

Re-Architecture Feature and Tech Spec

TerraPages HotLocality/AddressHelper

Version: 1.0

Release State Draft

Prepared By: John Hudson
Last updated: 17 February 2009



[This page left intentionally blank]

TerraPages HotLocality/AddressHelper
Re-Architecture Feature and Tech Spec



Document Title	Re-Architecture Spec
Document Subject	TerraPages HotLocality/AddressHelper
Version Number	1.0
Status	Draft
Summary	This document is intended for all stakeholders with a technical background who is interested in the more technical low level design details of TerraPages HotLocality.
Keywords	HotLocality, AddressHelper, G-NAF
© Copyright 2008 . All Rights Reserved.	

Modification History

Revision Date	Author	Version Number	Summary of Changes
07/08/2008	John Hudson	1.00	Original draft for comment.
26/08/2008	Marty Kopka	1.01	Minor comments on the initial draft release.
27/08/2008	John Hudson	1.02	Edited from comments, added Testing procedures, cleaned up method list, added how to add a map, or add event listener for easy map integration.
28/08/2008	John Hudson	1.03	Added callback/event handlers Added directory setting of options for constructors.
10/09/2008	John Hudson	1.04	Updated some of the method, needed to change the defaults. Updated the prefix method to use '_' rather than '.' As jQuery does not like the '.' In an Element ID
11/09/2008	John Hudson	1.05	Removed some redundant methods
18/09/2008	John Hudson	1.06	Added more options and fixed references to 'methods' which are now options
15/01/2009	Scott Henderson	1.07	Added Doco for new option.



TABLE OF CONTENTS

1.	INTRODUCTION.....	5
1.1.	PURPOSE	5
1.2.	AUDIENCE.....	5
1.3.	SCOPE	5
1.4.	OVERVIEW AND BACKGROUND.....	6
2.	REQUIREMENTS	7
2.1.	PROPOSED CHANGES	7
2.2.	OVERVIEW	7
2.2.1.	<i>Extensive Customisations</i>	7
2.2.2.	<i>Not within a FORM Element</i>	7
2.2.3.	<i>HTML ID's/NAME's</i>	8
2.2.4.	<i>JavaScript Object Model</i>	8
2.2.5.	<i>HTML CLASS's for styling</i>	8
2.2.6.	<i>Callback / Event Handlers</i>	8
2.2.7.	<i>Simple Access to data</i>	8
2.2.8.	<i>Removal of Redundant Functionality</i>	8
2.2.9.	<i>Backwards Compatibility</i>	8
2.2.10.	<i>TerraAdmin Integration</i>	9
2.2.11.	<i>Exposure of API</i>	9
3.	TECHNICAL OVERVIEW	10
3.1.	CUSTOMISATION	10
3.1.1.	<i>Options</i>	10
3.1.2.	<i>Simple Example</i>	14
3.1.3.	<i>Advanced Example</i>	14
3.2.	HTML ID'S/NAME'S	15
3.2.1.	<i>Simple Example</i>	16
3.2.2.	<i>Advanced Example</i>	16
3.3.	CLASS STYLING.....	17
3.3.1.	<i>Simple Example</i>	17
3.3.2.	<i>Advanced Example</i>	17
3.4.	CALLBACK / EVENT HANDLERS	18
3.4.1.	<i>Example:</i>	18
3.5.	SIMPLE DATA ACCESS	19
3.5.1.	<i>Top Level data access methods</i>	19
3.5.2.	<i>Fields Specific data access methods</i>	Error! Bookmark not defined.
3.5.3.	<i>JSON Examples</i>	19
3.6.	REDUNDANT CODE REMOVAL	20
3.7.	TESTING.....	20
3.7.1.	<i>Example Test Page:</i>	21
3.8.	BACKEND CHANGES	22

1. Introduction

1.1. Purpose

This document is intended to give LISAsoft Sales, Developers and any other interested stakeholders a Functional and Technical overview of the re-architecture of the TerraPages products AddressHelper and HotLocality. Lower level technical details will be covered for developers only, Sales staff are encouraged to give views on marketing potential.

