("LICENSE") is a legal agreement between you (either an individual or a single entity, also referred to as ("LICENSEE", "YOU") and NeuroSpeech Inc. ("NeuroSpeech"), for the software containing this LICENSE which may also include the software's source code written in a high–level computer language, associated media, printed materials, and "online" or electronic documentation (collectively referred to as "SOFTWARE"). The parties agree as follows:

1. LICENSE

Web Atoms License refers to right to use Web Atoms scripts into your application hosted or used in any other way in the context of a browser that is capable of rendering html and executing javascript.

Web Atoms License on domain name refers to HTTP Host Header, for every domain or subdomain, new license must be taken. Same license can only be used on other domain or subdomain if it is used for backup purpose or for mirroring exact same application.

License is applicable on any HTML text rendered by browser, offline, online or loaded through text with content type as html.

1.1 Licensing (License Type)

(a) Attributed License (Commercial as well as Non Commercial) With Link to Web Atoms

License is granted for one domain or application under following conditions.

Licensee must put a link (anchor tag) at bottom right corner of page by writing "Powered by Web Atoms" and must link "Web Atoms" to http://webatoms.neurospeech.com as shown in the following picture. The link must be human visible at the bottom of the application. Any tricks to hide or move it out of human view is not allowed.

Education or Non Profit Applications must use word "Sponsored by" instead of "Powered by".

This link must be placed in every page, wherever Web Atoms scripts are used under free license.

Your website or application must be registered with public Web Atoms Directory hosted at http://webatoms.neurospeech.com/directory, registration is free.

(b) Non Commercial Open Source License

Eligibility

Your project must be licensed under GPL, Apache, MIT or BSD License and must be a noncommercial license. Requires Pre-Written Approval from us.

(c) Yearly Subscription Licenses

(1) Site Subscription

Single Web Site on Multiple Web Server or CDN

(2) Server Subscription

Unlimited Web Sites on Single Server

(d) Site Perpetual License

(1) Site License

Single Web Site served from Single Server

(2) Load Balancer License

Single Web Site served from 10 Web Servers for Load Balancing

(3)Enterprise Site License

Single Web Site served from any number of servers or CDN

If you want to purchase and use web atoms license for unlimited time, then you need Commercial license which is available per Application or per Domain. Application area where users login and perform an activity, the application boundary is defined by following conditions.

In case of website, Application is One Domain Name. Domain Name includes sub domain as well, so multiple subdomains are referred as multiple applications.

Application can have multiple aliases, for example in.neurospeech.com is a website for Indian Customers and us.neurospeech.com is a website for US Customers, but both actually perform same operations on same set of users just distinguished by different region. This refers as aliases.

Application defines set of operations which one user can perform after login in one session.

If Applications use Single Sign On or Federated Login, or any other means of sharing authentication info across different domain names, each application is a separate application even though users are automatically authenticated.

In case of Installed/Local/Intranet Application, Application which is served by one

License Grant

License is granted per Application or per Domain.

License is granted only for specific version.

Application License includes one year of free upgrades.

2. SUPPORT SERVICES

2.1 Support

Licensor will use commercially reasonable efforts to provide Licensee with Software maintenance and support in accordance with its standard practices (as amended from time to time, Support Services). Licensor shall have no obligation to support any version other than the then current and immediate prior version. Licensee agrees that Licensor may charge in accordance with its then current policies for any support services resulting from (a) problems, errors or inquiries relating to any hardware, system, service or other software or (b) use of any unsupported version of the Software. As part of your Developer License you are entitled to NeuroSpeech's "Standard" Support Package for a limited period of time, which guarantees an initial response to (but not necessarily a complete resolution of) your issue within forty eight (48) hours during business days.

2.2 Updates

Service Packs and Major Version updates are available in accordance with the terms set forth a at http://neurospeech.com/wsclient/support.html

You may use the resulting upgraded product only in accordance with the terms of this LICENSE.

3. SOURCE CODE

(a) As part of the DEVELOPER LICENSE WITH SUBSCRIPTION AND SOURCE CODE you are granted a non-exclusive, non-transferable, non-sublicensable, revocable and limited license to access, use, copy and modify the Source Code, including the right to may make modifications, enhancements, derivative works and/or extensions ("Modifications") to the SOFTWARE or Source Code utilizing any such Source Code, subject to terms provided in this section,

(b) Under no circumstances may any portion of the SOFTWARE's source code or any modified version of the source code be distributed, disclosed or otherwise made available to any third party,

(c) You may not distribute the Source Code, or any Modifications to the SOFTWARE or Source Code, in source code form,

(d) NEUROSPEECH DOES NOT provide technical support for modified source code,

(e) The SOFTWARE's source code is provided "AS IS". Refunds are not available for licenses that include source code,

(f) You acknowledge that the SOFTWARE's source code contains valuable and proprietary trade secrets of NEUROSPEECH. You also acknowledge that all individuals employed by or belonging to your entity agree to expend every effort to insure its confidentiality. You agree to assume full responsibility for such employees' or contractors' use, or misuse, of such disclosed source code as if it was your use. These obligations shall not apply to any information generally available to the public, independently developed

or obtained without reliance on NEUROSPEECH information, or approved in writing for release by NEUROSPEECH without restriction.

4. REDISTRIBUTION

You may distribute the SOFTWARE (or, where applicable, the run–time portion of the SOFTWARE) provided that:

· You distribute the SOFTWARE, in object code form only, as part of solutions for internal company use, hosted applications, commercial solutions deployed at end–users' sites or shrink–wrapped software (PACKAGED PRODUCTS) in which the SOFTWARE is integrated.

· You ensure that the SOFTWARE is not distributed in any form that allows it to be reused by any application other than your solution. Technical guidelines will be provided upon request, contact support@neurospeech.com for more details.

· You duly inform your customers that they are not allowed to use the SOFTWARE independently from your solution. For use of the SOFTWARE in design–time (i.e. within a development environment such as Microsoft Visual Studio) your customers need to purchase the appropriate number of Developer Licenses from NeuroSpeech.

· You include a valid copyright message in your solution in a location viewable by the end–users (e.g. "About" box).

· You assume full responsibility for your customer's use of the SOFTWARE and must ensure that NeuroSpeech has no obligation to such customer or liability for such customer's use of the SOFTWARE.

All product licenses granted by you to your end user customers in accordance with the terms of this agreement are perpetual and royalty–free.

5. U.S. EXPORT REGULATIONS

This Software product is subject to export restriction under U.S LAWS, as this software may contain encryption technology. Prior written authorization from the U.S. government is required for direct or indirect exports and re-exports of NEUROSPEECH Product and Technology to any country embargoed or restricted by the U.S. Currently, the embargoed countries are Cuba, Iran, Libya, North Korea, Sudan, and Syria.

And also direct or indirect exports and re-exports are denied to the parties whose name are there in the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Department of Commerce Denied Person's List or Entity List.

Licensee bears all responsibility for export law compliance and will indemnify NEUROSPEECH against all claims based on Licensee's exporting the Software.

For more information regarding export regulations, please visit the Bureau of Industry and Security web site at http://www.bis.doc.gov

5. CONFIDENTIALITY

5.1 Scope

The term Confidential Information means all trade secrets, know-how, software and other financial, business or technical information of Licensor or any of its suppliers that is disclosed by or for Licensor in relation to this Agreement, but not including any information Licensee can demonstrate is (a) rightfully furnished to it without restriction by a third party without breach of any obligation to the Licensor, (b) generally available to the public without breach of this Agreement or (c) independently developed by it without reliance on such information. All Software and Documentation is Confidential Information.

5.2 Confidentiality

Except for the specific rights granted by this Agreement, Licensee shall not possess, use or disclose any Confidential Information without Licensor's prior written consent, and shall use reasonable care to protect the Confidential Information. Licensee shall be responsible for any breach of confidentiality by its employees.

6. PROPRIETARY RIGHTS

6.1 Restrictions.

Licensee shall not (a) use any Confidential Information to create any software or documentation that is similar to any Software or Documentation,

(b) disassemble, decompile, reverse engineer or otherwise try to discover any source code or underlying structures, ideas or algorithms of the Software (except and only to the extent these restrictions are expressly prohibited by applicable statutory law),

(c) Encumber, lease, rent, loan, sublicense, transfer or distribute any Software,

(d) Copy, adapt, merge, create derivative works of, translate, localize, port or otherwise modify any Software or Documentation,

(e) use the Software, or allow the transfer, transmission, export or re-export of all or any part of the Software or any product thereof, in violation of any export control laws or regulations of the United States or any other relevant jurisdiction or

(f) Permit any third party to engage in any of the foregoing prescribed acts. Licensee shall not use the Software for the benefit of any third party (e.g., time-share or service bureau arrangement) without Licensor's prior written consent, at its discretion.

6.2 No Implied License

Except for the limited rights and license expressly granted hereunder, no other license is granted, no other use is permitted and Licensor (and its suppliers) shall retain all right, title and interest in and to the

Software and Documentation (and all patent rights, copyright rights, trade secret rights and all other intellectual property and proprietary rights embodied therein).

6.3 Markings

Licensee shall not alter, obscure or remove any trademark, patent notice or other proprietary or legal notice displayed by or contained in any Software, Documentation or packaging.

6.4 Third Party Software

The Software may operate or interface with software or other technology (In-Licensed Code) that is in-licensed from, and owned by, third parties (Third Party Licensors). Licensee agrees that

(a) It will use In-Licensed Code in accordance with this Agreement and any other restrictions specified in the applicable license set forth or referenced in the Documentation,

(b) No Third Party Licensor makes any representation or warranty to Licensee concerning the In-Licensed Code or Software and

(c) No Third Party Licensor will have any obligation or liability to Licensee as a result of this Agreement or Licensee's use of the In-Licensed Code.

7. WARRANTY DISCLAIMERS

THE SOFTWARE AND SUPPORT SERVICES ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. LICENSOR DOES NOT WARRANT THAT THE SOFTWARE OR SUPPORT SERVICES WILL MEET LICENSEE'S REQUIREMENTS OR THAT THEY WILL BE UNINTERRUPTED OR ERROR-FREE. TO THE FULLEST EXTENT PERMITTED BY LAW, LICENSOR HEREBY DISCLAIMS (FOR ITSELF AND ITS SUPPLIERS) ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, ORAL OR WRITTEN, WITH RESPECT TO THE SOFTWARE AND SUPPORT SERVICES INCLUDING, WITHOUT LIMITATION, ALL IMPLIED WARRANTIES OF TITLE, NON-INFRINGEMENT, QUIET ENJOYMENT, INTEGRATION, MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE AND ALL WARRANTIES ARISING FROM ANY COURSE OF DEALING, COURSE OF PERFORMANCE OR USAGE OF TRADE.

NEUROSPEECH warrants solely that the SOFTWARE will perform substantially in accordance with the accompanying user documentation for a period of ninety (90) days. NEUROSPEECH does not warrant that use of the SOFTWARE will be uninterrupted or error free at all times and in all circumstances, nor that program errors will be corrected. This limited warranty shall not apply to any error or failure resulting from (i) machine error, (ii) LICENSEE’s failure to follow operating instructions, (iii) negligence or accident, or (iv) modifications to the SOFTWARE by any person or entity other than NEUROSPEECH. In the event of a breach of warranty, LICENSEE‘s sole and exclusive remedy, is repair of all or any portion of the SOFTWARE. If such remedy fails of its essential purpose, LICENSEE‘s sole remedy and NEUROSPEECH’s maximum liability shall be a refund of the paid purchase price for the defective SOFTWARE only. This limited warranty is only valid if NEUROSPEECH receives written notice of breach of warranty no later than thirty (30) days after the warranty period expires. EXCEPT FOR THE EXPRESS WARRANTIES SET FORTH IN THIS LICENSE, NEUROSPEECH DISCLAIMS ALL OTHER WARRANTIES, EXPRESS

OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF TITLE, NONINFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

8. LIMITATION OF LIABILITY

IN NO EVENT SHALL LICENSOR BE LIABLE UNDER ANY CONTRACT, NEGLIGENCE, STRICT LIABILITY OR OTHER LEGAL OR EQUITABLE THEORY FOR ANY SPECIAL, INCIDENTAL, PUNITIVE, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, INCLUDING DAMAGES RESULTING FROM INTERRUPTION OF USE, LOSS OR CORRUPTION OF DATA, OR LOST PROFITS, WHETHER OR NOT LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY REMEDY, ARISING OUT OF OR IN CONNECTION WITH THIS AGREEMENT. IN NO EVENT SHALL LICENSOR'S TOTAL AND AGGREGATE LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT OF THE LICENSE FEES PAID BY LICENSEE HEREUNDER.

9. TERM AND TERMINATION

9.1 Term

This Agreement shall commence on the Effective Date and continue in effect until terminated as provided herein.

9.2 Termination.

This Agreement shall automatically terminate without further action by any party, immediately if Licensee fails to comply with the terms of this agreement. NEUROSPEECH reserves the right to discontinue at any time. However, NEUROSPEECH will fulfill any contractual obligations to provide support for discontinued products that exist as of the date of discontinuation. Upon termination, you agree to destroy the SOFTWARE, including all accompanying documents and copies.

10. GENERAL PROVISIONS

10.1 Entire Agreement

This Agreement constitutes the entire agreement, and supersedes all prior negotiations, understandings or agreements (oral or written), between the parties about the subject matter of this Agreement. Terms set forth in Licensee's Purchase Order (or any similar document) that are in addition to or at variance with the terms of this Agreement are specifically waived by Licensee. All such terms are considered to be proposed material alterations of this Agreement and are hereby rejected. No waiver, consent or modification of this Agreement shall bind either party unless in writing and signed by the party against which enforcement is sought. The failure of either party to enforce its rights under this Agreement at any time for any period will not be construed as a waiver of such rights. If any provision of this Agreement is determined to be illegal or unenforceable, that provision will be limited or eliminated to the minimum extent necessary so that this Agreement will otherwise remain in full force and effect and enforceable.

10.2 Governing Law

This Agreement shall be governed by and construed in accordance with the laws of the United States of America, without regard to its conflicts of law provisions.

The United Nations Convention on Contracts for the International Sale of Goods does not apply.

10.3 Acknowledgment

Licensee acknowledges that:

(a) It has read and understands this Agreement,

(b) It has had an opportunity to have its legal counsel review this Agreement,

(c) This Agreement has the same force and effect as a signed agreement,

(d) Licensor requires identification of the User and Licensee before issuing this license and

(e) Issuance of this license does not constitute general publication of the Software or any other Confidential Information.

3 WHAT IS SINGLE PAGE APPLICATION?

3.1 History of HTML and Web Apps

Single Page Application refers to typical Client Server communication between browsers (Client) and HTTP Web Server. Earlier, technologies like ASP, PHP, JSP and recent ASP.NET used to serve HTML and that was rendered by client without much of processing. The first version of browser itself was very read-only book, where you could only read and navigate the contents of page. Then came along JavaScript, where little bit of data validation came in picture to verify the contents of form before posting it to server. Still major logic processing was performed on server. Nasty if-else based logic on server side scriptlets like ASP or PHP makes code so complex, it becomes nightmare to manage apps written on them. Not only that, more processing server requires more server resources for increasing number of users and slowing down the speed of application.

3.2 Speed of Internet

We started with few kilobytes of speed and we are now at few megabytes per second speed, however our hunger of speed always exceeds available speed due to availability of multiple tabs on browser, multiple websites to look at simultaneously with power of multitasking. Buying more servers just isn't enough to meet demands of speed for web apps.

3.3 UI Processing on Server

Most programming languages like ASP, PHP, JSP or PERL etc all process major logic regarding HTML elements on server. So server is busy calculating whether to hide or show an element based on some business logic for all 1000 users at once, where else CPU on all 1000 users is idle waiting for response from server. In older Client Server systems, Server only serves data after validating the request, and Client does UI processing on Client Machine. Unfortunately, HTML didn't adopt this pattern until arrival of AJAX, however major platforms are still doing it wrong, even in AJAX, and they are sending partial HTML elements.

3.4 Client Server Architecture

A true Client Server Architecture is the one, in which UI (HTML) stays at the client. And only data, in the form of JSON or any other compatible text format like CSV, is loaded from server. This reduces CPU activity on server, allowing it to serve more clients. And since each client (Browser) is now capable of running HTML5 and JavaScript, there are new frameworks evolving to support this paradigm.

3.5 Frameworks

So need of Client Server programming (now referred as Single Page Applications) created need of Frameworks that allows us to process UI at client. 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.

3.6 MVC vs. Component Oriented Development

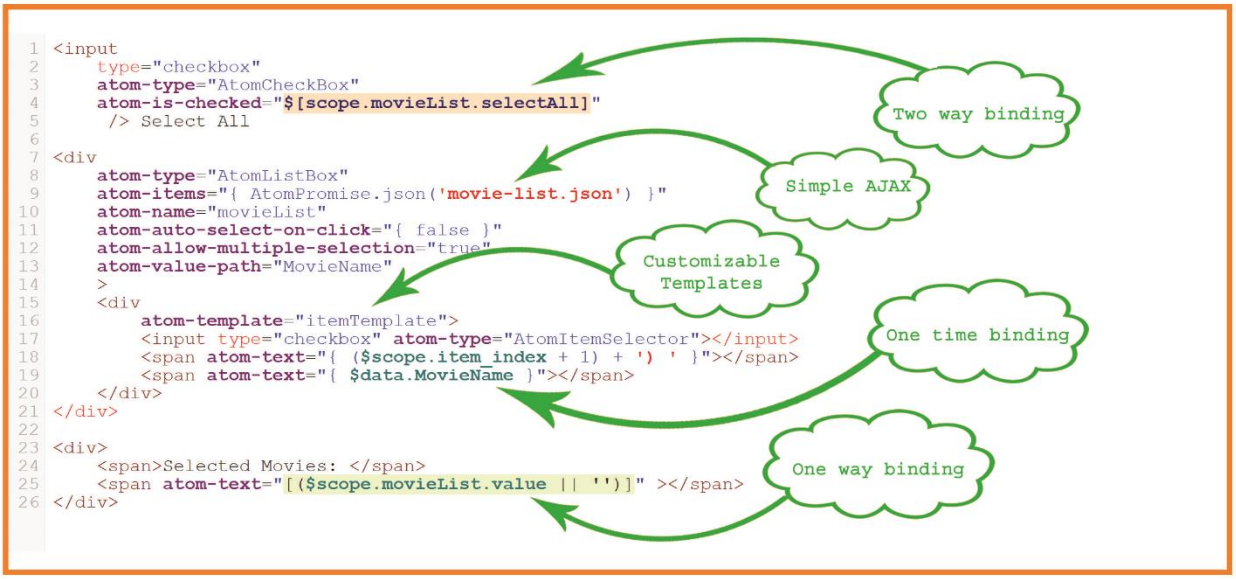
This section might hurt many programmers, as many programmers believe MVC is great way to create apps. For smaller apps, it seems easy, and for highly large complex apps it looks perfect solution. But for Mid-Sized business apps, MVC is little bit complicated when it comes to manage and maintain code. MVC is good where application has single direction of flow, Input -> Process -> Output. However for interactive UI it is not very helpful. Though Component Oriented Development does encapsulate MVC, but it turns out that Component has set of properties that refers model, visual template that represents view and a self-contained logic that Component performs and updates its own model that represents controller.

The most successful UI development platforms, Visual Basic, Adobe Flex, Windows Forms, Java Beans, Swings and many similar were focused more on Component Oriented Development (Custom Controls & User Controls).

Features of Component Oriented Development

1. Ability to Visualize User Interface at time of Development
2. Advanced UI Editors support Drag and Drop of Components
3. You can safely pack a logic in a Custom Components that other developers can use without affecting their or your code
4. Controls can raise Events for notifying other components
5. Properties of Control lets you choose particular pre coded behavior
6. Visual Appearance of Control can be customized with Templates

4 CORE ARCHITECTURE



4.1 Object Oriented Design

Over number of different Object Oriented Design approach in JavaScript, Web Atoms was designed using Flat Prototype Inheritance to avoid chain lookups. Most widely used inheritance pattern of prototype chaining takes longer time to resolve methods in base classes. This adds up to performance and leads to more CPU consumption. We have carefully chosen flat prototype pattern, in which we explicitly copy every base class prototype into current prototype if prototype does not contain same named method. As per your convenience you can modify this behavior as this is the 2nd global method designed inside Web Atoms JavaScript file. Each prototype carries `__typeName` and `__baseType` properties which you can use to inspect objects.

4.1.1 Static Linking

All classes are organized in inheritance order, so the base class comes first in the script and inherited class comes after. This design reduces setup time as the hierarchy is built and it is correct. Dynamic linking requires invoking complex logic in finding references and going through unnecessary huge iterations.