1.2. Audience

The intended audience for this document includes:

- Sales Staff
 - Project Manager
 - Sales Manager
- Technical Staff
 - Project Manager
 - Developers

1.3. Scope

This document is to outline the requirements intended for the re-architecture of TerraPages AddressHelper and HotLocality. The intended scope is to identify core requirements whilst highlighting key technical barriers and marketing focus.

“The simple use case” is used throughout this document, this is its meaning:

A customer whom wants an AddressHelper or HotLocality on their website, easily setup a copy & paste of JavaScript / HTML and the customer has the product.

1.4. Overview and Background

TerraPages HotLocality/AddressHelper is an application component that can be included in larger scale applications to verify that only valid Australian addresses are entered by users. TerraPages HotLocality functionality differs from existing AddressHelper functionality in that only a single context sensitive field is provided to obtain an address rather than a set of tokenised fields as in previous AddressHelper functionality.

Collecting valid addresses from consumers is crucial for many organisations and business processes, but assembling and maintaining a database of valid addresses is a challenging and expensive venture. To overcome these problems and provide a definitive national address file, PSMA developed G-NAF, a database containing all the government recognised addresses in Australia, along with their corresponding geographical positions.

TerraPages HotLocality/AddressHelper is intended to be an easy to use, web based address selection tool used in larger web based applications that is underpinned by the G-NAF dataset. TerraPages HotLocality/AddressHelper will be a valuable resource for organisations that require address validation, but cannot justify the investment of creating or maintaining their own address database.

The communication between the TerraPages HotLocality/AddressHelper web based solution and the G-NAF database for validation and geocoding is done via Asynchronous JavaScript and XML (AJAX) where after a JavaScript event is fired off, the application sends information entered by the user to the server which processes this data and sends the response back to the JavaScript via an XML like structure known as JSON (JavaScript Object Notation) and then onto the user. All this is done instantly in the background so the user can continue what they are doing without the need to wait for a server response. With regards to the TerraPages HotLocality/AddressHelper solution, this will be done once the user has finished typing in some data. The data entered into the text box will then be sent to the server which will process this data and query the database. The results from the database are then returned to the server, encapsulated in a Data Model, into a JSON object, back to the JavaScript and finally then passed to the browser to display the results.

A re-architecture of the base application is required to fixed missing requirements and future proof the application. Whiles re-architecting for a more configurable application, consideration will be added to make the simplest use case (that is of a user just wanting an address helper in a html page) available with default settings.

2. Requirements

2.1. Proposed Changes

- Extensive customisation of fields and formatting
- AddressHelper and HotLocality not within own HTML FORM field
- Correct usage of HTML ID's and NAMES
- Re-architecture of the HotLocality/AddressHelper JavaScript Object Model
- HTML class's for styling
- Callback / Event handlers
- Simple method for getting data from Objects
- Removal of redundant functionality
 - Popup functionality
 - Any reference to mapping (specifically images)
- Backwards Compatibility
- TerraAdmin JavaScript code-snippet Generator
- **Future Considerations**
 - Exposure of API from JavaScript tier

2.2. Overview

2.2.1. Extensive Customisations

The main point for re-architecting the product is to allow it to effortlessly integrate into other systems and the main complaint from developers was the product does not integrate easily. This must be a feature of the product, allowing each and every aspect to be customised.

Defaults must be assigned to as many fields as possible for the simple use case. This is to stop over complication where customisation becomes a chore and difficult for some to grasp the many features that may be changed.

A setup harness will be required (TerraAdmin is a possible candidate) in a subsequent release. This will create the JavaScript and HTML code snippets that a User will add to their system to integrate the AddressHelper/HotLocality.

2.2.2. Not within a FORM Element

To fully integrate HotLocality and AddressHelper into submit FORMs (ie: the customers) it must not by default be within its own FORM. Currently the JavaScript that runs both applications creates a table within a HTML FORM which can be submitted. The AH application must not be within its own FORM but rather be customisable so if the customer needs AddressHelper or HotLocality within their own FORM it is possible.