Each class is coded as a separate JavaScript file, and before build, we have a text template processor, which organizes Base Class references and combines all JavaScript files into One Web Atoms JavaScript file.

4.1.2 Debug Script

Debug version of Web Atoms JavaScript is nothing but better readable JavaScript version as well as it contains File Name and Line Number of individual JavaScript File. As managing JavaScript project was becoming very difficult, we had to divide the framework in number of smaller files and combine them in proper inheritance hierarchy.

```
/*Line 230 - 'AtomItemsControl.js' */    set_filter: function (f) {
/*Line 231 - 'AtomItemsControl.js' */        if (f == this._filter)
/*Line 232 - 'AtomItemsControl.js' */            return;
/*Line 233 - 'AtomItemsControl.js' */        this._filter = f;
/*Line 234 - 'AtomItemsControl.js' */        this._filteredItems = null;
/*Line 235 - 'AtomItemsControl.js' */        this.applyFilter();
/*Line 236 - 'AtomItemsControl.js' */    }
```

First line indicates that it is Line 230 of AtomItemsControl.js file. This helps in debugging huge code base easily and manage classes in individual smaller files.

4.2 Property Inference

Since JavaScript did not have a default property syntax, we had to come up with a property inference that should be consistent with member and method. And the most important part, we wanted to use function as property value as well. This means, in Model View Commands pattern, we can set function delegates as property which can be used later on to invoke them on any certain event.

Here is how we evaluate property,

```
    getValue: function (target, key) {
        if (!target)
            return null;
        var f = target["get_" + key];
        if (!f)
            return target[key];
        return f.apply(target);
    }
```

And here is how we set the property,

```
    setValue: function (target, key, value) {
        if (!target || value === undefined)
            return;
        var oldValue = AtomBinder.getValue(target, key);
        if (oldValue == value)
            return;
        var f = target["set_" + key];
        if (!f)
            target[key] = value;
        else
            f.apply(target, [value]);
        this.refreshValue(target, key, oldValue, value);
    }
```

In both examples, we first search if get_/set_ functions exist on the target, then we invoke them, otherwise we set the value as a direct member assignment. We also check if property value is actually different then current value or not, this helps in preventing infinite loop of changes.

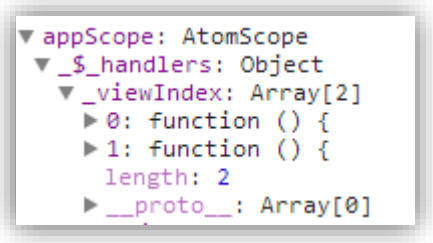
4.2.1 Trouble with Single Method Property Pattern

Typical single method property pattern such as the one found in jQuery or similar frameworks does not work well with evaluating mixed property patterns. For example, to evaluate a.b, b could me a single method property function or it could be a member storing delegate to some other function.

Web Atoms is built on Commands Pattern, where Commands are delegates (function with closures) which are invoked on certain event. However, Property Evaluator is not supposed to invoke these delegates, they are simply supposed to store these delegates.

All Atom Controls have properties defined as get_/set_ methods. So Property Evaluator can easily distinguish between Command Delegates and Properties.

4.3 Distributed Binding



To achieve high performance in Data Binding, we wanted to reduce overhead in updating HTML elements per change. So we created event handlers in target object itself under hidden variable `_$_handlers`. This member then stores all delegates to bound objects which should listen to change of particular property. So if one object's one property is updated, only event listener for that particular property are intimated, for same object, if there are other property change listeners, they will not be intimated. This increases performance with very little cost of more memory.

This gives us freedom in creating many objects within one object without worrying about performance overhead. Most platforms tell you not to put more things in scope, Web Atoms does not impose such restrictions.

4.4 Enumerator Pattern

jQuery or similar enumerator pattern imposes lots of complex closures and makes logic look little different than that of earlier object oriented approach. So we created class `AtomEnumerator`, which has next and current methods as it is available in different programming languages such as Java and C#.

Enumerator pattern has advantages, as it does not need inline functions, and logic appears very clear to the reader. And we can still use "this" as expected "this" in object oriented programming construct. jQuery etc. breaks such constructs as it changes the meaning of "this" inside the inline function. However, it becomes more complicated to debug and perform simpler logic and also increases closures.

Enumerator Pattern is simple, and we can reduce number of iterations when needed.

4.5 Child Enumerator

Using jQuery or querySelector equivalent turns out to be an expensive operation, as it enumerates everything node, first to query and collect the result and second to actually enumerate the results to process our operation. Web Atoms heavily depends upon child enumeration for creating and organizing the UI hierarchy. To improve speed, we navigate the DOM as graph and we walk and perform operations simultaneously to reduce number of iterations.

4.6 HTML Element Template

In Web Atoms, all controls depend heavily on templates to customize user interface. Most of the times, we think that changing CSS is sufficient for changing the display, but some complex logic such as putting checkbox for selection or image for preview and little complex table structure to display data, all these are only possible with HTML Element Template. Instead of creating and parsing templates, Web Atoms use `cloneNode` method of HTML Element to create new elements quickly.

4.7 Control Life Cycle

Every control is derived from `AtomControl` and it has three pass creation life cycle, Create, Initialize and Update UI. Three pass means, your entire DOM is navigated thrice, however this may seem little performance overhead, but this is necessary to ensure smooth creation as well as cascaded bindings.

1. First Constructor is Called for the Control
2. createChildren method is called on Control
    a. createChildren of base class is called.
    b. Create Children enumerates each child and removes templates and saves presenters
    c. Removing templates from node hierarchy ensures templates will not be initialized unless needed and will not affect execution. Template will be stored in current object with name.
    d. Constructor of each child control will be invoked and Create Children will be called on each child.
3. Initialize Method is called on Control.
    a. In which, base class's initialize method is called.
    b. Scope is initialized as per hierarchy and local-scope attribute.
    c. Name is applied.
    d. Initialize Properties method is queued.
        i. Initialize Properties enumerate every child and queues Initialize Method for each child.
    e. initializaionComplete is queued, in which.
        i. onCreationComplete is called for current Control
        ii. updateUI is called on current control

5 SCRIPT AND CSS INSTALLATION

Before adding Web Atoms related resources in HTML HEAD Section, you must download jQuery version 1.8.2 onwards and include them in your HTML HEAD Section before adding any of Web Atoms resources.

Web Atoms need only two resources,

- web-atoms.js or web-atoms-debug.js JavaScript file in HTML HEAD Section
- web-atoms.css Style Sheet in HTML HEAD Section

Please make sure you comply with the licensing of Web Atoms for every page that adds these two resources. Any custom controls or your application related scripts must appear after these two resources.

5.1 No Minimizer

Please note, web-atoms.js is already minimized to maximum extent possible, since Web Atoms uses reflection (to inference properties based on get_/set_ methods), running minimizer can destroy all the logic and Web Atoms will not function.

5.2 Customize CSS

Web Atoms CSS file defines default look and feel of Web Atoms controls, you can definitely override them in your custom CSS or replace this CSS completely. However, before replacing, you must take care that all required Web Atoms Style Selectors are coded properly to make them work right.

```
<!-- Download and Comply with jQuery Terms independently -->
<script src="/Scripts/jquery-1.8.2.js"></script>
<!-- Web Atoms JavaScript -->
<script src="/Scripts/web-atoms-debug.js"></script>
<!-- Web Atoms Style Sheet -->
<link href="/Style/web-atoms.css" rel="stylesheet" />
```

You are allowed to rename the files and organize them as per your needs and coding conventions. But each page importing Web Atoms JavaScript and Style Sheet must comply with Web Atoms Licensing policies.

5.3 Visual Studio Syntax Colorizer

Editing Web Atoms files are little difficult without proper syntax color that represents different form of binding expressions. You can download and use free “Web Atoms Syntax Highlighter” from this URL.

http://visualstudiogallery.msdn.microsoft.com/ecd49e57-2348-4e4c-90c8-9d21d1d50a1e

5.4 Google Chrome Extension

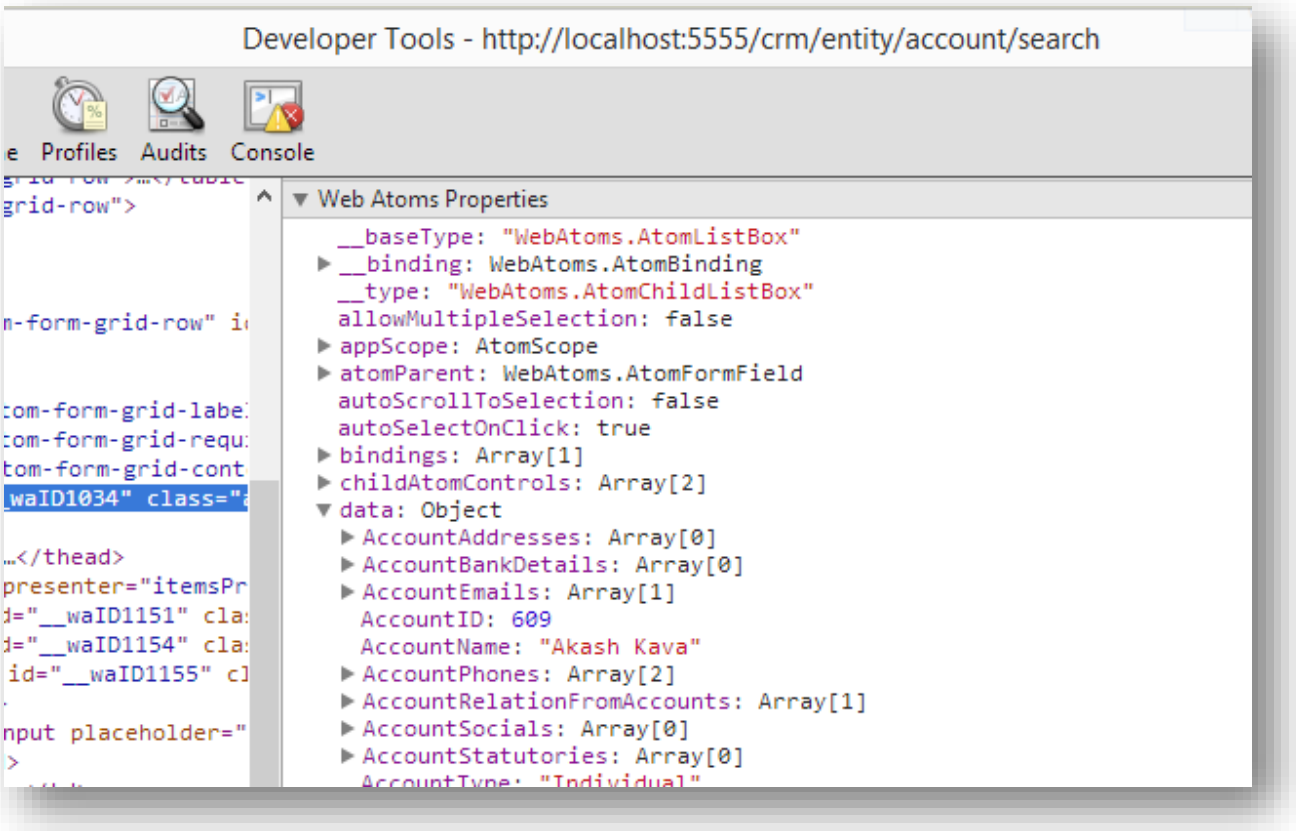
We have created Web Atoms Properties Pane to inspect `AtomControl` associated with the element and inspect its properties. Since `AtomControl` can host many elements and can host bindings for many element, you have to travel up words in hierarchy to inspect associated `AtomControl` and its properties. However most of the time, data property is inherited so finding data property can be time consuming for debugging purposes.

We have created Google Chrome Extension available at Google Chrome Web Store at following URL.

https://chrome.google.com/webstore/detail/web-atoms-properties-pane/mboefdeomkdlbekljcecbfimmecffgje?hl=en-US

Or you can search for “Web Atoms Properties Pane” in google chrome web store.

Once you install, you will see Web Atoms Properties Pane in Chrome Developer Tools after you right click on an element and click on “Inspect Element”.



6 ATOM CONTROL

Control is a class (with respect to OOPs) containing set of methods and event handlers to modify behavior of attached HTML Element. We have created set of controls that gives you ability to perform certain standard operations such as list, docking and view stack. View stack can be thought of Tab Control, you can customize look and feel of controls to make them look like any way you want.

Atom Control also applies set of default styles, and provides ability to customize style for all controls as well as just one control.

6.1 Initialize Atom Control

First attribute of element must be `atom-type` if you want to initialize it as a control. Controls define special properties that are now available with the element. And you can set them as follow.

As shown in the example below, we have associated `AtomControl` with div and now it can perform data binding as we need.

```
<!-- 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>
```

6.2 Control Name

To resolve `id` issue, and to name controls independently in different scope, Web Atoms stores instances of controls in current scope with `atom-name` value specified. Id in HTML has to be unique, but scope control names have to be unique only in one scope. You can have same name in different scopes. Check out `localScope` example for more details.

With `$scope.controlName.propertyName`, you can access other sibling control properties in same scope. You can access them from other scope such as `parent`, `localScope` or `appScope` as well.

6.3 Inherited Data Property

`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. If `data` property is not set then it is 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.

6.4 Owner Property

Child elements of `AtomControl` can access other properties of control by using `$owner.property`. Usually all properties are only accessed through `$owner.property` syntax, but to make it easy, we have defined shortcuts for `$owner.data` as `$data`, `$owner.scope` as `$scope` and `$owner.appScope` as `$appScope`.

6.5 Binding Html Attributes

To bind any attribute on any element, you can simply use binding expression with prefix `atom-` that will set corresponding attribute on the element. Different types of binding (One Time, One Way and Two Ways) will be explained in binding chapter.

```
<a  atom-href="{$data.Url}"
    atom-target="{$data.Target}">
    <img atom-src="{$data.ImageUrl}" />
</a>
```

Property Inference first checks if any `get_`/`set_` method exists on `AtomControl` or not, otherwise it simply sets the attribute. However, both elements `a` and `img` do not have `AtomControl` associated with it, so they will simply have attributes set. First preference is given to `AtomControl` properties.

6.5.1 Text Attribute

To set, internal Text (`innerText` or `textContent` or jQuery's `$(e).text('text')`) of JavaScript, you can set `atom-text` property. This sets text for current element.

```
<span atom-text="{ 'Test' }"></span>
<!-- Gets Converted to -->
<span>Text</span>
```

And by doing this, `atom-text` is now bindable, so you can combine any expression in curly braces or square braces and bind the text of current element.

6.5.2 Is Enabled Attribute

To enable or disable any element based on a Boolean condition you can use `atom-is-enabled` attribute on any element for binding as shown below.

```
<input
    type="checkbox"
    atom-checked="$[data.IsDescriptionAvailable]"
<input
    type="text"
    atom-is-enabled="[$data.IsDescriptionAvailable]"
    atom-value="$[data.Description]"/>
```

6.5.3 Checked Attribute

To use checkboxes to perform binding to Boolean data, you can use `atom-checked` attribute as shown below. Checked can only be used with input with type checkbox.

```
<script type="text/javascript">
    ({
        termsAgreed: false
    })
</script>

<input
    type="checkbox"
    atom-checked="$[scope.termsAgreed]"/>

<br />

<span
    atom-text="[$scope.termsAgreed ? 'Terms agreed' : '(Please Accept Terms)']"
    style-color="[$scope.termsAgreed ? 'inherit' : 'red']"></span>
```

6.5.4 Class Attribute

Class attribute behaves little differently than any other HTML attributes. Class attribute basically toggles between different values provided.

```
<div atom-type="AtomListBox">
    <div
        atom-class="[$scope.itemSelected ? 'a-selected-item' : 'a-item']">
    </div>
</div>
```

- In above example, it item was selected, `a-selected-item` class will be added to element's class using jQuery's `addClass` method.
- And when binding changes, a-item class will be added to element's class using `addClass` and `a-selected-item class` will be removed.
- **Class attribute is context aware**, it will undo its last class before setting new class.
- Also class attribute does not override any existing css classes assigned to the element.

Class attribute also accepts Inline Object Definition as shown below.

```
<div
    atom-class="[ { 'a-selected-item': $scope.itemSelected, 'a-item': !$scope.itemSelected } ]">
</div>
```

You will find that earlier example was smaller in this case, however you have choice of using class attribute the way you want to use it.

6.5.5 Style Attribute

Writing long style becomes difficult to analyze and maintain, although styles must be managed mostly by css, but when style comes from database or any user data, than you cannot create css class.

```
<div
    style-color="#FF00FF"
    style-background-color="{ $scope.itemSelected ? 'blue' : 'white' }"
    style-width="{ $data.Width + 'px' }">
</div>
```

You can apply any valid style attribute on any element, and you can also perform binding on style attributes individually. This makes code more readable and you have various options to customize them.

6.5.6 Html Attribute

Use of atom-html should be avoided as long as possible, it is only available for extensibility. This property does not use any Web Atoms functionality and you should use Templates to create your designs.

We recommend using template mechanism instead of atom-html property, however in sometimes it may be helpful to know that you can set `innerHTML` by using `atom-html` property. You must know that `atom-html` does not initialize any binding within itself, it is a pure HTML replacement. If you want binding to work correctly, then you must use corresponding templates.

```
<div atom-html="{ '<span>Text</span>' }"></div>
<!-- Gets Converted to -->
<div><span>Text</span></div>
```

6.5.7 Absolute Position Attribute

Sometimes creating absolutely positioned element requires a little big code, so we created a short hand, which is bindable as shown below. It accepts an array of values or string with comma separated values for Left, Top, Width and Height;

```
<div atom-abs-pos="25,25,200,100"></div>
<!-- Gets Converted to -->
<div style="position:absolute; left:25px; top:25px; width: 200px; height:100px;">
</div>
```

You can specify null to skip the parameter.

```
<div atom-abs-pos="25,25,null,100"></div>
<!-- Gets Converted to -->
<div style="position:absolute; left:25px; top:25px; height:100px;">
</div>
```

However, this should only be used with caution, and other layout options such as layout controls or css should be used instead.

6.6 Atom Prefix

HTML5 specifications states that any extra attribute on element should be named with `data-` prefix. However, since Web Atoms is UI framework and Web Atoms has specific named property called `data` which carries data as per English language. Using data prefix causes confusion, as we are talking about UI attributes and not specifically data.

```
<span data-text="{$data.FirstName}"></span>
```

Above code looks confusing. Not only this, Web Atoms has many controls and many properties, each properties are `atom-` properties. To make it completely HTML5 compliant, we tried following.

```
<span data-atom-text="{$data.FirstName}"></span>
```

Which is also very confusing, and it gets longer and longer. In our future version, we are planning to write pre-processor for Web Atoms, which will convert your code from Web Atoms to HTML5 compliant HTML+JS by stripping all `atom-` attributes and encapsulate them into a separate script initializer which will also help in obfuscating the code.

Even for that transition, you will not have to change your code.

7 SCOPE

We have introduced concept of scope (UI scope) in web atoms to emulate UI scope as it exists in other UI frameworks such as flex, Silverlight etc. UI scope basically creates boundary and it allows us to identify and reference named properties and named controls that is independent of other scopes. `AtomApplication` starts with a global scope referred as scope in same scope and as `appScope` in nested scope.

Usually when you declare a function or variable, it executes in context of current window, JavaScript provides scoping within boundaries of function. However, this scoping has nothing to do with element within which JavaScript is executing. And there is no way to separate variable references in different scope based on its position in the element.

7.1 Initialization

As we have come up with a pattern of defining `model` named global variable for the page, we have included same model in `$appScope`.

```
<script type='text/javascript'>
    var model = { name: 'Web Atoms' };
</script>
```

Now, `$scope.model` in global scope, or `$appScope.model` refers to exact same variable defined above.

If you have any other global variables that you would like add in the scope, you have to initialize scope as following. Let's say you have a variable named `jConfigValues`, defined in some JavaScript.

```
<script type='text/javascript'>
    ({ jConfigValues: jConfigValues })
</script>
```

To emulate private scope based on position of script within Html Element and to bind script to specific Element's private scope, Scope scripts were specially designed so that its normal execution does not affect anything, but they are executed in proper scope.

```
<script type="text/javascript">
   ({
       start: 0,
       size: 20,
       infoCommand: function () {
           alert('Done');
       }
   })
</script>
```

Above script tag, causes `start`, `size` and `infoCommand` to be initialized as properties in respective scope. `infoCommand` is a function, and it can be referenced in same scope.

Note: Please note the difference here, script shown in previous section is a simple JavaScript defined in page header. Script shown in this example is a Session Initialization Script which only appears within Web Atoms enabled HTML Elements.

Scope initialization is defined this way, because this form of script is actually just parsed by JavaScript, but nothing is executed as it is a valid object literal expression. Since scope is nested, and based on control life cycle, scope will be initialized at its perfect position in UI Hierarchy, this is just stored as reference code to be executed later on.

Scope initialization script nodes are removed from Document while parsing further children. For safety, any JavaScript other than Scope Initialization must only be kept in Document's Head Section.

7.2 Scope (Current Scope)

Term `$scope` is referred as current scope of the control. Since scope is managed by Web Atoms Framework, you can always climb up the hierarchy through `$scope.parent` etc.

7.3 Scope Control Name

By setting `id` attribute, HTML element is accessible easily. But when we are dealing with multi item controls such as List, Grid etc., referencing items could be very difficult. Scope helps us in referencing different items in same scope very easily by setting `atom-name` attribute of the `AtomControl`. Once you apply atom-name attribute, you can easily refer properties of controls by calling `$scope.controlName.propertyName` pattern. In binding expression, property names are automatically resolved so you do not have to use `Atom.get` or `Atom.set` methods.

7.4 Scope Template Name

Just like Scope Control Name, you can apply `atom-template-name` attribute to any HTML element and that element is removed from DOM and referenced for resolving templates at runtime.

Templates are usually HTML elements with extra attributes, which are used by Web Atoms Template System. Apart from inline templates, scope templates gives you advantage of changing templates at runtime.

Once `atom-template-name` attribute is applied to any element, element is then removed from the DOM and stored as template in scope. You can bind any template of any control to `$scope.templateName` as shown below. In the following example, `$scope[$scope.view]` determines the template to be used at runtime. Toggle button bar changes the selected template name, that causes `atom-item-template` to be set to new value and all items are now rendered with new template. Scope Template can contain any atom controls as well, just like inline templates.

```
<div
    atom-template-name="small"
    style-background-color="{$data.value}">
    <span atom-text="{$data.label}"></span>
</div>

<div
    atom-template-name="medium"
    style-background-color="{$data.value}">
    <span atom-text="{$data.label}"></span>
</div>

<div
    atom-type="AtomItemsControl"
    atom-items="{ $scope.list }"
    atom-item-template="[$scope[$scope.view]]">
</div>
```

In above example, small and medium elements are stored in Scope as templates. You can assign template to any template property of any control and you can bind a dynamic expression which when changes, UI is recreated with new template.

7.5 Application Scope (Global Scope)

Unless, scope initialization appears inside any Items control, scope is considered as a global Scope and is referred as `$appScope`. `$appScope` is same as `$scope` for all controls in the same scope. However nested scope controls can access global scope by using $appScope.

7.5.1 Navigation History

In AJAX applications, saving state of page is very important. As in browser, we need back button functionality to function exactly like the way we get with simple web pages. Web Atoms makes it easy to maintain navigation state by serializing global scope (appScope) values into URL separated by `#` token. Scope initialization is ignored if values exists in URL hash and it works without writing any extra scripts.

```
<script type="text/javascript">
    ({
        view: 'red',
        list: [
            { label: 'Oranges', value: 'orange' },
            { label: 'Apples', value: 'red' },
            { label: 'Grapes', value: 'green' }
        ],
        display: function (item) {
            if (!item) return '';
            return item.label + ' are ' + item.value;
        }
    })
</script>
<div
    atom-name="buttonBar"
    atom-type="AtomToggleButtonBar"
    atom-items="{ $scope.list }"
    atom-value="$[scope.view]">
    <span
        atom-template="itemTemplate"
        atom-text="[$data.label]"></span>
</div>
<span atom-text="[$scope.display($scope.buttonBar.selectedItem)]" ></span>
```

Let's review above example.

- `view` property in scope is initialized to 'red'. So when you first load the page, view is set to 'red'.
- We have set two way binding between `$scope.view` and `value` property of `AtomToggleButtonBar`. This causes `AtomToggleButtonBar` to set Apple as default selected item when the page is loaded first.
- URL does not contain any hashtag to begin with.
- Now when we change `AtomToggleButtonBar` by clicking on it, note that URL now contains hashtag for 'view' property. This means scope has saved 'view' on URL.
- Now if you will refresh the page, view will be set to the value that was saved in URL. And `AtomToggleButtonBar` will be defaulted to last selected value.

7.5.2 URL Hash Change

URL hash changes whenever any scope property is modified by Web Atoms as a result of any User Action or any AJAX result. All the properties of appScope are saved onto URL. Size of URL depends upon browser, web atoms has no limit, but we recommend using only smaller values like integer indexes or short keywords to be saved.

7.5.3 Eligible App Scope Value Types for URL Hash

You may notice that in above example, we also have a function called 'display' and a list called 'list', but both were not serialized on URL. Here is why, scope values are only saved on URL under following conditions.

- Only Global (Root Level) or appScope values are saved on URL.

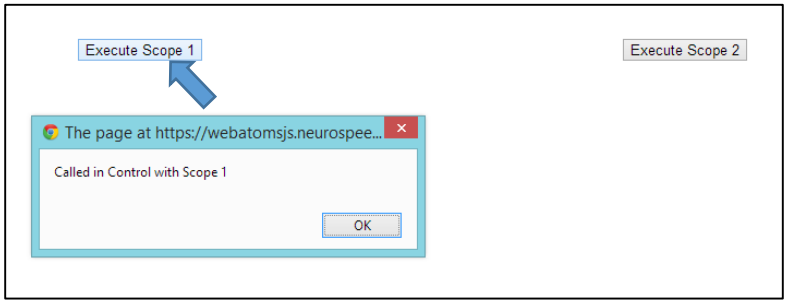
- Value must be of type "String", "Number" or "Boolean" only. Anything out of this, object, function or array are ignored.
- Corresponding property name of value must not start with underscore. Anything that starts with _ is not saved onto URL. Sometimes puting everything on URL causes lots of navigation history cycles that user may get confused. So only few scope properties must be chosen that should be saved on URL. Rest must be named with _ underscore prefix to avoid confusion.

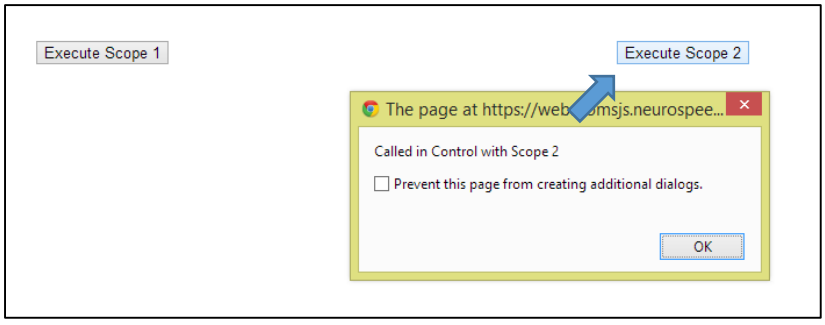
7.6 Local Scope

In languages like Flex and Silverlight, ability to create User Controls provided isolation for component authors to avoid naming conflict. However, HTML does not have any naming scope related to HTML Element. Function level private scope is only applicable in scripting context which is independent of visual hierarchy of HTML Elements.

In order to reduce naming conflict, and to provide private data store, Web Atoms provides Local Scope for Atom Controls. By marking `AtomControl` with `atom-local-scope=true` attribute, every child element and child `AtomControl` becomes part of private local scope created at this element. Control and its descendants can communicate with each other via `localScope`. Following example illustrates how two different control on same page, with same named scope items are isolated for their execution.

```
<div
    atom-type="AtomControl"
    atom-local-scope="true"
    atom-abs-pos="100,100,500,200">
    <script type="text/javascript">
        ({
            name: "Scope 1",
            run: function (scope, sender) {
                alert("Called in Control with " + scope.name);
            }
        })
    </script>
    <button
        atom-event-click="{$localScope.run}"
        >Execute <span atom-text="{$localScope.name}"></span></button>
</div>

<div
    atom-type="AtomControl"
    atom-local-scope="true"
    atom-abs-pos="600,100,500,200">
    <script type="text/javascript">
        ({
            name: "Scope 2",
            run: function (scope, sender) {
                alert("Called in Control with " + scope.name);
            }
        })
    </script>
    <button
        atom-event-click="{$localScope.run}"
        >Execute <span atom-text="{$localScope.name}"></span></button>
</div>
```





8 ATOM CONTROL PROPERTIES

Following properties are accessible from current control. And their usage and understanding is explained below.

8.1 `scope` Property

Scope refers to current scope in which current control was created. Based on position of the control/element, the scope will change. However scope refers to same scope as current control's scope.

8.2 `appScope` Property

Following example will demonstrate perfect difference between `scope` and `appScope`.

```
<div atom-type="AtomApplication">
    <script type="text/javascript">
        ({
            searchText: ''
        })
    </script>

    <div atom-dock="Top" style="height:30px">
        <input type="search" atom-value="$[scope.searchText]" placeholder="Search" />
    </div>
    <div
        atom-dock="Fill"
        atom-type="AtomItemsControl"
        atom-items="{ AtomPromise.json('/db/movieList') }">
        <div
            atom-template="itemTemplate"
            style-display="[ $data.Name.indexOf($appScope.searchText)>0 ? '' : 'none' ]">
            <span atom-text="{ ($scope.itemIndex + 1 )+  ')'}"></span>
            <span atom-text="{ $data.Name }"></span>
        </div>
    </div>

</div>
```

8.3 `localScope` Property

Following code illustrates how `localScope` is accessed.

```
<div atom-type="AtomApplication">
    <script type="text/javascript">
        ({
            searchText: ''
        })
    </script>

    <div atom-dock="Top" style="height:30px">
        <input type="search" atom-value="$[scope.searchText]" placeholder="Search" />
    </div>
    <div atom-type="AtomViewStack">
        <div
            atom-type="AtomDockPanel"
            atom-local-scope="true">
            <script type="text/javascript">
                ({
                    movieType: 'Action'
                })
            </script>
            <span
                atom-type="AtomToggleButtonBar"
                atom-items="{ AtomPromise.json('/db/movieTypes') }"
                atom-value="$[scope.movieType]">
            </span>
            <div
                atom-dock="Fill"
                atom-type="AtomItemsControl"
                atom-items="{ AtomPromise.json('/db/movieList') }">
                <div
                    atom-template="itemTemplate"
                    style-display="[ $data.Name.indexOf($appScope.searchText)>0 ? 'block' : 'none' ]"
                    style-background-color="[$data.Type == $localScope.movieType ? 'yellow' : 'inherit']">
                    <span atom-text="{ ($scope.itemIndex + 1 )+  ')'}"></span>
                    <span atom-text="{ $data.Name }"></span>
                </div>
            </div>
        </div>
    </div>
</div>
```

8.4 `atomParent` Property

When we design complex user interface and we have to reference n-level of parent for some sort of binding. We can reach n-level parent by accessing `atomParent` property. This is only useful when atom-name is not available. However, in any case, atom-name should be preferred over `atomParent` unless you do not have any choice.

```
<div
    atom-type="AtomViewStack">
    <div
        atom-type="AtomDockPanel">
        <div
            atom-text="{$owner.atomParent.selectedIndex}"></div>
    </div>
</div>
```

8.5 `templateParent` Property

Template Parent holds reference of a control which contained the template in the first place. Also note, `atomParent` is not a good idea to reach parent inside a template because template may not be an immediate child of the `templateParent` which defined the template. At runtime, templates are added inside specific template presenters, for example, in an AtomItemsControl `itemTemplate` is added inside `itemsPresenter` element. For the record, `itemsPresenter` itself may be any type of `AtomControl`.

```
<div
    atom-type="AtomNavigatorList"
    atom-new-item="{ { MovieName: '', MovieCategory: '' } }"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }"
    atom-name="movieList"
    style="width:400px; height:400px; margin:50px;"
    >
    <table atom-template="gridTemplate">
        <thead>
            <tr>
                <th>Name</th>
            </tr>
        </thead>
        <tbody atom-presenter="itemsPresenter">
            <tr atom-template="itemTemplate">
                <td atom-text="{$data.MovieName}"></td>
            </tr>
        </tbody>
    </table>
    <span
        atom-template="detailHeaderTemplate"
        atom-text="['Movie: ' + $data.MovieName]">
    </span>
    <div
        atom-template="detailTemplate"
        atom-type="AtomFormLayout">
        <input
            atom-label="Movie Name:"
            atom-value="$[data.MovieName]"/>
        <input
            atom-label="Movie Category:"
            atom-value="$[data.MovieCategory]"/>
        <button
            atom-event-click="{$owner.templateParent.backCommand}"
            >Save</button>
    </div>
</div>
```

In above example, button is inside `AtomFormLayout`, and `AtomFormLayout` automatically encapsulates item inside `AtomFormField` control, since this is done at runtime, and if you are not aware of it the `atomParent` sequence will fail.

9 ASYNCHRONOUS PROGRAMMING WITH WEB ATOMS

9.1 Pain of writing asynchronous tasks

Writing Asynchronous tasks can become very tedious as it involves complex closures or inline anonymous functions or function references. Let us look at typical asynchronous code for loading JSON content from any URL.

```
<script type="text/javascript">

    function failed(){
        alert('Something went wrong');
    }

    function loadData(id, product, orders) {
        $.ajax('/create/entity/product', {
            data: {
                entity: product
            },
            success: function (r) {
                $.ajax('/save/entity/orders?productID=' + r.ProductID, {
                    data: {
                        entityList: orders
                    },
                    success: allSaved,
                    error: failed
                });
            },
            error: failed
        });
    }

    function allSaved(){
        alert('Saved successfully')
    }
</script>
```

If we were supposed to write this once and forget it, it was easy to do, but that's not the case, we always have to maintain this code. More complex nested function calls, worse the code becomes.

We purposely demonstrated the code that is not organized in neither alphabetical nor in order of execution. And this is what the real life scenario, no matter how much people try to adhere to great design, not everybody follow same path and it leads to unsolvable maze.

So JavaScript does have promises, but same way, writing endless nested anonymous methods or complex naming pattern dose not ease pain of asynchronous programming. Whichever JavaScript framework you adopt, you will always end up with complex unsolvable maze.

So after point of time, above complex arrows become your reality and you struggle yourself to move around them in JavaScript in debugger, if you are dealing older version of browsers or limited versions such as mobile, you will have nightmare debugging those.

9.2 AtomPromise

Since Web Atoms is based on components, you can assign `AtomPromise` to any property. However the assignment looks as synchronous, but it works asynchronously. As when value is set on any property, property system tests the type of value, if it is `AtomPromise` then property system hooks itself for the finish event of promise and assigns value to corresponding property automatically.

```
<select
    atom-type="AtomComboBox"
    atom-items="{ AtomPromise.json('/config/country-list.json')}"
    atom-value="$[scope.selectedCountry]">
</select>
```

In above example, we want to load list of countries in this combo box, however in 3rd line it appears as if we are loading the countries synchronously and setting value to items property of combo box. But in reality, we are just setting up a promise here, JavaScript call does not block here, in fact when property system evaluates this expression, it recognizes it as a valid promise and hooks up an event for completion in background.

`AtomPromise.json` loads data from given URL in JSON format as the name suggests, it internally calls `AtomPromise.ajax` which wraps jQuery's `$.ajax` method. There are various other `AtomPromise` methods available and you can completely change or customize promises to your needs.

9.3 AtomForm

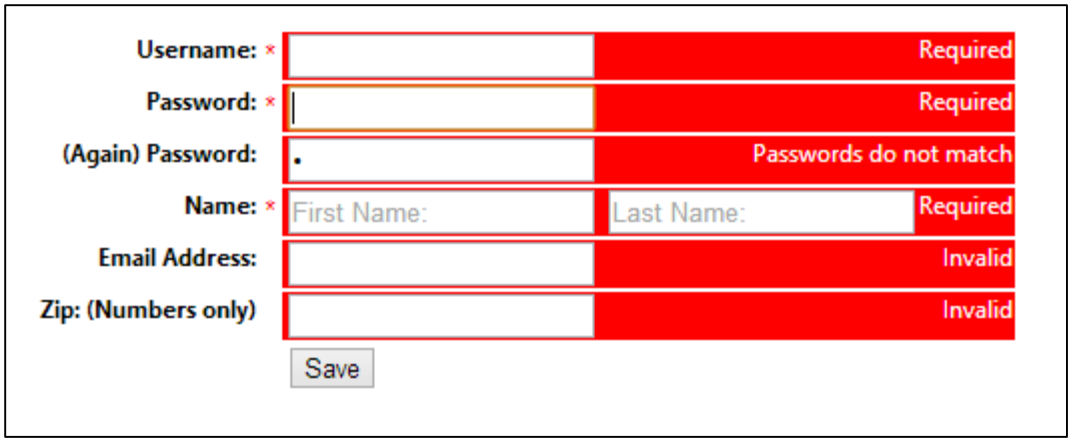
`AtomForm` control posts data in JSON format to given URL and it merges back the post results in same data. This was designed to merge return values in same object by considering a fact that we may alter values on server side to meet certain standards such as capitalization and we need key for newly added objects.

`AtomForm` is a versatile control, it performs validations on data and with use of `AtomFormLayout`, it is easy to design complex forms in minutes. `AtomFormLayout` takes care of visual appearance of form items, validation and display of error message. And it is completely customizable.

In the following code sample, you can see a fully functional signup form without writing a simple line of JavaScript.

`AtomFormLayout` recognizes certain extra attributes such as required, is-valid, error, data-type or regex, and it performs validation on those fields automatically.

```
<div
    atom-type="AtomForm"
    atom-post-url="/Url/Insert"
    atom-data="{ { FirstName: '', LastName: '', Password: '', Password2:'' } }"
    atom-success-message="Information Saved Correctly">
    <div
        atom-type="AtomFormLayout">

        <input
            type="text"
            atom-label="Username:"
            atom-required="true"
            atom-value="$[data.Username]"/>

        <input
            type="password"
            atom-label="Password:"
            atom-value="$[data.Password]"
            atom-required="true"/>

        <input
            type="password"
            atom-label="(Again) Password:"
            atom-value="$[data.Password2]"
            atom-is-valid="[$data.Password == $data.Password2]"
            atom-error="[$owner.isValid ? '' : 'Passwords do not match']" />

        <span
            atom-label="Name:"
            atom-required="true"
            atom-is-valid="[$data.FirstName && $data.LastName]">
            <input
                placeholder="First Name:"
                type="text"
                atom-value="$[data.FirstName]" />

            <input
                placeholder="Last Name:"
                type="text"
                atom-value="$[data.LastName]" />
        </span>

        <input
            atom-label="Email Address:"
            type="text"
            atom-data-type="email"
            atom-value="$[data.EmailAddress]" />

        <input
            atom-label="Zip: (Numbers only)"
            type="text"
            atom-regex="/[0-9]+/"
            atom-value="$[data.ZipCode]" />

        <input type="submit" value="Save" />

    </div>
</div>
```



9.4 AtomPostButton

`AtomPostButton` on the other hand behaves exactly as the Form, but it does not validate anything and it just performs HTTP POST when button was clicked. It does not merge results back into data as `AtomForm` does it. This is useful for changing some status of items such as changing its rating, or deleting and undeleting items in a Grid.

```
    <td  atom-type="AtomPostButton"
        atom-post-url="/app/entity/book/delete"
        atom-post-data="{ { BookID: $data.BookID } }"
        atom-confirm="true"
        atom-confirm-message="Are you sure you want to delete this book?"
        atom-next="{$owner.templateParent.removeItemCommand}"
        class="atom-delete-button">Delete</td>
```

In above example, `AtomPostButton` will first ask for confirmation, with confirm method of JavaScript. Only if user confirms, then it will perform HTTP POST to given URL. Since while deleting, we should only post `BookID` to server. If atom-post-data is not specified, contents of data property will be posted as JSON to server.

9.5 Benefits of Web Atoms

As you can see, with help of `AtomPromise` and controls like `AtomPostButton`, `AtomForm`, Web Atoms offers complete script less programming. You no longer have complex method calls calling each other. In fact, atom-next applied to each control will perform next logic as per Action Sets.

10 Data Binding

Various frameworks like Flex, Silverlight, and WPF etc. introduced data binding mechanisms that helped us in creating templates of User Interface elements and differentiate between data and view. Declarative data binding helps in understanding relation between user interface and data (usually called as model).

However, HTML and JavaScript currently do not support any forms of data binding. But HTML does allow you to define extra attributes, and JavaScript can process them.