2.2.3. HTML ID's/NAME's

The JavaScript that currently creates the table and INPUT fields does so with field ID's hardcoded to "subject" and "propertyNumber" etc..., this works fine for the simple use case, but for advanced users who need to add the HotLocality or AddressHelper to an existing application where field names are set differently this is not the case.

2.2.4. JavaScript Object Model

The need for a generic Object Model has arisen from different strains of the application becoming too different in structure. This needs to be reined in and a generic Object Model created where both applications are using the same architecture and method structure. This is a '*back end*' change and will not affect any customer's but will future proof the application to further changes.

2.2.5. HTML CLASS's for styling

With all the customisations and changes to the architecture a new way to customise the style of the AddressHelper and HotLocality field's needs to be implemented. This will focus on using the HTML attribute CLASS. CLASS is used by CSS styling files to define the style of an ELEMENT. Another styling technique will be to use wrappers to wrap the input fields. This will ensure the specific stylisation of fields.

2.2.6. Callback / Event Handlers

Will be used by developers to listen for actions generated by HotLocality and AddressHelper. These actions can have functions assigned to them which will allow the browser to respond when addresses are available.

2.2.7. Simple Access to data

All client side JavaScript Objects will have simple access methods to get any data. JSON objects will be used as the standard medium giving the customer access in a standard format.

2.2.8. Removal of Redundant Functionality

There are several things that are not within the domain of this project. They have been added and do not fit with the requirements of the product. Namely they are:

- Retrieve map – this should be another product requirement OpenLayers/Google/Yahoo and be used once the data is available to the client (browser).
- Popup functionality will be removed – If the customer wants popup functionality this option will be available via different JavaScript snippet, not part of the main code-base.

The simple customer wants to integrate an address helper into their website, if a popup is wanted we show them how to create and charge them development time.

2.2.9. Backwards Compatibility

Older versions of AddressHelper and HotLocality will be maintained for backwards compatibility. We will support two previous major releases of HotLocality and AddressHelper only. After this the customer will need to update their compatibility or face having an incompatible version. This can be made easy with simple to read instructions on how to upgrade to the new version on each release. This type of agreement should be considered when contracts are created for customers.

2.2.10. TerraAdmin Integration

Will allow any customer access to:

- JavaScript Generators
 - The JavaScript Snippet generator will allow the customers to generate all the HTML and JavaScript code to drop into their HTML to get AddressHelper or HotLocality running.
 - This may even have a Wizard style how to, giving customers an easy setup process
- Documentation
 - Further detailed documentation for extra customisation
- Developer API's
 - Not in next revision.

2.2.11. Exposure of API

In a future release it is envisioned that an API may be needed to expose the backend system to developers, this will be taken into consideration for future proofing. This will allow developers to develop against a set of very fast TerraPages GNAF capable services.

This will not be in the next revision of this release.

3. Technical Overview

3.1. Customisation

3.1.1. Options

The JavaScript Objects (HotLocality and AddressHelper) will allow each application a multitude of settable options, below are some of the customisable options. A comprehensive list will be available with explanation closed to release date.

Application

- **servicePointURL** – set the location to where AJAX requests will be sent

Input Parameters	Examples	Default Value	In Previous Version
Valid URL	http://terrapages.net/addresshelper	NULL, Must be set	No

- **prefix** – set a prefix to the NAME of each INPUT field

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“HI1”, “234”, “truckStore” “mud_store”	NULL	No

- **styleclass** – set the CSS style class for the entire app, this will allow advanced users to wrap the apps in their own styling and keep the default field CSS names

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“blue_inputs”, “mudStore_green”	NULL	No

- **delay** – set the delay time for showing any suggestions

Input Parameters	Examples	Default Value	In Previous Version
Integer 0 - 10	1, 10	2	Yes, v1.0 as setDelay()

- **geocode** – set TRUE/FALSE if the result set should contain a Geocode of the address suggestions (if one is available)

Input Parameters	Examples	Default Value	In Previous Version
Boolean TRUE, FALSE	true, false	false	Yes, v1.0, this is slightly different where



			this allows access to the data rather than forcing its display.
--	--	--	---

- **longitudeElementID** – set the element where the Longitude will be displayed, default is null, if set will also override the geocode option to TRUE