Web Atoms will process xml attribute values between curly braces {} and square brackets. And convert them into suitable binding expressions.

In the following sample, you can see how `firstName` and `lastName` properties of model are bound to UI element. One time and one way binding expressions can contain any form of JavaScript expression. (JavaScript expression is entire expression after equal to (=) sign in assignment statement till semi colon.)

```
<!-- 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>
```

Binding Expressions can only reference global JavaScript variables and properties of control ($owner) that holds the element. Others are just shortcuts. In short, `$data` is equivalent to `$owner.data`, `$scope` is equivalent to `$owner.scope`.

Note: Binding will only work for these keywords. $data (data property of current control), $scope, $owner (self, since this is not accessible here), $appScope (application level scope). Any property after these keywords will be bound automatically.

10.1 One Time Binding

Binding expression within curly braces are evaluated at time of initialization of control. You can consider them as simple `eval` expression which will be instantly evaluated at time of building of page. `Eval` expressions are executed in private scope in function constructor.

All expressions in following sample are one time binding expressions.

```
<div
    atom-type="AtomControl"
    atom-data="{ {firstName: 'Akash', lastName:'Kava', favColor: 'orange'} }">

    <span
        atom-text="{$data.firstName + ' ' + $data.lastName}"
        style-color="{$data.favColor}"></span>
</div>
```

Here is how it works,

- `atom-type` set to `AtomControl` will make element an `AtomControl` that supports binding
- `atom-data` will be initialized to anonymous object, please note outer curly braces are used to identify it as one time binding expression, and actual expression is inside curly braces.
- `atom-text` will be evaluated as concatenation of `firstName` and `lastName`.
- `style-color` will be evaluated as `styleColor` set as `favColor` property of object. Style binding has special syntax prefixed with `style-` and it will be explained later in this help.

Note: You must always use One Time Binding that uses least memory. One Time binding will never refresh its data.

10.1.1 JavaScript Native Type Values

One Time Binding is also used to set native values in properties. For example, setting false to a Boolean property can be tricky as `"false"` string literal is considered as true. Curley braces expressions are simply evaluated by function constructor and the result is stored in the property. So `"{ false }"` evaluates to a proper Boolean False value.

10.2 One way Binding

Building data aware application requires updating UI automatically when data changes. One way binding refreshes element automatically when data is modified by any one. To make it simpler, you have to write your binding expressions in square brackets `[]`.

In the following example, by convention, variable named model is automatically assigned in scope at application level. `AtomBinder` manages bindings between elements, if you are modifying property of an object, we recommend using `Atom.set`. However in Two Way Binding (explained in next section), binding is automatically updated by Web Atoms.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Atom-Control Sample</title>
    <script src="/Scripts/jquery-1.8.2.js"></script>
    <link href="/Style/WebAtoms.css" rel="stylesheet" />
    <script src="/Scripts/WebAtoms.js"></script>

    <script type="text/javascript">
        // model named variable is same as $scope.model
        var model = {
            time: ""
        };

        // updating time property of model
        function updateTime() {
            Atom.set(model, "time", (new Date()).toLocaleString());
        }

        setInterval(updateTime, 1000);
    </script>
</head>
<body>
<div atom-type="AtomApplication">
    <span atom-text="[ 'Current Time is ' + $scope.model.time ]"></span>
</div>
</body>
</html>
```

10.3 Two Way Binding

Two Way binding is most important part of Web Atoms Framework, which makes it very easy to update the data dynamically without writing any JavaScript. Binding framework automatically updates any relative one way binding and updates the changes throughout the page.

In following example, there is a toggle button bar, which displays list of items and one can be selected. Binding expression `atom-selected-item` is written with `$` prefix, which makes this expression as two way binding, which means any time either side expression is modified, other will be updated at the same time. Here, whenever selected item will be changed by user (by clicking and changing the default selected item of Toggle Button Bar), this binding expression will save selected item as selection named property in scope.

So, whenever `$[scope.selection]` will be updated by atom-selected-item by user interaction, other elements bound to `$scope.selection` will be updated automatically.

```
<script type="text/javascript">
    ({
        list: [
            { label: "Orange", value: 5.00, color: "orange" },
            { label: "Apple", value: 6.00, color: "red" },
            { label: "Grapes", value: 2.00, color: "green" }]
    })
</script>
<div
    atom-type="AtomToggleButtonBar"
    atom-items="[$scope.list]"
    atom-selected-item="$[scope.selection]">
    <span
        atom-template="itemTemplate"
        atom-text="{$data.label}"></span>
</div>

<div atom-type="AtomControl">
    <span
        atom-text="[$scope.selection.label]"
        style-background-color="[$scope.selection.color]"></span>
</div>
```

10.3.1 Two Way Binding Update Event

By default, two way binding is updated on "change" event of HTML Element. However, sometimes you may want to update binding on any other event for example "keyup" event. It can be done by typing comma separated events in brackets after declaration of two way binding.

```
<script type="text/javascript">
    ({
        user: {
            firstName: '',
            lastName: ''
        }
    })
</script>
<div
    atom-type="AtomControl"
    atom-data="{$scope.user}">
    <input type="text"
        atom-value="$[data.firstName](keyup)"
        placeholder="First Name" /><br />
    <input type="text"
        atom-value="$[data.lastName](keyup)"
        placeholder="Last Name" /><br />
    <input type="button" value="Save" />
    <span
        atom-text="['Name is ' + $data.firstName + ' ' + $data.lastName]">
    </span>
</div>
```

10.4 Style Binding

Along with text and values, Web Atoms provides easy way to perform (one way and one time) binding on element style as well. This form of binding allows us to easily split style into different attributes and bind them independently as shown in example below.

```
<script type="text/javascript">
    ({
        list: [
            { label: 'Orange', itemColor: 'orange', itemWidth: 100 },
            { label: 'Apple', itemColor: 'red', itemWidth: 200 }
        ]
    })
</script>

<div
    atom-type="AtomItemsControl"
    atom-items="{ $scope.list }">
    <div
        atom-template="itemTemplate"
        atom-text="{$data.label}"
        style-color="#000000"
        style-background-color="{$data.itemColor}"
        style-width="{ $data.itemWidth + 'px' }">
    </div>
</div>
```

We have displayed 2 forms of binding.

`style-color` is set to `#000000` (black) color at time of initialization. This is same as writing inline style, but this helps us in dividing individual style properties.

`style-background-color` will be set to corresponding item's `itemColor` property.

`style-width` will be set to an expression that will add `'px'` at the end as required by element style.

10.5 Event Binding

Event handling in Web Atoms is done little differently, as you may have learned about Action Set, Web Atoms is more focused towards invoking Action Sets instead of executing a callback. However in an ActionSet you can certainly specify a callback as well.

```
<script type="text/javascript">
    ({
        done: function () {
            alert("Done");
        }
    })
</script>
<button
    atom-event-click="{ $scope.done }">Click Me</button>
```

The only reason to use Action Set is, you can achieve isolation in different scope and not interfere with other components with similar names.

11 ACTIONSET

As discussed in asynchronous programming chapter, we realized that we wanted to remove dependency upon JavaScript functions and wanted to make it more configurable with attribute specifications.

11.1 Events Processing

Typically, JavaScript requires function references to be attached to events, whether you use native JavaScript methods or you use framework like jQuery. Same as asynchronous programming, event processing also becomes big pain. As you end up with lots and lots of functions, and a complex chain of events that you struggle to name everything to understand and analyze your code.

11.2 Simple Actions

To simplify events processing, we classified following simple actions that can be declared on attributes for events processing. You can specify four types of data on ActionSet. String, object literal, function reference and array of string, object literal or function reference, array can also contain array.

11.3 String URL Action

Let's say we want to create a button, which should navigate to http://webatomsjs.com. Of course we can write same with simple anchor tag. But benefit of String URL Action is, it is bindable, you can customize behavior by changing content of next attribute itself.

```
    <button atom-event-click="http://webatomsjs.com">Go</button>
    <script type="text/javascript">
        var url = "http://webatomsjs.com";
    </script>
    <button atom-event-click="{ home ? Atom.refreshWindowCommand : url }">Go</button>
```

As you can notice, first button works just like an anchor tag, but second button has a conditional action set, which depends upon value of home variable. `Atom.refreshWindowCommand` is a function reference where else URL is a variable.

When we will click the button, if home is true, the same page will be refreshed. Otherwise, browser will navigate to URL if contents of URL is a string.

11.4 Function Reference

As we have seen in previous example, click event is bindable and we can bind it to a function reference. Which will be invoked when we will click the button.

```
<script type="text/javascript">
    ({
        alertMe: function () {
            alert('Hellow World');
        }
    })
</script>
<button atom-event-click="{ $scope.alertMe }">Go</button>
```

As you can see, `alertMe` is a function reference in current `$scope`. You can also reference any function defined anywhere in JavaScript Context, however in that case, that function reference is a global reference and which can lead to same name conflicts. We recommend using everything in the form of scope member, so that when you combine or reuse code, your code will have least naming conflicts.

11.5 Object Literal

Object Literal action set can contain set of properties that will represent certain action to be performed against current element or `AtomControl` associated with it. This is little complicated to understand, as this is a very special way of executing instructions in Web Atoms. Mostly, this is used as way of toggling some values of properties.

First let us consider, function reference sample of increasing a counter stored in scope.

```
<script type="text/javascript">
    ({
        counter: 1,
        incrementCounter: function (scope, sender) {
            var v = Atom.get(scope, "counter");
            Atom.set(scope, "counter", v + 1);
        }
    })
</script>
<span
    atom-text="[$scope.counter]"></span>
<button
    atom-event-click="{$scope.incrementCounter}" ></button>
```

As you can see, we have to write lots of code for just toggling some value, which requires calling `Atom.set` and `Atom.get` methods which refreshes corresponding bindings. But to make things easier, we can use object literal syntax to specify what values we want to merge and where.

```
<script type="text/javascript">
    ({
        counter: 1
    })
</script>
<span
    atom-text="[$scope.counter]"></span>
<button
    atom-event-click="[ { scope: { counter: $scope.counter + 1 } } ]" ></button>
```

In this example, outer curly braces specifies that it is a one way binding, and inner curly braces specifies it is an object literal syntax. We have object literal with property scope, which designates that we have to merge values inside scope in the scope of current atom control.

1. `$scope.counter` is 1 when page starts.
2. Span displays 1 as the text.
3. Contents of `atom-event-click` is `{ scope: { counter: 2 } }`
4. When we click the button, `ActionSet { scope: { counter: 2 } }` gets evaluated, and value of counter in scope changes to 2.

5. Now `$scope.counter` is 2
6. So the span displays 2 as the text.
7. And contents of `atom-event-click` gets updated to `{ scope: { counter: 3 } }`
8. If you click button again, counter increases to next value.

Object Literal syntax can be used to change properties in scope, appScope and data of any control. Also note that, we can change multiple values in single object literal syntax.

Here is another example,

```
<script type="text/javascript">
    ({
        showDetails: false
    })
</script>
<button
    atom-event-click="[{ scope: { showDetails: !$scope.showDetails } }]"
    atom-text="[$scope.showDetails ? 'Hide' : 'Show']"></button>
<div style-display="[$scope.showDetails ? 'block' : 'none']">
</div>
```

We are simply changing `showDetails` value in scope, at the same time, button's text also toggles between Hide and Show and corresponding div also hides or shows as its display `css` changes.

11.5.1 Object Literal Scope

As you have seen in previous example, you can update values of scope of current `AtomControl` associated with current HTML Element.

11.5.2 Object Literal AppScope

Same as above, but here the values will be merged into global application level scope.

11.5.3 Object Literal Data

Data literal action will merge specified property in data associated with current element. Let's review following sample.

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

    <button
        atom-event-click="[ { data: { Quantity: $data.Quantity + 1 } } ]">
        Increase</button>
    <button
        atom-is-enabled="[ $data.Quantity > 1 ]"
        atom-event-click="[ { data: { Quantity: $data.Quantity - 1 } } ]">
        Decrease</button>
```

In this example, we have an input that is bound to Quantity, and we want to provide buttons to increase or decrease quantities. As you can see, action set is a literal object with property data which contains an object literal with field Quantity.

11.5.4 Object Literal Alert

Alert literal will display an alert to user. This is useful if you want to put an alert in case when user forgot to take certain action. Let's review this example, if there is a list box and user is supposed to select one item from it, but if user has forgotten, we should not allow form to be submitted.

```
<button
atom-event-click="[ !$scope.list.value ? { alert: 'Please select an item' } :
$scope.form.submitCommand ]"></button>
```

11.5.5 Object Literal Confirm

Confirm literal will display a confirmation box before executing next action. This action requires an array with first item as string (message) and next item should be subsequent action set. Please note, last item in array is a new action set.

```
    <button
        atom-event-click="[ { confirm: [ 'Are you sure you want to
proceed?',$scope.form.submitCommand] } ]"></button>
```

11.5.6 Object Literal Timeout

Timeout literal expects an array with first parameter as timeout duration and 2nd parameter as new action set to be performed after timeout duration. In the following example, we will turn `showDetails` member to true when mouse enters the div, however, when user leaves the div, we want to introduce little delay so that user can notice that there are more details when user hovers the div.

```
<script type="text/javascript/scope">
    ({
        showDetails: false
    })
</script>
<div
    atom-event-mouseenter="{ { scope: { showDetails: true } } }"
    atom-event-mouseout="{ { timeOut: [ 500, { scope: { showDetails: true } } ] } }">
    <span>Title</span>
    <div
        style-display="[ $scope.showDetails ? '' : 'none' ]">

    </div>
</div>
```

12 Creating an Application

12.1 AtomApplication

`AtomApplication` is root Control, whose children will be treated as Atom Controls and everything underneath will be treated as a Web Atoms Application. To create an application, you have to add a div tag in the body section of your page as shown below.

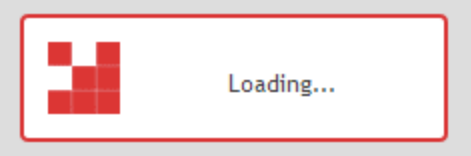
```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <link href="/WebAtoms.css" rel="stylesheet" />
    <script src="/Scripts/jquery-1.8.2.js"></script>
    <script src="/Scripts/WebAtoms.js"></script>
</head>
<body>
    <div
        atom-type="AtomApplication"
        atom-title="{ 'Sample Page - ' + (new Date()) }">
        <div
            atom-dock="Top"
            style="height:50px; background-color:greenyellow">
            Header
        </div>
        <div
            atom-dock="Fill"
            style="overflow:auto">
            This is application area docked to middle and stretches
            to full available area as much as it can.
        </div>
        <div
            atom-dock="Bottom"
            style="height:30px; background-color:greenyellow">
            Footer
            <a
                href="http://web-atoms.neurospeech.com"
                target="_blank"
                style="right:5px; position:absolute"
                >Powered by Web Atoms</a>
        </div>
    </div>
</body>
</html>
```

`AtomApplication` acts a Dock Panel by default, so it layouts its children based on `atom-dock` attribute.

12.1.1 Title

`AtomApplication` has a property named title, which is synchronized with document's title property, so if you set application's title, document's title is also updated. In above example, just for the demo, we have displayed current date in the title.

12.1.2 Busy Template



By default, this animation is displayed whenever any `ajax` call is under processing. This is set inside busy template of Atom Application, which you can easily customize for your needs. You must make sure that this animation is modal that means while this is running, user cannot access anything in background. This is for your own safety, changing any UI while `ajax` call is pending might result in unexpected results. However, in busy template you can certainly change this behavior.

```
<div
    atom-template="busyTemplate"
    style="position:absolute;left:0px;top:0px;z-index:10000; visibility:hidden"
    style-width="[$owner.appWidth + 'px']"
    style-height="[$owner.appHeight + 'px']"
    style-visibility="[$owner.isBusy ? 'inherit' : 'hidden']">
    <div
           class="atom-busy-window"
           style="position:absolute"
           style-left="[(($owner.appWidth/2)-100) + 'px']"
           style-top="[(($owner.appHeight/2)-25) + 'px']">

           <div atom-abs-pos="12,12,36,36"
            class="atom-busy-image"></div>
           <div atom-abs-pos="56,24,145,null"
            atom-text="[$owner.busyMessage || 'Loading...']"></div>

        <div atom-abs-pos="0,48"
            style="height:3px; background-color:green"
            style-display="[$owner.isBusy &amp;&amp; $owner.progress ? 'block' : 'none']"
            style-width="[$owner.progress + '%']"></div>

    </div>
</div>
```

The outermost div occupied entire screen, thus by blocking access to any background element. First child with class `atom-busy-window` is actually what is visible with a red border and animation inside. Busy Template will display progress as well if file upload is in process. You can change this template as per your design, but please remember to bind `isBusy` and other important bindings.

12.2 Dock Panel

`AtomApplication` is derived from `AtomDockPanel`, so it inherits behavior of docking and works exactly like a dock .For more details about Dock Panel in section Dock Panel

12.3 Run as Page

In case if you do not want your panels to dock inside `AtomApplication`, you can set atom-run-as-page as true as shown below, this will turn off docking and page will simply render as html page. However it will still perform all the binding and other web atoms tasks.

```
<div atom-type="AtomApplication" atom-render-as-page="true">
```

13 Data Binding to Array

In Business applications, we typically need to manage collection of objects (Arrays in JavaScript), they appear in the form of table, drop down list etc. And we need to filter, sort and page them, either locally or remotely. To make all operations easier, Web Atoms comes with `AtomItemsControl` a huge control with template manager, selection manager and style manager for array of items.

All other controls such as List Box, Combo Box and Toggle Button Bar are derived from `AtomItemsControl` and they only appear and function little differently but `AtomItemsControl` does most of work.

13.1 Label Value Pair

Most tedious part of any UI framework is to manage with Label/Value pair. Drop Down Menus or Toggle Button Bars lets you select an item from collection. But mostly in database, we store a value which is probably numeric and we use a text to display, for example country code and country name. Web Atoms `AtomItemsControl` was designed to manage Label/Value pair very nicely that fits in real world application.

```
<script type="text/javascript">
    ({
        list: [
            { label: 'Orange', value: 1, itemColor: 'orange', itemWidth: 100 },
            { label: 'Apple', value: 2, itemColor: 'red', itemWidth: 200 }
        ]
    })
</script>

<div
    atom-type="AtomListBox"
    atom-name="theList"
    atom-items="{ $scope.list }">
    <div
        atom-template="itemTemplate"
        atom-text="{$data.label}"
        style-color="#000000"
        style-background-color="{$data.itemColor}"
        style-width="{ $data.itemWidth + 'px' }">
    </div>
</div>
```

`AtomItemsControl` has property called `value`, which stores selected item's `value` field. And value field is identified with `atom-value-path` property which is set to value by default.

In above example, value field stores number that identifies the object. Expression `$scope.theList.value` will display selected item's value. You can also look for `$scope.theList.selectedItem.value`. Advantage with value property over `selectedItem`'s value property is, it can retain values in case items are not loaded yet.

13.2 Cascaded Selections

The most difficult UI elements are ones that are cascaded, for example you have Country Selector and then you have State Selector. For new screens, everything can be empty, but in case of modifying existing items, retaining selection values are pain. For example, state will not show up correctly unless country is selected first. Sometimes in AJAX operations, if both countries and states are loaded from server, we have no guarantee or order to determine which one will finish first. And this leads to selection nightmares.

Web Atoms was designed to overcome such cascaded situation very carefully by designing selection procedure to avoid conflicts, giving high priority to user initiated selection, preserving existing values over default values.

```
<select
	atom-name="country"
	atom-label="Country:"
	atom-type="AtomComboBox"
	atom-required="true"
	atom-default-value="US"
	atom-items="{AtomPromise.cachedJson('/db/Countries')}"
	atom-value="$[$data.Country]"></select>

<select
	atom-type="AtomComboBox"
	atom-label="State / Province :"
	atom-required="true"
	atom-items="[AtomPromise.cachedJson('/db/Countries/' + $data.Country)]"
	atom-value="$[$data.State]"></select>
```

This example is perfect case of Label/Value pair along with Cascaded Selections. Countries contain country code as value and country name as label. And States have label and value for name and state code respectively. This code works well for adding new items as well as modifying existing items.

In case if list of state is not loaded, value preserves current selection and selects and displays current selection properly even if list of state is loaded after the selected value was already set from database.

In case of countries, while adding new record, “US” will be chosen as a default country.

14 ITEM COLLECTIONS

14.1 Items Control

AtomItemsControl is base class for all multi item controls such as AtomComboBox, `AtomListBox` etc. AtomItemsControl basically provides multi items management and provides templating mechanism for children. Each child is associated with one item in items array. AtomItemsControl provides selection mechanism including property value selection.

AtomItemsControl does not provide any styles for children, it only provides logic for items manipulation such as filter, sort, collection change notification and updates.

14.1.1 Scope

Every item of `AtomItemsControl` is created with a new Scope, so each item has its own private scope, and scope is managed by Web Atoms. This ensures scope bindings templates to remain private with each item and they do not conflict with one another. To refer to global scope, your binding must use `$appScope` instead of just `$scope`.

Scope provides additional properties as follow,

`itemIndex`	Index of current Item
`itemIsFirst`	True if this is the first item in the list
`itemIsLast`	True if this is the last item in the list
`itemExpanded`	This may be useful to perform tree kind of functionality
`itemSelected`	True if the current item is selected, you can also bind this with two way binding to update selection.

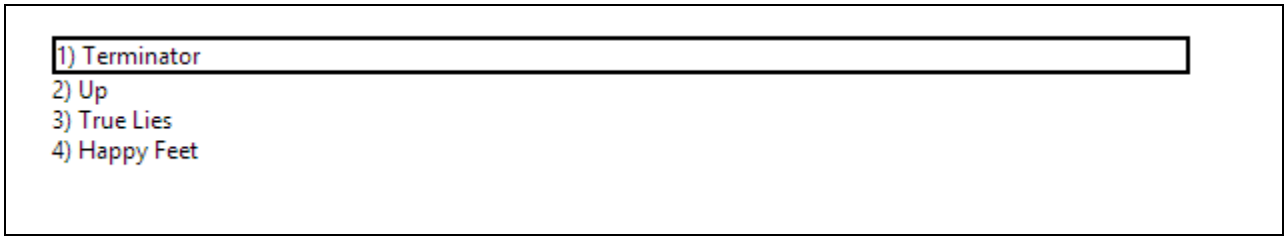
14.1.2 Item Template

```
<div
    atom-type="AtomItemsControl"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }"
    atom-name="movieList">
    <div
        atom-template="itemTemplate"
        style-border-style="{ $scope.itemIsFirst ? 'solid' : 'none' }"
        style-border-width="2px"
        style-background-color="{ $scope.itemIsOdd ? 'lightgray' : 'white' }">
        <span atom-text="{ ($scope.itemIndex + 1) + ') ' }"></span>
        <span atom-text="{ $data.MovieName }"></span>
    </div>
</div>
```

In above example, we have following bindings,

- First item of the list has a border style set as 'solid'
- Every odd item is having light gray background color
- First span indicates index of current item (we have added one to make it look like it starts from 1), indexes in JavaScript are zero based.

- Finally name of Movie by binding `MovieName` property to text.



14.1.3 Server Side Filtering

To perform server side filtering in AJAX, we have to send filtering parameters in query string and it can be done as shown below.

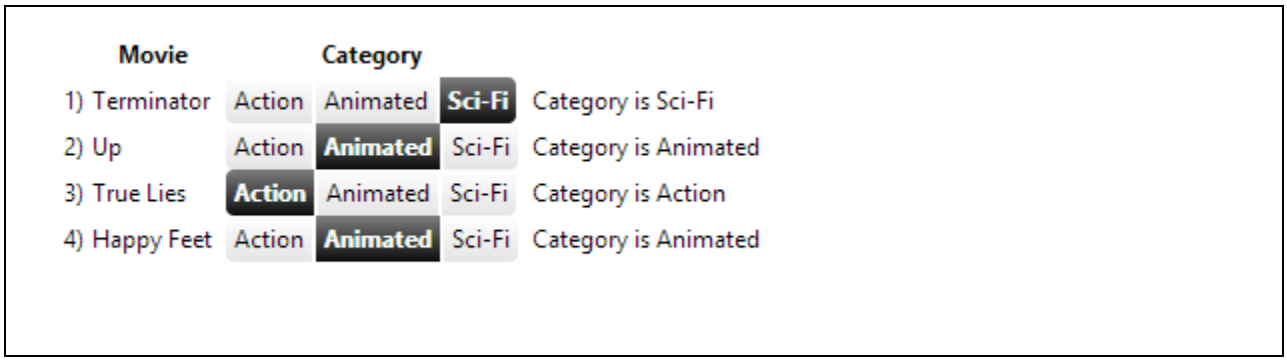
```
<script type="text/javascript">
    ({ searchText: '' })
</script>
<div>
   <input type="search" placeholder="Search For" atom-value="$[scope.searchText]"/>
</div>
<div
    atom-type="AtomItemsControl"
    atom-items="[ AtomPromise.json('/db/products', { name: $scope.searchText }) ]">
```

We have to initialize scope member `searchText` to an empty string. This is important, because if `searchText` is undefined all products will not be loaded and promise will not be set.

We are storing search text box's value in `$scope.searchText` which is used in AtomPromise.json as parameter. This is then converted to query string and sent to server. Great part is, any time text is modified, and items will be reloaded from server automatically.

14.1.4 Table Template

To make AtomItemsControl look like table (With column headers etc.), you have to mark an element as `itemsPresenter` which will contain `itemTemplate`. Here `tbody` element becomes `itemsPresenter` (an element which holds the items) and its immediate child becomes `itemTemplate`.



In above sample, we want to use table HTML element to display same list in tabular format. Also note that we are demonstrating nested scope in each row. Each row contains a toggle button bar that is bound to category of the movie and the last column displays selected category. However, in this example, we are using `$scope.categoryList.value` which is equivalent to `$data.MovieCategory`, but here you can

see that in nested scope, each scope evaluates expressions in isolation without interfering with other sibling scope.

```
<script type="text/javascript">
    ({
        categories: [
            { label: 'Action', value: 'Action' },
            { label: 'Animated', value: 'Animated' },
            { label: 'Sci-Fi', value: 'Sci-Fi' }
        ]
    })
</script>

<div
    atom-type="AtomItemsControl"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }">
    <table>
        <thead>
            <tr>
                <th></th>
                <th>Movie</th>
                <th>Category</th>
                <th></th>
            </tr>
        </thead>
        <tbody
            atom-presenter="itemsPresenter">
            <tr
                atom-template="itemTemplate">
                <td atom-text="{($scope.itemIndex + 1) + ')'}"></td>
                <td atom-text="{$data.MovieName}"></td>
                <td>
                    <span
                        atom-type="AtomToggleButtonBar"
                        atom-name="categoryList"
                        atom-items="{$appScope.categories}"
                        atom-value="$[data.MovieCategory]"></span>
                </td>
                <td
                    atom-text="['Category is ' + $scope.categoryList.value ]">
                </td>
            </tr>
        </tbody>
    </table>
</div>
```

In this example, you can see that `AtomToggleButtonBar` is named as "`categoryList`" and its selected value is displayed in last column, every item is in its own scope so we can bind properties of control with name.

14.2 List Box

`AtomListBox` is derived from `AtomItemsControl`, it acts as `ListBox` and `DataGrid`. Providing simple `itemTemplate` makes it work like `ListBox` and providing Table as template makes it look like `DataGrid`. Other classes like `AtomToggleButtonBar` and `AtomLinkBar` are derived from `AtomListBox` and their styles and templates are different.

```
<div
    atom-type="AtomListBox"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }"
    atom-name="movieList">
    <div
        atom-template="itemTemplate">
        <span atom-text="{ ($scope.itemIndex + 1) + ') ' }"></span>
        <span atom-text="{ $data.MovieName }"></span>
    </div>
</div>

<div>
    <span>Selected Movie: </span>
    <span
        atom-text="[($scope.movieList.selectedItem.MovieName || '')]" >
    </span>
    <span
        atom-text="['('+($scope.movieList.selectedItem.MovieCategory||'')+')']" >
    </span>
</div>
```

1) Terminator
2) Man of Steel
3) Iron Man
4) Avatar
5) Up
6) How to train your dragon
7) True Lies
8) Commander
9) Rumble in the Bronx
10) Happy Feet
11) Dumb & Dumber

Selected Movie: Iron Man (Sci-Fi)

14.2.1 Custom Selector

By default, `AutoSelectOnClick` is true, it means, when any part of item is clicked, item will be selected. However if you need only one button or checkbox to work for selection, then it can be customized in following ways.

> Note: To set any atom property as false, it must always be set in curly braces so that we identify it as JavaScript expression, otherwise in JavaScript string with "false" is considered as true.

14.2.1.1 Button as Selector

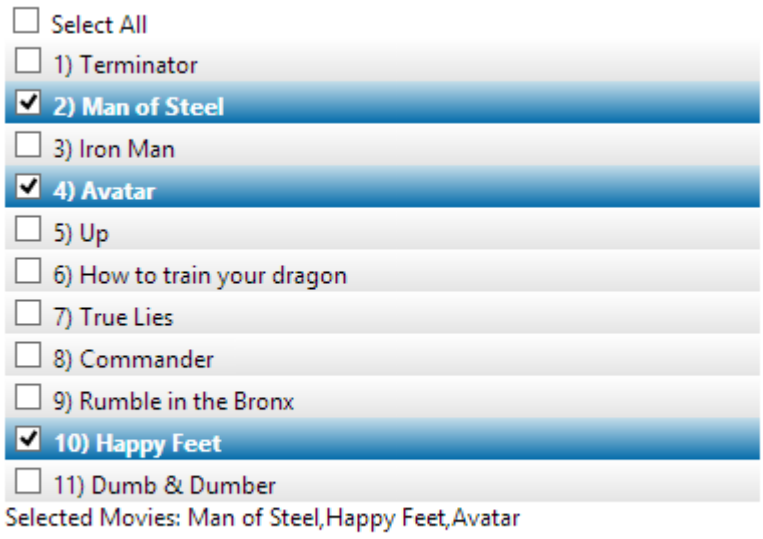
```
<div
    atom-type="AtomListBox"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }"
    atom-name="movieList"
    atom-auto-select-on-click="{ false }">
    <div
        atom-template="itemTemplate">
        <button atom-event-click="{$owner.templateParent.selectCommand}">Select</button>
        <span atom-text="{ ($scope.itemIndex + 1) + ') ' }"></span>
        <span atom-text="{ $data.MovieName }"></span>
    </div>
</div>
```

`AtomListBox` contains `selectCommand`, which can be used for click event, when invoked, it performs toggle selection.

14.2.1.2 Checkbox as Selector for Multiple Selection

For multiple selection, on tablet or mobile kind of devices, absence of real keyboard limits our ability to provide ctrl+ click to create multiple selection. However, checkboxes represent perfect candidate for multiple selection. And in Web Atoms, it is the easiest way to provide selection.

```
<input
    type="checkbox"
    atom-checked="$[scope.movieList.selectAll]"/> Select All

<div
    atom-type="AtomListBox"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }"
    atom-name="movieList"
    atom-auto-select-on-click="{ false }"
    atom-allow-multiple-selection="true"
    atom-value-path="MovieName">
    <div
        atom-template="itemTemplate">
        <input type="checkbox" atom-checked="$[scope.itemSelected]"></input>
        <span atom-text="{ ($scope.itemIndex + 1) + ') ' }"></span>
        <span atom-text="{ $data.MovieName }"></span>
    </div>
</div>

<div>
    <span>Selected Movies: </span>
    <span atom-text="[($scope.movieList.value || '')]" ></span>
</div>
```



As you can see, we provide `selectAll` and `itemSelected` properties to create a perfect multiple selection pattern.

14.2.2 DataGrid

As we have seen, we can simply change template to turn `AtomListBox` into a `DataGrid`.

However to turn `AtomListBox` into `DataGrid`, we need to little more work here. We need a sorting mechanism. To add sorting, we have created a special control `AtomSortableColumn` which wraps sorting of `DataGrid` functionality.

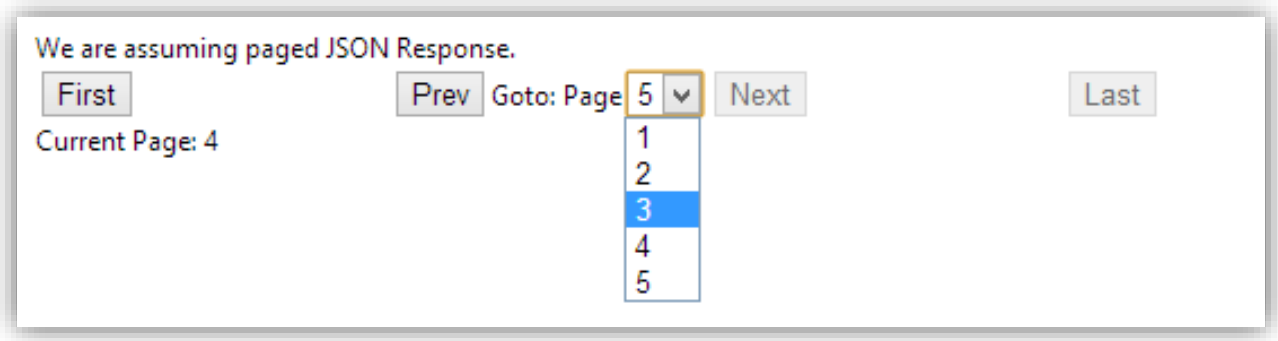
- First you need a `sortPath` scope variable set to default Sorting Column along with direction.
- For performing client side sorting, you can set `atom-sort-path` of `AtomListBox` to `sortPath` of scope.
- For server side sorting, you can ignore `atom-sort-path` and bind `$scope.sortPath` into `AtomPromise` to perform server side sorting.
- Each column's header, usually a `th` element, must have control `AtomSortableColumn` applied to it.
- `AtomSortableColumn` should have a label.
- And `atom-sort-field` which specifies sort path to sort by current column.
- And most important, atom-value should be set to `$scope.sortPath`, this two way binding manages the selection automatically and displays visual sorting direction as well as changes as necessary.
- By default, `atom-default-direction` is set to `asc`, however for columns containing Date etc., if you want to sort them by descending by default, you can manually set `atom-default-direction` to `desc`;

```
<script type="text/javascript">
    ({
        sortPath: 'MovieName asc'
    })
</script>

<div
    atom-type="AtomListBox"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }"
    atom-name="movieList"
    atom-auto-select-on-click="{ false }"
    atom-allow-multiple-selection="true"
    atom-value-path="MovieName"
    atom-sort-path="[$scope.sortPath]">
    <table>
        <thead>
            <tr>
                <th>
                    <input
                        type="checkbox"
                        atom-type="AtomCheckBox"
                        atom-is-checked="$[scope.movieList.selectAll]"/>
                </th>
                <th
                    atom-type="AtomSortableColumn"
                    atom-label="Movie"
                    atom-sort-field="MovieName"
                    atom-value="$[scope.sortPath]"></th>
                <th
                    atom-type="AtomSortableColumn"
                    atom-label="Category"
                    atom-sort-field="MovieCategory"
                    atom-value="$[scope.sortPath]"></th>
            </tr>
        </thead>
        <tbody
            atom-presenter="itemsPresenter">
            <tr atom-template="itemTemplate">
                <td><input type="checkbox" atom-type="AtomItemSelector"/></td>
                <td atom-text="{ $data.MovieName }"></td>
                <td atom-text="{ $data.MovieCategory }"></td>
            </tr>
        </tbody>
    </table>
</div>
```

14.3 Data Pager

We often require paging of data sets. Paging requirements vary based on type of data and user's navigation habits. So we have created fully customizable Data Pager, which lets you display paging UI very easily and you can customize it to any way you want it.

```
<script type="text/javascript">
    ({
        start:0,
        list: {
            items: [{ label:'1' }, {label:'2'}],
            total: 10,
            merge: true
        }
    })
</script>

    <div>We are assuming paged JSON Response contains items and total properties.</div>

<div
    atom-type="AtomDataPager"
    atom-items="{$scope.list}"
    atom-items-path="items"
    atom-total-path="total"
    atom-page-size="2"
    atom-current-page="$[scope.start]"
    style="width:500px;"
    ></div>

<div
    atom-text="['Current Page: ' + $scope.start]"></div>
```

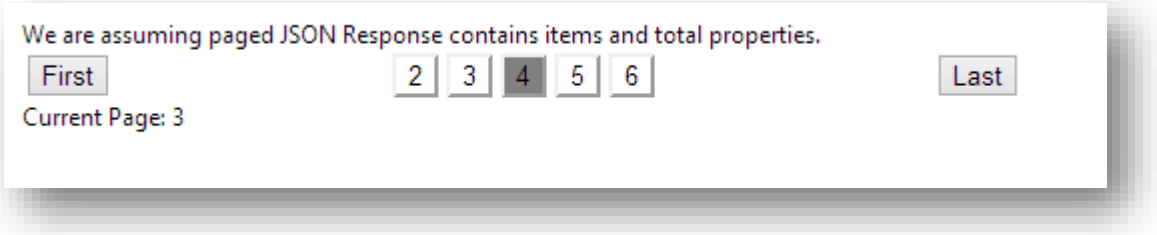


You may have to multiply start with size in case if your data store expects record number to skip instead of pages.

This is the default layout of Data Pager, we will show you a different layout so that you can easily customize look and feel.

The following example will customize Data Paget to display individual links to pages, however the page Array has been sliced to length of only 5 items.

```
<script type="text/javascript">
    ({
        start: 0,
        list: {
            items: [{ label: '1' }, { label: '2' }],
            total: 100,
            merge: true
        }
    })
</script>

    <div>We are assuming paged JSON Response contains items and total properties.</div>

<div
    atom-type="AtomDataPager"
    atom-items="{$scope.list}"
    atom-items-path="items"
    atom-total-path="total"
    atom-page-size="2"
    atom-current-page="$[scope.start]">
    <span
        atom-template="template"
        atom-type="AtomItemsControl"
        atom-items="[($owner.templateParent.pages).slice( Math.max(0, $owner.templateParent.currentPage-2),
$owner.templateParent.currentPage +10).slice(0,5)]">

        <button
            atom-is-enabled="[$owner.templateParent.pages.length > 1 && $owner.templateParent.currentPage]"
            event-click="{$owner.templateParent.goFirstCommand}"
            style="float:left">
                First
        </button>

        <span atom-presenter="itemsPresenter">
            <button
                atom-template="itemTemplate"
                atom-text="{$data.label}"
                atom-event-click="{ { appScope: { start: $data.value }} }"
                style-background-color="[ $data.value == $appScope.start ? 'gray' : 'inherit' ]"></button>
        </span>

        <button
            event-click="{$owner.templateParent.goLastCommand}"
            atom-is-enabled="[$owner.templateParent.pages.length > 1 && $owner.templateParent.currentPage &lt;
$owner.templateParent.pages.length -1 ]"
            style="float:right">
                Last
        </button>
    </span>
</div>

<div
    atom-text="['Current Page: ' + $scope.start]"></div>
```



This is just a sample, you can customize the template to any way you want it. Another alternative would be to display all the pages, but simply use relative width to crop it to a viewable viewport, in which you can customize the look and feel and also animate it little.

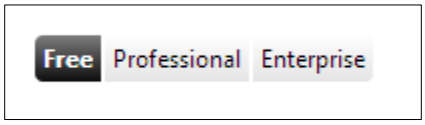
14.4 Combo Box

`AtomComboBox` can only be created with SELECT HTML Element. This control manages selection and operates as regular combo box. Since it is derived from `AtomItemsControl`, it manages binding completely by itself. You can refer Cascaded Selections for a sample.

14.5 Toggle Button Bar

Toggle Button Bar can have only one selected item, and it has first item selected by default if none selected. This control is derived from `AtomListBox`, but it applies its own styles.

```
<script type="text/javascript">
    ({
        accountTypes: [
            { label: 'Free', value: 0 },
            { label: 'Professional', value: 99 },
            { label: 'Enterprise', value: 999 }
        ]
    })
</script>

<span
    atom-type="AtomToggleButtonBar"
    atom-items="{$scope.accountTypes}"
    atom-value="$[data.AccountType]"></span>
```



In this example, let us consider that we want user to sign up for different type of accounts, and we give them choice to select one. `AtomToggleButtonBar` choose first item as default if none selected already.

Instead of drop downs, for smaller items `AtomToggleButtonBar` is easier for users to choose required item. You can change its looks simply by applying one property `atom-show-tabs` as true.

And it appears as tab.

If you combine, Toggle Button Bar and View Stack, you have your own Tab Control.

15 BUILDING HTML USER INTERFACE

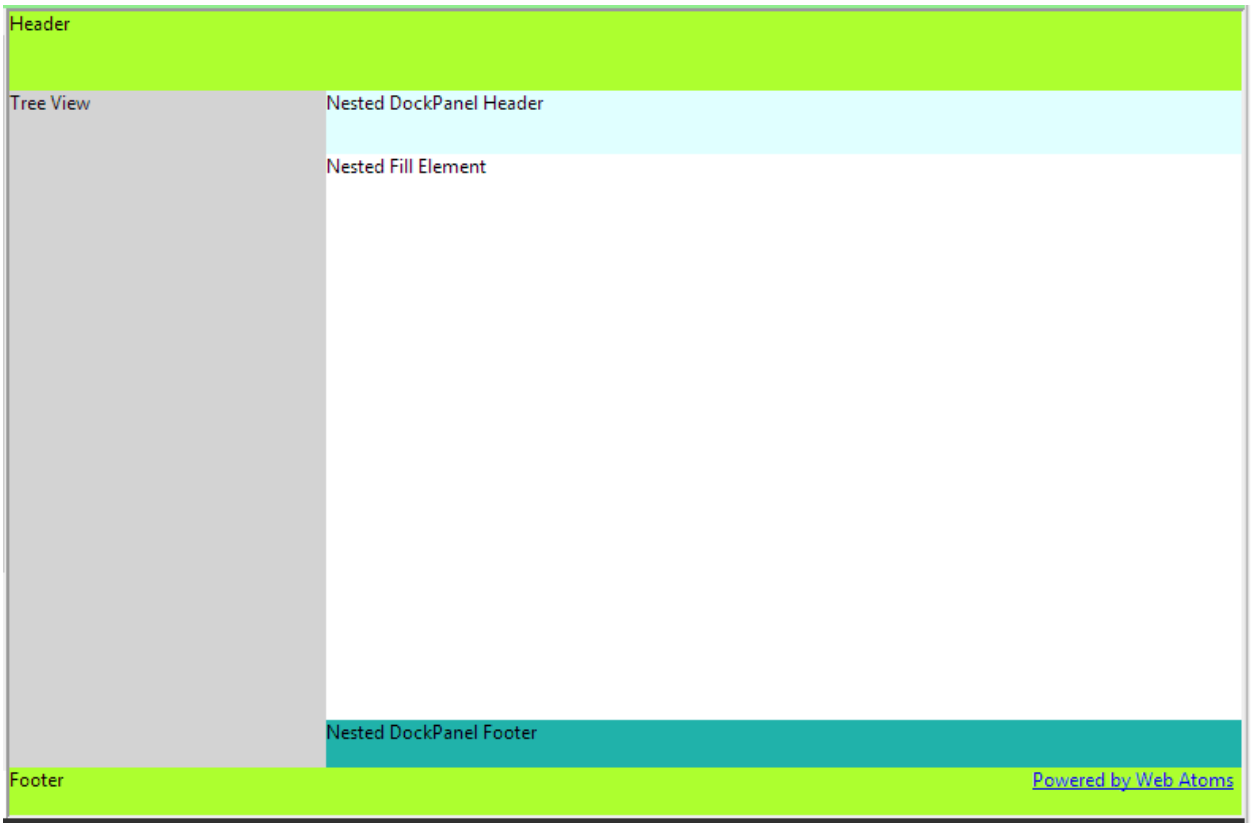
15.1 Dock Panel

Dock panel provides basic docking functionalities and it resizes its children automatically. Following example displays nested Dock Panels, you can resize browser to see how resizing works perfectly on all browsers. It also resizes correctly while orientation of mobile devices changes.

```
<div
    atom-type="AtomApplication">
    <div
        atom-dock="Top"
        style="height:50px; background-color:greenyellow">
        Header
    </div>
    <div
        atom-dock="Left"
        style="width:200px; background-color: lightgray">
        Tree View
    </div>
    <div
        atom-dock="Fill"
        atom-type="AtomDockPanel">
        <div
            atom-dock="Top"
            style="height:40px; background-color: lightcyan">
            Nested DockPanel Header
        </div>
        <div
            atom-dock="Fill">
            Nested Fill Element
        </div>
        <div
            atom-dock="Bottom"
            style="height:30px; background-color: lightseagreen">
            Nested DockPanel Footer
        </div>
    </div>
    <div
        atom-dock="Bottom"
        style="height:30px; background-color:greenyellow">
        Footer
        <a
            href="http://webatomsjs.neurospeech.com"
            target="_blank"
            style="right:5px; position:absolute">Powered by Web Atoms</a>
    </div>
</div>
```

Dock properties are not bindable as of now.

Note: `AtomDockPanel` should only be used for top level docking, as it consumes lots of resources, using too many dock panels will slow down the browser.



You must always use `AtomDockPanel` to create docking layout instead of mixing with any jQuery plugin, because this control will substitute best available method for docking based on new CSS availability.

15.2 View Stack

`AtomViewStack` displays only the selected child as full view and all other children are hidden. `AtomViewStack` can host multiple children and only one children can be visible based on some state. `AtomViewStack` is derived from `AtomItemsControl`, so it can also create individual children based on the items set.

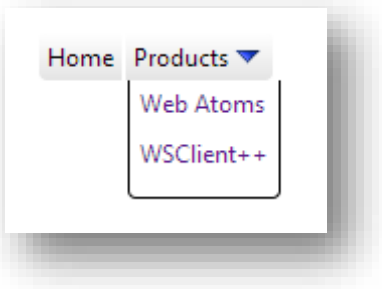
```
<script type="text/javascript">
    ({
        links: [
            { label: "Home", value: "home" },
            { label: "About", value: "about" },
            { label: "Contact", value: "contact" },
        ],
        view: "home"
    })
</script>

<span
    atom-dock="Top"
    atom-type="AtomToggleButtonBar"
    atom-name="buttonBar"
    atom-items="{ $scope.links }"
    atom-value="$[scope.view]"
    style="height:30px;text-align:center"></span>

<div
    atom-dock="Fill"
    atom-type="AtomViewStack"
    atom-selected-index="[$scope.buttonBar.selectedIndex]">
    <div style="background-color:lightyellow">
        This is home page....
    </div>
    <div style="background-color:lightblue">
        This is About page....
    </div>
    <div style="background-color:lightgreen">
        This is Contact page....
    </div>
</div>
```

15.3 Link Bar (Menu Bar)

`AtomLinkBar` is same as `AtomToggleButtonBar`, but the default button is selected based on the current page's URI scheme. This is useful control in creating top navigation links usually found on any web applications, where the current page or category is highlighted.

`AtomLinkBar` can display menu for children items as well.

```
<script type="text/javascript">
    ({
        links: [
            {
                label: "Home",
                value: "http://neurospeech.com/"
            },
            {
                label: "Products", value: "",
                links: [
                    {
                        label: "Web Atoms",
                        value: "http://webatomsjs.neurospeech.com"
                    },
                    {
                        label: "WSClient++",
                        value: "http://wsclient.neurospeech.com"
                    }
                ]
            }
        ]
    })
</script>

<div
    atom-type="AtomLinkBar"
    atom-items-path="links"
    atom-items="{$scope.links}">
</div>
```



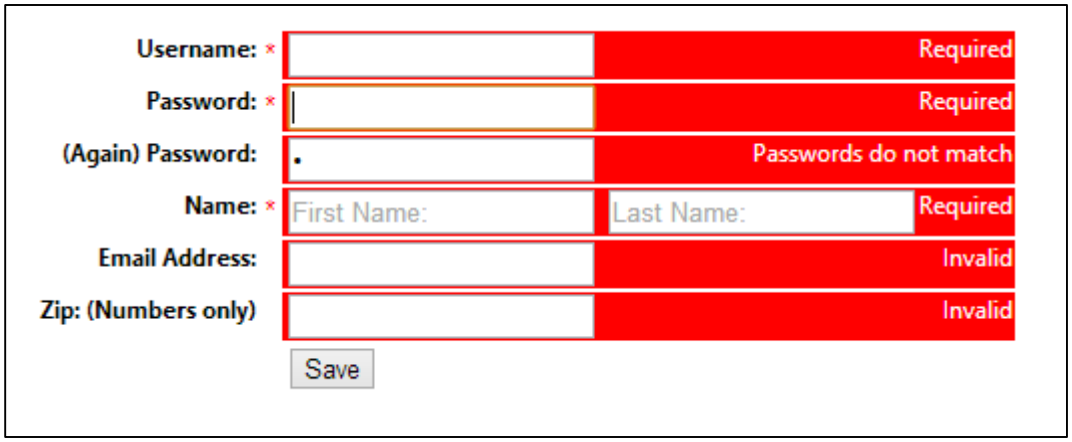
15.4 Form Layout

`AtomFormLayout` is a special control, which encapsulates general form layout, i.e. display a label and required "*" and error label for erroneous data. `AtomFormLayout` encapsulates every control within a class called `AtomFormField`, you need not define this type, and it is automatically created by `AtomFormLayout`.

```
<div
    atom-type="AtomForm"
    atom-post-url="/Url/Insert"
    atom-data="{ { FirstName: '', LastName: '', Password: '', Password2:'' } }"
    atom-success-message="Information Saved Correctly">
    <div
        atom-type="AtomFormLayout">

        <input
            type="text"
            atom-label="Username:"
            atom-required="true"
            atom-value="$[data.Username]"/>

        <input
            type="password"
            atom-label="Password:"
            atom-value="$[data.Password]"
            atom-required="true"/>

        <input
            type="password"
            atom-label="(Again) Password:"
            atom-value="$[data.Password2]"
            atom-is-valid="[$data.Password == $data.Password2]"
            atom-error="[$owner.isValid ? '' : 'Passwords do not match']"/>

        <span
            atom-label="Name:"
            atom-required="true"
            atom-is-valid="[$data.FirstName && $data.LastName]">
            <input
                placeholder="First Name:"
                type="text"
                atom-value="$[data.FirstName]" />

            <input
                placeholder="Last Name:"
                type="text"
                atom-value="$[data.LastName]" />
        </span>

        <input type="submit" value="Save" />

    </div>
</div>
```

Every children of `AtomFormLayout` can define, `atom-label`, `atom-required`, `atom-is-valid`, `atom-data-type`. `atom-regex`, `atom-field-value` and `atom-field-class`.

atom-label	Label will be displayed on left side of form with right aligned by default.
atom-required	This will setup default validation for field to be required, if set true, either value of element or atom-field-value cannot be empty.
atom-is-valid	In case when you have multiple required elements in one field, you can use this to bind to a valid expression, or you can use this to bind to custom expression that evaluate to true.
atom-data-type	If you setup data-type as email, field will be validated for perfect email address.
atom-regex	If you setup regex, it will be tested against atom-field-value or value of element.
atom-field-value	If child control does not have value property, then you can bind this to any property of data that will be validated.
atom-field-visible	You can bind this property to hide/show the field based on some Boolean condition
atom-field-class	CSS class that should be applied to entire Field Row.

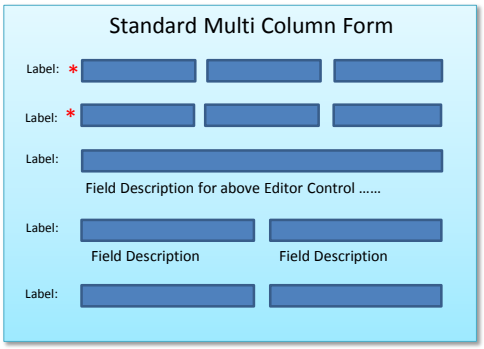


15.5 Form Grid Layout

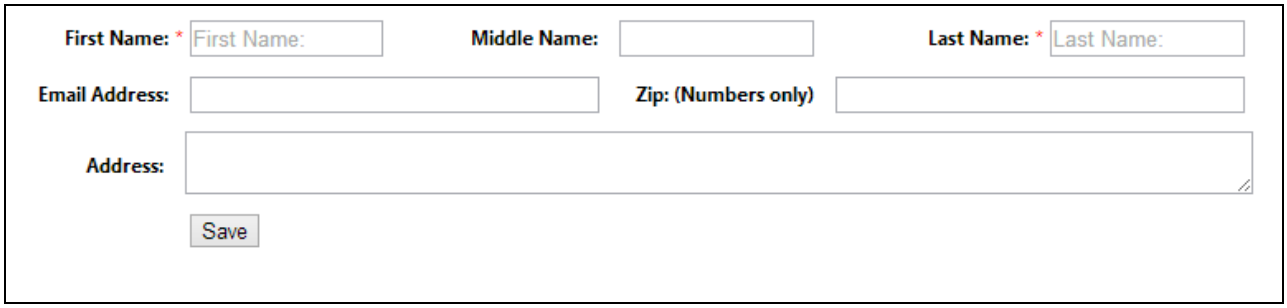
As we have seen the Form Layout manages simple form layout, however when we need multi column layout, it becomes very difficult to manage it in HTML and it involves complex layout CSS. To make things easier, we have created Form Grid Layout which lets you add multiple cells in one column and form design stays consistent as needed.

Standard Form Design

- Multi Column
- Variable Columns per Row
- Required Red Asterisk (*)
- Description at the Bottom



So we want to achieve following layout. If you look carefully, 1st row has three columns, 2nd row has two columns and last two rows have single column elements.



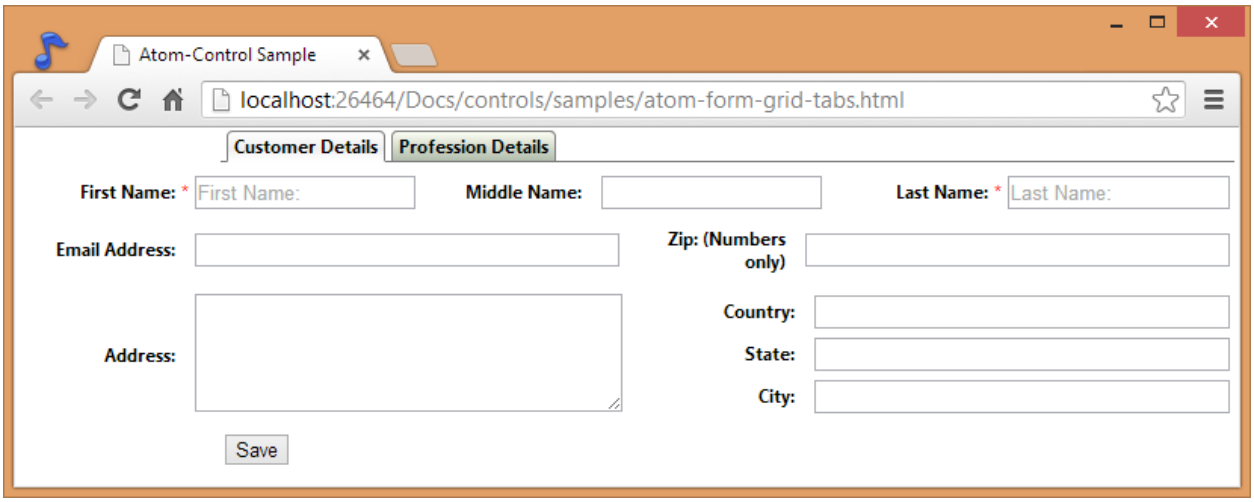
Doing this in HTML will require you to either create lots of nested tables or write complex CSS. However there is now way you can write a dynamic CSS which will change its width as per screen changes. `AtomFormGridLayout` does this by automatically creating tables as needed. However, there is no possible CSS solution for such layout which resizes automatically.

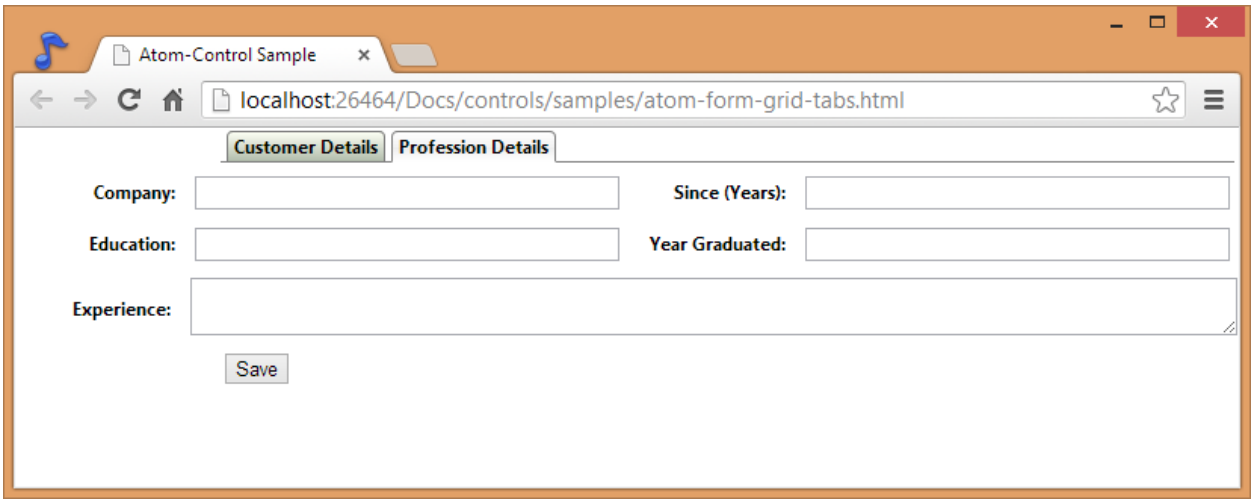
```
    <div
        atom-type="AtomFormGridLayout"
        atom-min-label-width="120">

        <span atom-type="AtomFormRow">
            <input
                atom-label="First Name:"
                atom-required="true"
                type="text"
                atom-value="$[data.FirstName]" />
            <input
                atom-label="Middle Name:"
                atom-value="$[data.MiddleName]"/>
            <input
                atom-label="Last Name:"
                atom-required="true"
                type="text"
                atom-value="$[data.LastName]" />
        </span>

        <span atom-type="AtomFormRow">
            <input
                atom-label="Email Address:"
                type="text"
                atom-data-type="email"
                atom-value="$[data.EmailAddress]"/>

            <input
                atom-label="Zip: (Numbers only)"
                type="text"
                atom-regex="/[0-9]+/"
                atom-value="$[data.ZipCode]"/>
        </span>

        <textarea
            atom-label="Address:"
            atom-value="$[data.Address]"></textarea>

        <input type="submit" value="Save" />

    </div>
```

15.6 Form Grid Layout with Tabs

Form Grid Layout is fun, and there is even more fun with Tabs, usually if the form is very big, you can divide form screens into different tabs.

```
        <div atom-type="AtomFormTabControl">
            <div atom-type="AtomFormTab" atom-label="Customer Details">
                <span atom-type="AtomFormRow">
                    <input atom-label="First Name:"
                        atom-required="true"
                        atom-value="$[data.FirstName]" />
                    <input atom-label="Middle Name:"
                        atom-value="$[data.MiddleName]" />

                    <input atom-label="Last Name:"
                        atom-required="true"
                        atom-value="$[data.LastName]" />
                </span>
                <span atom-type="AtomFormRow">
                    <input atom-label="Email Address:"
                        atom-data-type="email"
                        atom-value="$[data.EmailAddress]" />
                    <input atom-label="Zip: (Numbers only)"
                        atom-regex="/[0-9]+/"
                        atom-value="$[data.ZipCode]" />
                </span>
                <span atom-type="AtomFormRow">
                    <textarea
                        atom-label="Address:"
                        atom-value="$[data.Address]"></textarea>
                    <div atom-type="AtomFormGridLayout">
                        <input atom-label="Country:"/>
                        <input atom-label="State:"/>
                        <input atom-label="City:"/>
                    </div>
                </span>
            </div>
            <div atom-type="AtomFormTab" atom-label="Profession Details">
                <span atom-type="AtomFormRow">
                    <input atom-label="Company:"
                        atom-value="$[data.Company]" />
                    <input atom-label="Since (Years):"
                        atom-regex="/[0-9]+/"
                        atom-value="$[data.SinceYears]" />
                </span>
                <span atom-type="AtomFormRow">
                    <input atom-label="Education:"
                        atom-value="$[data.Education]" />
                    <input atom-label="Year Graduated:"
                        atom-regex="/[0-9]+/"
                        atom-value="$[data.GraduationYear]"/>
                </span>
                <textarea atom-label="Experience:"
                    atom-value="$[data.Address]"></textarea>
            </div>
        </div>
```





As you can see, the form layout remains consistent throughout the form including Tabs and that probably covers all aspects of form design.

You should add `AtomFormTabControl` inside `AtomFormGridLayout` and also you can add `AtomFormTab` and set its label.

Label property is not bindable at this time, however in future it will be bindable.

15.7 Form Items

15.7.1 HTML Elements

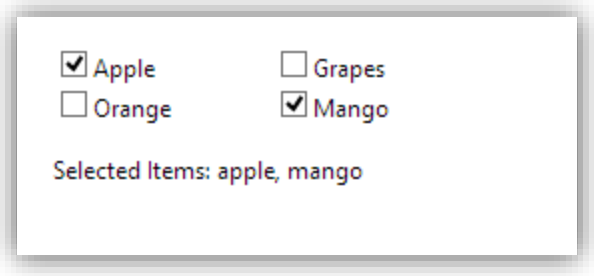
As you have seen in previous example, you can use any default HTML form input elements that is supported by the browser and binding should be applied on value attribute except for checkboxes.

15.7.2 CheckBoxList

`AtomCheckBoxList` is inherited from `AtomListBox`, but by default it allows multiple selection and automatically provides comma separated values as value property. You can control layout property by giving it a table layout, which accepts number of columns, cell width and cell height parameters. In addition, you can also control your layout by overriding atom-check-box-list css styles.

```
<script type="text/javascript">
    ({
        items: [
            { label: 'Apple', value: 'apple' },
            { label: 'Orange', value: 'orange' },
            { label: 'Grapes', value: 'grapes' },
            { label: 'Mango', value: 'mango' }
        ],
        fruits: ''
    })
</script>

<div
    atom-type="AtomCheckBoxList"
    atom-items="{$scope.items}"
    atom-value="$[scope.fruits]"
    atom-layout="{Atom.tableLayout(2,100,20)}"
    ></div>

<br />

<span
    atom-text="[$scope.fruits ? ('Selected Items: ' + $scope.fruits) : '(Please Select
Items)']"
    style-color="[$scope.fruits ? 'inherit' : 'red']"></span>
```

Selected items are separated by ", " and are available as value property. In above example, value property is stored in scope.



15.7.3 Date Control

This control is simple, old style, three drop down for Year, Month and Day for selecting Date. Calendar style date controls are little complicated to choose for longer year range. But this is still the most simple way to choose date yet.

Value property of `AtomDateControl` accepts date as JavaScript Date object, or Microsoft Rest Date Format in `/Date(334343433)/` form or simple parsable Date format.

```
<script type="text/javascript">
    ({
        date: new Date()
    })
</script>

<div
    atom-type="AtomDateControl"
    atom-start-year="{ -5 }"
    atom-end-year="{ +20 }"
    atom-value="$[scope.date]"></div>

<div
    atom-text="[$scope.date]"></div>
```

In this control, date is finally stored as JavaScript Date Object and it is sent to JSON as `/Date(3434333)/` format.

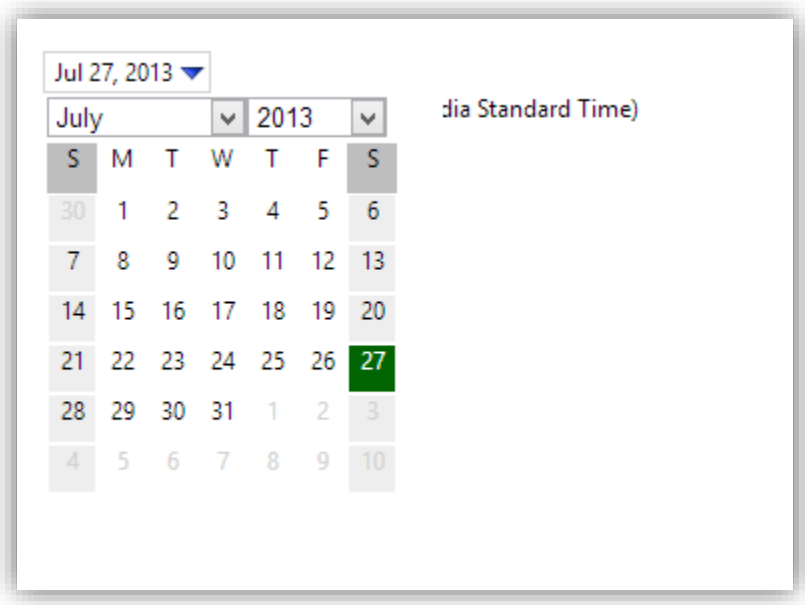
15.7.4 Date Field

This control hosts a Date Selector field in Calendar format.

Value property of `AtomDateField` accepts date as JavaScript Date object, or Microsoft Rest Date Format in `/Date(334343433)/` form or simple parsable Date format.

```
<script type="text/javascript">
    ({
        date: new Date()
    })
</script>

    <div
        atom-type="AtomDateField"
        atom-start-year="{ -5 }"
        atom-end-year="{ +20 }"
        atom-value="$[scope.date]">
    </div>
    <div
        atom-text="[$scope.date]">
    </div>
```

Year drop down is loaded with range specified by `startYear` and `endYear` properties. Both these properties are relative to current Year and they are added to current Year and the range is loaded automatically. In above example, you can see that year starts 5 years back in the past till 20 years in the future. Also note, in order to make things simpler, we have set values in curly braces, so values are recognized as proper JavaScript numbers.

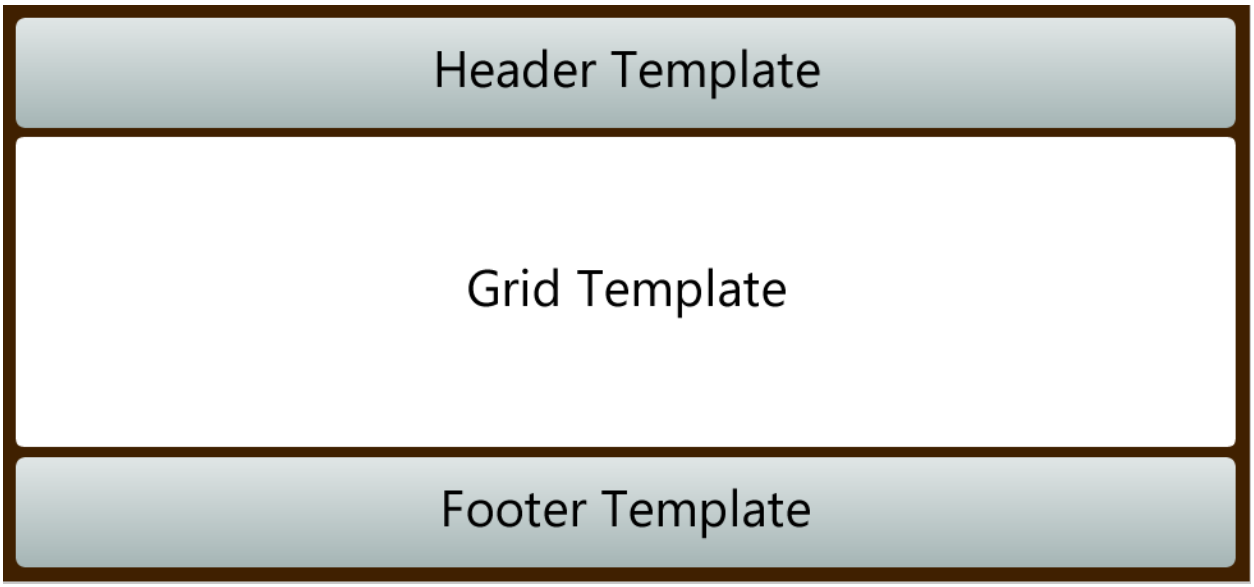


15.8 Navigator List

Navigator list is derived from `AtomListBox` and it adds UI Navigation similar to what is found in iPhone mobile application. This control simplifies Master Detail View. When we select an item or add new item, detail view is slided in. And by clicking on back button, list will appear again.

Navigator list contains Data Pager as well.

List view is outlined as shown in following image. This is default view, navigator list starts with following view. You can customize template as per your requirements.



When we click on an item, detail view opens and it is outlined as follow.

- Detail Header Template is required.

- Either Detail Template or Detail Url must be defined. Detail URL should be bound to selected item's property which will be loaded when selected item will be selected.

```
<div
    atom-type="AtomNavigatorList"
    atom-new-item="{ { MovieName: '', MovieCategory: '' } }"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }"
    atom-name="movieList"
    style="width:400px; height:400px; margin:50px;">
    <span atom-template="headerTemplate">
        <input placeholder="Search Movies" atom-value="$[scope.searchText]" />
    </span>
    <table atom-template="gridTemplate">
        <thead>
            <tr>
                <th>Name</th>
            </tr>
        </thead>
        <tbody atom-presenter="itemsPresenter">
            <tr atom-template="itemTemplate">
                <td atom-text="{$data.MovieName}"></td>
            </tr>
        </tbody>
    </table>
    <span
        atom-template="detailHeaderTemplate"
        atom-text="['Movie: ' + $data.MovieName]">
    </span>
    <div
        atom-template="detailTemplate"
        atom-type="AtomFormLayout">
        <input
            atom-label="Movie Name:"
            atom-value="$[data.MovieName]"/>
        <input
            atom-label="Movie Category:"
            atom-value="$[data.MovieCategory]"/>
        <button
            atom-event-click="{$owner.templateParent.backCommand}"
            >Save</button>
    </div>
</div>
```

In above sample, `detailTemplate` is used to display the details. You can hide Search and Add New buttons by customizing templates. New Item property is used to create a new object with default values,

For New Item property, every property in object must be initialized to some default empty values since undefined is not a bindable value.

As this is derived from `AtomListBox`, you can change item template and table headers for sorting.

Following code snippet illustrates how to use Data Pager in footer.

```
<div
    atom-type="AtomNavigatorList"
    atom-new-item="{ { MovieName: '', MovieCategory: '' } }"
    atom-items="{ AtomPromise.json('/docs/controls/samples/movie-list.json') }"
    atom-name="movieList"
    style="width:400px; height:400px; margin:50px;">

    <span atom-template="headerTemplate">
        <input placeholder="Search Movies" atom-value="$[scope.searchText]" />
    </span>

    <div
       atom-template="footerTemplate"
       atom-type="AtomDataPager"
       atom-items="[$owner.templateParent.items]"
       atom-current-page="$[$owner.templateParent.currentPage]"
       atom-page-size="$[$owner.templateParent.pageSize]"></div>

    <table atom-template="gridTemplate">
        <thead>
            <tr>
                <th>Name</th>
            </tr>
        </thead>
        <tbody atom-presenter="itemsPresenter">
            <tr atom-template="itemTemplate">
                <td atom-text="{$data.MovieName}"></td>
            </tr>
        </tbody>
    </table>

    <span
        atom-template="detailHeaderTemplate"
        atom-text="['Movie: ' + $data.MovieName]">
    </span>

    <div
        atom-template="detailTemplate"
        atom-type="AtomFormLayout">
        <input
            atom-label="Movie Name:"
            atom-value="$[data.MovieName]"/>
        <input
            atom-label="Movie Category:"
            atom-value="$[data.MovieCategory]"/>
        <button
            atom-event-click="{$owner.templateParent.backCommand}"
            >Save</button>
    </div>
</div>
```

15.9 Window

`AtomWindow` can host one div or it can host an `IFrame` and contains a close button along with a window frame. Elements behind `AtomWindow` are not accessible as Window is modal by default.

```
<!-- setting control properties requires square brackets
    because window appears after the button, so window was not
    created and will be updated by framework later on. Using
    one-time binding will not work for controls defined later
    in the visual hierarchy -->
<button
    atom-type="AtomButton"
    atom-next="[$scope.theWindow.openCommand]"
    atom-data="{ { label:'Open Window Button' } }"
    >Open Window</button>


<div
    atom-type="AtomWindow"
    atom-name="theWindow"
    atom-title="This is a window">
    <!-- To host elements, div must be a window template -->
    <!-- As opener Data will change after the window was
        created, using curley braces will not work for openerData-->
    <div
        atom-template="windowTemplate"
        >
        This is inside the window.
        <div atom-text="[ 'Opener Data: ' + $owner.openerData.label ]"></div>
        <button
            atom-type="AtomButton"
            atom-next="{$scope.theWindow.closeCommand}"
            >Close</button>
    </div>
</div>
```

Important properties to note.

- `closeCommand` - you can bind window's close command to close button.
- `opener` - control that opened this window
- `openerData` - data property of window opener

15.9.1 Hosted IFrame

```
<!-- setting control properties requires square brackets
    because window appears after the button, so window was not
    created and will be updated by framework later on. Using
    one-time binding will not work for controls defined later
    in the visual hierarchy -->

<button
    atom-type="AtomButton"
    atom-next="[$scope.theWindow.openCommand]"
    atom-data="{ { label:'NeuroSpeech Frame', value: 'http://neurospeech.com' } }"
    >Open NeuroSpeech.com</button>

<button
    atom-type="AtomButton"
    atom-next="[$scope.theWindow.openCommand]"
    atom-data="{ { label:'WebAtoms Frame', value: '/webatoms/docs' } }"
    >Open WebAtoms</button>

<!--
    openerData will be set after creation of window, so one way binding
    must be used
    -->
<div
    atom-type="AtomWindow"
    atom-name="theWindow"
    atom-title="[$owner.openerData.label]"
    atom-url="[$owner.openerData.value]"
    >
</div>
```

In this example, we are sending value through `openerData` property, we are setting `Title` and `Url` of `iFrame` window to its own `openerData` property, which will be set when any of the button will be clicked.

15.10 Wizard

Wizard is a set of views that allows you to go back and forth by clicking buttons “Next” and “Back”. Wizards are multi step UI actions, where you expect users to finish their data entry one step at a time. Although you can use a tabbed form, but still order of completion and database transactions per step gives you more control over tabs.

```
<div
    atom-type="AtomWizard"
    atom-name="theWizard"
    atom-local-scope="true">
    <!-- View Template Does Nothing, it just holds references of child steps -->
    <div atom-template="viewTemplate">
        <!-- Step 1 View-->
        <div atom-next="[$scope.step1Form.submitCommand]">
            <div
                atom-type="AtomForm"
                atom-name="step1Form"
                atom-next="{$localScope.owner.goNextCommand}">
                <div>This is step 1 of wizard</div>
                <div atom-type="AtomFormLayout">
                    <input
                        type="text"
                        atom-label="Name:"
                        atom-required="true"/>
                </div>
            </div>
        </div>
        <!-- Step 2 View-->
        <div atom-next="[$scope.step2Form.submitCommand]">
            <div
                atom-type="AtomForm"
                atom-name="step2Form"
                atom-next="{$localScope.owner.goNextCommand}">
                <div>This is step 2 of wizard</div>
                <div atom-type="AtomFormLayout">
                    <input
                        type="text"
                        atom-label="Email:"
                        atom-required="true"/>
                </div>
            </div>
        </div>
        <!-- Step 3 View-->
        <div>
            This is step 3 of wizard
        </div>
        <!-- Final View-->
        <div>
            This is the final step
        </div>
    </div>
</div>
```

The way Wizard works is, Wizard has a `viewTemplate` which contains multiple views for each step.

1. Each View has `atom-next` action set, which is invoked when user presses “Next” button on the wizard.
2. If `action-next` is not set, by default wizard will move to next step.
3. In case when we want to save the form data, we have to intercept `atom-next` of the view.
4. We will set `atom-next` of step view to `step1Form.submitCommand`
5. And we have set form’s `atom-next` to wizard’s `goNextCommand`.
6. In above sample, we have set `atom-next` to `step1Form.submitCommand`, which means when we click next button, form will be submitted.
7. After success of form post result, form’s `atom-next` will move wizard to next step.

`AtomWizard` has atom-next action set which will be invoked after the last step was finished.

The best use of wizard will be to put it inside an `AtomWindow` that will delete current state of `AtomWizard` and reset it back when user opens a window. If you host `AtomWizard` inside `AtomWindow`, you can set `closeWindow` command of `AtomWindow` as `atom-next` of `AtomWizard`.

16 DIFFERENCE BETWEEN SCOPE AND DATA

While creating an application or component, choosing between scope or data can be little confusing. But once you know following differences, you will be able to choose storage correctly.

Scope

- Scope stores UI reference with given name, so you can access other UI controls through scope.
- Top level Scope serializes primitive data types such as string, number and Boolean onto URL with hashtags, that helps in navigating back and forward in browser. Whatever that represents state of the user interface, but not the data, should be stored on scope. For example, which column to sort, Search query, and any layout related information like selected tab etc.
- Since scope is visually nested as well, you can access parent scope in children and you can access appScope from anywhere.
- Scope should be used to store information temporarily till lifespan of the page only.
- You can only add items to scope, but you cannot change scope itself, scope is initialized only by the framework.

Data

- Data can only be accessed through data property of current control. If not set, data will be inherited from visual parent.
- Data can be set by control, once set, all its descendants will access same data.
- Data does not have references to any visual parent what so ever.
- For forms processing, controls such as Form and Post Button only post contents of data property.
- Only the items you want to persist to database and that are not representing temporary UI state, should be stored in data.

17 LIBRARY REFERENCE

Web Atoms has many useful library functions, and as they are standardized, we will be including them in our library. You can certainly use undocumented methods, but they may be removed or replaced in future.

17.1 Atom Module

17.1.1 `Atom.get(target,property)` Method

`Atom.get` method is useful to inference property as property can be get_ function or it is a member.

17.1.2 `Atom.set(target,property,value)` Method

`Atom.set` method sets property of specified target, however this also invokes binding and forces each referenced binding to refresh the UI Element. That is the reason, we recommend using `Atom.set` over directly setting properties on any object.

17.1.3 `Atom.add(targetArray,item)` Method

`Atom.add` method adds given item in the `targetArray` at the end. This method also causes all controls to refresh which are bound to `targetArray`.

17.1.4 `Atom.insert(targetArray,index,item)` Method

`Atom.insert` method inserts item in the `targetArray` at the given index. This method also causes all controls to refresh which are bound to `targetArray`.

17.1.5 `Atom.remove(targetArray,item)` Method

`Atom.remove` method removes item from the `targetArray`, only if the item was found in the `targetArray`. If it was removed, it will then refresh all its corresponding bindings. This method will not throw or report any error if item was not found.

17.1.6 `Atom.clone` Method

Since Web Atoms uses distributed binding, each instance of object has certain hidden properties with prefix `_$_` and all these properties must not be serialized anywhere. To make cloning easier, we have provided convenient `Atom.clone` method which clones given object without any hidden properties. This method also converts JavaScript Date into `/DateISO(AtomJson)/` string.

17.1.7 `Atom.refresh(target,property)` Method

In case if the value of property is modified by some different routine and you want to force UI to update, then you can use `Atom.refresh` method.

17.1.8 `Atom.refreshArray(targetArray)` Method

Same as `Atom.refresh`, but this method is for refreshing Array.

17.1.9 `Atom.clearArray(targetArray)` Method

This method clears the array and refreshes all the bindings.

17.1.10 `Atom.url(url,query,hash)` Method

Contacting different parameters for URL requires proper encoding. `Atom.url` provides `url` encoding as well as it converts objects and arrays into JSON equivalent as well.

```
<a
    atom-href="{ Atom.url('/productList.php', { category: 'Arts and Crafts' }) }"></a>

<!-- Becomes -->
<a
    atom-href="/productList.php?category=Arts+and+Crafts"></a>
```

The reason, `url` method expects an anonymous object as second parameter is, it escapes each member and adds them up as proper query string.

17.1.10.1 `query` Parameter

`query` is of type anonymous object, each member will be escaped and added into final URL. This helps in binding individual query string parameter to some other scope variable.

17.1.10.2 `hash` Parameter

`hash` is same as query, but hash anonymous objects are appended as URL hash instead of query string. This `url` hash is important to initialize default scope values on destination page.

17.1.11 `Atom.time` Method

This method returns current time's milliseconds as integer, this is useful method to use inside any binding expression.

17.1.12 `Atom.refreshWindowCommand` Method

This method refreshes current method, however this is not a simple `location.reload()`, instead it forces server side refresh by appending current time as `_v` in query string to enforce refresh. This is useful command for binding that requires refreshing of entire page.

17.1.13 `Atom.merge( dest, src, update)` Method

Sometimes it is useful to merge some default values in data of any control while binding or initializing. This method merges all values from `src` into `dest` and returns the same `dest`. If update was true, merge will use `Atom.set` instead of simply setting value, which will cause refreshing of bindings.

17.1.14 `Atom.csv(array,path,separator)` Method

As we retrieve objects with some sort of label-value pattern, most of the times we need some comma separated values of internal properties. For example, if we retrieve list of countries, we need comma separated values of all country codes. In following example, countries array item has a property called `CountryCode`.

```
Atom.csv(countries,"CountryCode",", ");
```

17.1.15 `Atom.query` Method

Creates instance of `AtomQuery`, which can be used to query object with JSON Query Language Explained in `AtomQuery` section.

17.2 AtomPromise Module

AtomPromise methods are mostly wrapper around AtomPromise object which has set of standard JavaScript Promise methods and a little more common functionality.

17.2.1 JSON Post Encoding

By default, HTTP POST are encoded as `formModel=JSON` in HTML form encoded way to support previous ASP.NET Web Server. And on server side, you could use `Request.Form["formModel"]` to retrieve JSON equivalent, and you can then deserialize JSON into Dictionary in .NET.

This is done to preserve the null and empty values of string. Sending Form Key=Value pairs does not send hierarchical data and it does not send empty string values correctly.

To have your own serialization method, in case of PHP or any other framework, you have to use following different code snippets.

17.2.2 JSON Serialized as Form Value as `formModel` field (Default)

```
<script type="text/javascript">
    AtomConfig.ajax.jsonPostEncode = function (data, options) {
        options.contentType = 'application/x-www-form-urlencoded';
        return { formModel: JSON.stringify(data) };
    };
</script>
```

17.2.3 JSON Serialization

To change the default behavior, you can change `jsonPostEncode` as shown below, however this scheme does not work well with file upload.

```
<script type="text/javascript">
    AtomConfig.ajax.jsonPostEncode = function (data, options) {
        options.contentType = 'application/json';
        return JSON.stringify(data);
    };
</script>
```

17.2.4 `AtomPromise.json(url,query,options)` Method

This method combines `url` with query string (query parameter) by using **`Atom.url`** method and invokes jQuery's **`$.ajax`** method which is wrapped inside **`AtomPromise.ajax`** method. To post any data, data must be passed inside options parameter. However, please remember that data is first encoded with **`JSON.stringify`** as **`formModel`** and it is sent to jQuery's **`$.ajax`**.

17.2.4.1 `options` Parameter

`cache`	True/false, by default it is false. This causes a current time in milliseconds parameter added after url.
`ifModified`	True/false, by default it is undefined, passed to jQuery AJAX method as it is.
`versionUrl`	True/False, by default it is false. To cache JSON objects by versioning URL. Appends a version query parameter if set to true.

In following example, you can see how we can bind name query string parameter and that gets sent to server in query string.

```
<script type="text/javascript">
    ({ searchText: '' })
</script>
<div>
   <input type="search" placeholder="Search For" atom-value="$[scope.searchText]"/>
</div>
<div
    atom-type="AtomItemsControl"
    atom-items="[ AtomPromise.json('/db/products', { name: $scope.searchText }) ]">
```

17.2.5 `AtomPromise.get(url,query,options)` Method

Same as `json` method but this performs GET method and returns a string value in promise completion.

17.2.6 `AtomPromise.cachedJson(url,query,options)` Method

Same as AtomPromise.json, but result is cached for given `url`. Also note that this method caches results only on the basis of `url`, not query. If you want your query string to be cached, then you should call Atom.url before passing `url` to this method. Results of this promise are cached in `sessionStorage` of the browser as well as on current page. This reduces parsing time on each fetch as if you load list of `config` values for multiple controls from same source, only one result is bound to both controls.

17.2.7 `AtomPromise.configLabel(url,value,options)` Method

As we play more with label-value pair data pattern. We mostly store value in resultant database field, however to display the selection in User Interface, we need actual label. This is a handy method which loads array of label-pair and caches them and it retrieves label for corresponding value that we need.

In the following example, we are loading countries JSON Array from given url, and value is retrieved, this method will return label ‘United States’ for corresponding value of ‘US’ when promise completes.

```
AtomPromise.configLabel('/db/countries','US',{ valuePath: 'CountryCode'});
```

17.3 `AtomQuery` Module

`AtomQuery` Module provides JSON Query capabilities for given array. First we will learn JSON Query Language which is designed specifically to write queries in Object Literal form along with binding.

All string operations are Case Insensitive except they are marked with “CS”.

Following Grammar represents JSON Query;

```
query  : '{' queryExp ( ',' queryExp )*  '}'  ;
queryExp :    "'" propertyName ( (':'|':!') operator )? "'" ':' value
       |      '$or'  ':' query
       |      '$and' ':' query
       |      '$not' ':' query   ;
operator :    '=='
       |      '>='
       |      '<='
       |      'any'
       |      'in'
       |      'startsWith'  |  'startsWithCS'
       |      'endsWith'    |  'endsWithCS'
       |      'contains'    |  'containsCS'   ;
```

17.3.1 JSON Data Store

Let’s consider, we have following data available, also note this data is hierarchical and we want to query nested array as well.

```
var items = [
    {
        OrderID: 1,
        Customer: {
            CustomerID: 1, Name: 'Akash Kava',
        },
        Items: [
            { ProductID: 3, ProductName: 'Star Wars DVD' },
            { ProductID: 4, ProductName: 'iPhone'}
        ]
    },
    {
        OrderID: 2,
        Customer: {
            CustomerID: 2, Name: 'John Carter'
        },
        Items: [
            { ProductID: 7, ProductName: 'Universe DVD' },
            { ProductID: 4, ProductName: 'iPhone'}
        ]
    },
    {
        OrderID: 3,
        Customer: {
            CustomerID: 3, Name: 'Jessica Carter'
        },
        Items: [
            { ProductID: 7, ProductName: 'Universe DVD' },
            { ProductID: 9, ProductName: 'Samsung Phone' }
        ]
    }
];
```

17.3.2 Simple Comparison Query

To retrieve object or array of object, you can simply pass object literal with value and property.

```
// retreives order with OrderID 2
var order = Atom.query(items).firstOrDefault({ OrderID: 2 });

// retrieves an Enumerator (which you can enumerate or convert to array)
// for all orders with OrderID > 2
var orders = Atom.query(items).where({ 'OrderID:>=': 2 }).toArray();
```

To compare with any other JavaScript compare operator, the key of Object must contain a `':'` (colon) and the operator as shown in above example. Also in above example, where method returns `AtomEnumerator`, which you can convert to Array.

17.3.3 Nested Property Comparison

```
// retreives Order whose's Customer.CustomerID is 3
var order = Atom.query(items).firstOrDefault({ 'Customer.CustomerID': 3 });

// retreives Orders whose's Customer.CustomerID > 2
var orders = Atom.query(items).where({ 'Customer.CustomerID:>=': 2 }).toArray();
```

17.3.4 Composite Query

By default, multiple property conditions are combined with “AND” Boolean operator.

```
// retrieves Orders for multiple conditions
// both conditions are combined with AND boolean operator by default
var order = Atom.query(items)
    .firstOrDefault({ 'OrderID:>=': 2, 'Customer.CustomerID:>=': 2 });

// retrieves Orders for multiple conditions
// both conditions are combined with OR operator
// $or contains a nested condition block
var order = Atom.query(items)
    .firstOrDefault({ '$or': { 'OrderID:>=': 2, 'Customer.CustomerID:>=': 2 } });
```

17.3.5 In Query

In operator is special operator, where you can specify an array of acceptable values. This is same as “IN” operator found in SQL QUERY.

```
// retrieves Order with given OrderIDs in array
var orders = Atom.query(items)
    .where({ 'OrderID:in': [2, 3, 5, 6] }).toArray();

// in also works for strings
var orders = Atom.query(items)
    .where({ 'Customer.Name:in': ['Akash Kava','John Carter'] }).toArray();
```

17.3.6 Any Query

Any query comes from C# LINQ, it basically evaluates given condition on property that is of type Array. For example, each Order contains multiple Items, and we want to search based on criteria for Items.

```
// retrieves all orders which contains Items with given ProductID
// any has a nested condition which is queried on Items array
var orders = Atom.query(items)
    .where({ 'Items:any': { ProductID: 2 } }).toArray();

// retrieves all orders which contains Items with given ProductIDs
// any has a nested condition which is queried on Items array
var orders = Atom.query(items)
    .where({ 'Items:any': { 'ProductID:in': [2, 3, 4] } }).toArray();
```

17.3.7 Useful Operators

After `':'` colon, you can apply any JavaScript compare operator. However, there are special operators we have added for you to easily do string operations which are as follow.

- `contains`
- `startsWith`
- `endsWith`
- `containsCS`
- `startsWithCS`
- `endsWithCS`

All “CS” methods are “Case Sensitive”.

17.4 `AtomQuery` Extension Methods

17.4.1 `sum` Method

Sum method will add up all the fields specified in the parameter along with the sum method call.

```
var total = Atom.query(orders)
    .where( { OrderStatus:'completed' } )
    .sum('total');
```

17.4.2 `count` Method

Count method will count number of items in given query, if the parameter is passed, parameter will be used to filter the query further and it will return the count.

```
// both the queries will return same results
total = Atom.query(orders)
    .where( { OrderStatus:'completed' } )
    .count();

total = Atom.query(orders)
    .count( { OrderStatus:'completed' } );
```

17.4.3 `any` Method

Any method will return true if there are any items in the query. If the parameter is passed, parameter will be used to filter the query further and it will return true if items exist after filtering.

```
// both the queries will return same results
any = Atom.query(orders)
    .where( { OrderStatus:'completed' } )
    .any();

any = Atom.query(orders)
    .any( { OrderStatus:'completed' } );
```

17.5 `AtomBinder` Module

This class is only provided as reference, you must avoid using this class and must use `Atom.set/get/add/insert/remove` methods instead. `AtomBinder` is core of Web Atoms, these methods actually evaluate properties based on availability of getter/setter definitions.

17.5.1 `AtomBinder.setValue(target,property,value)` Method

This method sets given value as property of the target. However, this method first calls `AtomBinder.getValue` to check if the value is same or not, in order to avoid recursive `setValue` calls. This method first checks if "`set_{property}`" method is available or not, if available it calls "`set_{property}`" method or it just assigns value to the property. This method calls `AtomBinder.refreshValue` if the value was modified.

17.5.2 `AtomBinder.getValue(target,property)` Method

This method retrieves property of the target. This method first checks if "`get_{property}`" method is available, if available it calls "`get_{property}`" method, otherwise it returns the member with property name.

17.5.3 `AtomBinder.refreshValue(target,property)` Method

This method refreshes all watchers (UI bindings) associated with given target, but only for specific property.

17.5.4 `AtomBinder.add_WatchHandler(target,property,handler)` Method

You can add Watch Handler to watch the given target, however you will only receive notifications for specified properties only.

17.5.5 `AtomBinder.remove_WatchHandler(target,property,handler)` Method

You can use this method to remove your Watch Handler for specified property to stop receiving property change notifications.

17.6 `AtomDate` Class

To display Dates, (Web Atoms automatically parses MVC Date formats into JavaScript Dates), you need following helper methods in Binding expressions to display date correctly.

17.6.1 `AtomDate.toMMDDYY` Method

Formats given JavaScript date object into MM/DD/YYYY format, this is for compatibility with old date formats.

17.6.2 `AtomDate.toShortDateString` Method

This method formats given date in “Jan 13 2014” format. If given date is string, it will parse and format the date.

17.6.3 `AtomDate.toDateTimeString` Method

This method formats given date object in Date and Time format. Date format is same as the one explained in previous example.

17.6.4 `AtomDate.parse` Method

This method tries to parse given date string into JavaScript date format, this parser does consider ASP.NET MVC Date format.

17.7 `AtomFileSize` Class

17.7.1 `AtomFileSize.toFileSize` Method

This formats given file size numeric value into human readable file size value such as kb, mb, gb etc.

18 WALKTHROUGHS

18.1 Post changes to server on AtomComboBox

`AtomComboBox` can only be used with SELECT element and it has special functionality to post data to HTTP server when any selection has been changed.

```
<script type="text/javascript">
  ({
      ratingList: [
        { label: "Save Rating As", value: -1 },
        { label: "3 Star Rating", value: 1 },
        { label: "2 Star Rating", value: 2 },
        { label: "1 Star Rating", value: 3 },
        { label: "No Rating", value: 0 },
      ],
      ratingSaved: function (scope, sender) {
          // combo box has selected item and
          // selected item has value property
          // that is available as value of combo box
          var value = Atom.get(sender, "value");
          // special case when selectedIndex is zero,
          if (value == -1)
              return;
          var items = Atom.get(scope, "itemList.selectedItems");
          var ae = new AtomEnumerator(items);
          while (ae.next()) {
              var item = ae.current();
              // this will force current Ratings
              // of all items to update its UI
              Atom.set(item, "Ratings", value);
          }
      }
  })
</script>
<select atom-name="Choice"
    atom-type="AtomComboBox"
    atom-post-url="/App/Entity/Products?command=ModifyRating"
    atom-post-data="[ { ids: $scope.itemList.value, rating:$owner.value } ]"
    atom-next="{$scope.ratingSaved}"
    atom-items="{ratingList}">
</select>
```

Let's assume, you have number of items displayed in the list and you want to change rating of selected items from 1 to 3 or nothing.

1. We create an array in scope, which displays 3 ratings, first prompt and last item to reset the ratings.
2. First item is called prompt, that is what is displayed to user by default. The reason this is the first item, instead of attribute prompt, is that we will reset combo box back to first item once the operation is over. Since serializing scope for prompt attribute can be complicated.
3. We create a method in scope, named `ratingSaved`, which will be called after the POST operation was successful.

4. In this method, we get access to selected value, “sender” is `AtomComboBox` itself, as it was initiated by user click operation.
5. We check if the value is zero, in this case, it is first item, then we return the method.
6. We then get selected Items property of `itemList` and set each item’s rating, which will update its UI, without refreshing and reloading all items from server.
7. Finally we will set post data property of `AtomComboBox` to set of properties that we want to POST to server.
8. And we will set a post url, in this case, post url must be set, empty post url will just act as normal combo box, it will not post any changes to same page.
9. We will set reference to `ratingSaved` as next action set for AtomComboBox, which will be called after the POST operation was successful.

18.2 Filtering `AtomPromise.json` from server

While we design screen to load collection of items from server, we often require sending filtering criteria to server. To make things easier and to make AJAX calls filterable with bindable parameters, we have designed AtomPromise.json in such a way that it accepts object literal and formats query string based on properties of object literal.

```
<script type="text/javascript">
    ({
        searchText: '',
        orderBy: 'CustomerName'
    })
</script>
<div atom-dock="Top" style="height:30px">
    <input type="search" atom-value="$[scope.searchText]"/>
    <!-- A Dummy Button for User, Since binding will only update
        if User moves focus out from text box or press enter,
        so we create a dummy button, which user will click,
        that will result in losing focus from text box and thus
        updating binding and that will reload the list-->
    <button>Search</button>
</div>
<div
    atom-type="AtomListBox"
    atom-items="[ AtomPromise.json( '/db/customers/query' , { name: $scope.searchText,
orderBy: $scope.orderBy } ) ]">
    <table>
        <thead>
            <tr>
                <th
                    atom-type="AtomSortableColumn"
                    atom-label="ID"
                    atom-sort-field="CustomerID"
                    atom-value="$[scope.orderBy]"></th>
                <th
                    atom-type="AtomSortableColumn"
                    atom-label="Name"
                    atom-sort-field="CustomerName"
                    atom-value="$[scope.orderBy]"></th>
                <th
                    atom-type="AtomSortableColumn"
                    atom-label="Email"
                    atom-sort-field="CustomerEmail"
                    atom-value="$[scope.orderBy]"></th>
            </tr>
        </thead>
        <tbody atom-presenter="itemsPresenter">
            <tr atom-template="itemTemplate">
                <td atom-text="{$data.CustomerID}"></td>
                <td atom-text="{$data.CustomerName}"></td>
                <td atom-text="{$data.CustomerEmail}"></td>
            </tr>
        </tbody>
    </table>
</div>
```

1. First Initialize variables in a Scope Script
2. `searchText` to empty text
3. `orderBy` to '`CustomerName`'
4. Since binding ignores '`undefined`' values, not initializing scope variables will result in no binding.
5. Now let's create a text box and bind it to `$scope.searchText`.
6. Since binding only refreshes input controls on change event or when you press enter, we will need some button for user to click. So we will add a button. This button does not do anything, but user just feels that user should click somewhere in order to search, user will click on button, that will result in change event fired and that will actually trigger search.
7. To create search while you are typing, you can bind events in two way binding for `keyup` event.
8. Now inside `AtomListBox` we will set `AtomPromise` for items property.
9. In this Promise, we have set two query string parameters and both are updated automatically and that causes refreshing of list as entire binding is one way binding for items property.
10. For each Table Column Header, we are attaching `AtomSortableColumn` which does two things.
    - It applies sorting indicators if its own `value` is same as that of its own `atom-sort-field`. As you can see, value is bound to a `$scope.orderBy` variable. It will change its css class based on value of current `$scope.orderBy`.
    - When you click the header, it changes `$scope.orderBy` to its own atom-soft-field, thus by making all sortable columns to refresh its css and reload items.

18.3 Filtering `AtomItemsControl` items locally

AtomItemsControl and all its derived controls supports local filtering. When designing an application, filtering is an important design constraint and we will explain how to filter data locally to reduce server round trips. This must be done carefully as it will impact performance, but we have successfully played with 100 to 1000 items for local filtering and it works well. However based on size of data and UI elements, you must test your performance on both mobile and desktop and choose which filtering method you want to apply.

```
<script type="text/javascript">
    ({
        searchText: '',
        orderBy: 'CustomerName',
        filterDelegate: function (q) {
            if (!(q && q.name))
                return null;
            return function (data) {
                return data.CustomerName.indexOf(q.name) == 0;
            }
        }
    })
</script>
<div atom-dock="Top" style="height:30px">
    <input type="search" atom-value="$[scope.searchText]"/>     <button>Search</button>
</div>
<div atom-type="AtomListBox"
    atom-items="[ AtomPromise.json( '/db/customers/query') ]"
    atom-sort-path="[$scope.orderBy]"
    atom-filter="[ $scope.filterDelegate({ name: $scope.searchText}) ]">
    <table>
        <thead>
            <tr>
                <th atom-type="AtomSortableColumn"
                    atom-label="ID"
                    atom-sort-field="CustomerID"
                    atom-value="$[scope.orderBy]"></th>
                <th atom-type="AtomSortableColumn"
                    atom-label="Name"
                    atom-sort-field="CustomerName"
                    atom-value="$[scope.orderBy]"></th>
                <th atom-type="AtomSortableColumn"
                    atom-label="Email"
                    atom-sort-field="CustomerEmail"
                    atom-value="$[scope.orderBy]"></th>
            </tr>
        </thead>
        <tbody atom-presenter="itemsPresenter">
            <tr atom-template="itemTemplate">
                <td atom-text="{$data.CustomerID}"></td>
                <td atom-text="{$data.CustomerName}"></td>
                <td atom-text="{$data.CustomerEmail}"></td>
            </tr>
        </tbody>
    </table>
</div>
```

This example is same as the previous, but this loads all the items and provides filtering and sorting locally.

1. Note that we have not send any parameters in AtomPromise.
2. To sort items, it is simple, you can set `atom-sort-path` property to `$scope.orderBy`. If you want to perform your own custom sorting, than can create a function in scope and bind to it as well.
3. To set filter, we will create a function named `filterDelegate` in the scope and we will call that function in binding expression.
4. `filterDelegate` function does not filter anything, but it returns a function that will be used for filtering the data items. `filterDelegate` stores filtering parameter as closure variable `q`.

Same example can be rewritten as following to use built-in `Atom.query`.

```
<script type="text/javascript">
    ({
        searchText: '',
        orderBy: 'CustomerName'
    })
</script>
<div atom-dock="Top" style="height:30px">
    <input type="search" atom-value="$[scope.searchText]"/>     <button>Search</button>
</div>
<div atom-type="AtomListBox"
    atom-items="[ AtomPromise.json( '/db/customers/query') ]"
    atom-sort-path="[$scope.orderBy]"
    atom-filter="[ { 'CustomerName:contains': ($scope.searchText || undefined) } ]">
    <table>
        <thead>
            <tr>
                <th atom-type="AtomSortableColumn"
                    atom-label="ID"
                    atom-sort-field="CustomerID"
                    atom-value="$[scope.orderBy]"></th>
                <th atom-type="AtomSortableColumn"
                    atom-label="Name"
                    atom-sort-field="CustomerName"
                    atom-value="$[scope.orderBy]"></th>
                <th atom-type="AtomSortableColumn"
                    atom-label="Email"
                    atom-sort-field="CustomerEmail"
                    atom-value="$[scope.orderBy]"></th>
            </tr>
        </thead>
        <tbody atom-presenter="itemsPresenter">
            <tr atom-template="itemTemplate">
                <td atom-text="{$data.CustomerID}"></td>
                <td atom-text="{$data.CustomerName}"></td>
                <td atom-text="{$data.CustomerEmail}"></td>
            </tr>
        </tbody>
    </table>
</div>
```

In this example, when we set criteria as undefined, the filter object is ignored. Otherwise, it uses filter object with `Atom.query` to filter items. Please see `AtomQuery` section for more details.