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“long_field”	NULL	No

- **latitudeElementID** – set the element where the Latitude will be displayed, default is null, if set will also override the geocode method to TRUE

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“lat_field”	NULL	No

- **maxReturnValues** – set the maximum number of returned values from the server

Input Parameters	Examples	Default Value	In Previous Version
Integer 1 - 100	1, 100	10	Yes, v1.0 as setNumberAddressesReturned()

- **displayElementID** – set the field where the current closest match is displayed

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“mudStore_dist_box”	NULL	No

- **saintyCheck** – check if the application is responsive or if there is an error, checks:
 - the availability of the service backend
 - All settings are correctly set (defaults are set and set values correct)

Return Value	Examples	Default Value	In Previous Version
TRUE if all is OK FALSE otherwise	<pre>If (hotLocality.saintyCheck()) { hotLocality.create(); }</pre>	NA	No

- **AddressHelper({dictionary})** – Constructor for AddressHelper, sets all options using an array input

Return Value	Examples	Default Value	In Previous Version

TerraPages HotLocality/AddressHelper
Re-Architecture Feature and Tech Spec



new AddressHelper	AddressHelper({id:'ah', delay:5, maxReturnValues:10})	NA	No
-------------------	---	----	----

- **HotLocality({dictionary})** – Constructor for HotLocality, sets all options using an array input

Return Value	Examples	Default Value	In Previous Version
new HotLocality	HotLocality({id:'hl', delay:5, maxReturnValues:10})	NA	No

- **suggestionLoadingStyleClass** – Sets the style of the input field while the suggestions are loading

Input Value	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	mudStore_loading	NA	No

- **suggestionResultsStyleClass** – Sets the style of the input field suggestions

Input Value	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	mudStore_result	NA	No

- **suggestionOddStyleClass** – Sets the style suggestions odd parts

Input Value	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	mudStore_odd	NA	No

- **suggestionEvenStyleClass** – Sets the style suggestions even parts

Input Value	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	mudStore_even	NA	No



- **suggestionOverStyleClass** – Sets the style suggestions over (mouse) parts

Input Value	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	mudStore_over	NA	No

- **result** – Set a function to call once there is any results

Input Value	Examples	Default Value	In Previous Version
function	doMyBidding	NA	No

- **geocodeResult** – Set a function to call once there is any geocode results

Input Value	Examples	Default Value	In Previous Version
function	doMyBidding	NA	No

INPUT fields:

- **id** – set the name of the INPUT field

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“mudStore_1”	Field Names: “suburb” “unitNumber” “propertyNumber”	No

- **elementId** – set the HTML ELEMENT of type INPUT that will ‘host’ the HotLocality/AddressHelper field, this will be used where a web application has already setup the INPUT fields and does not want to replace any HTML INPUT’s. (experimental, may not work, hoping to get this working with **Events**)

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“mudStore_suburb”	NULL	No

- **styleClass** – set the style class for this INPUT field

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“dark_blue1”	NULL	No

- **wrapper** – set the DIV wrapper for the INPUT field, this will allow advanced users to wrap the apps in their own wrappers for styling

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“field_box”	NULL	No

- **style** – if the customer uses their own style then this can be overwritten it is encouraged that the setClass be used instead which changes the CSS class rather than direct manipulation of the style

Input Parameters	Examples	Default Value	In Previous Version
String – a-z String – A-Z String – 0 – 9 String – Special Chars (“_”)	“mudStore” “special_red_for_suburb”	NULL	No

INPUT field settings will overwrite the application level settings if they are set otherwise default or application level settings take precedence.

3.1.2. Simple Example

The following is a snippet of how a HotLocality Object will be setup within a HTML page:

```
<script type="text/javascript">
    var hotLocality = new HotLocality('hotlocality');

    window.onload=function(){
        if (hotLocality.sanityCheck()){
            hotLocality.create();
        }
    }
</script>

<div id="hotlocality" />
```

Which will output:

```
<input type="text" style="freeFormAddressField" id="freeFormAddressField"
name="freeFormAddressField" onkeydown="hotLocality.showSuggestions(this,
event)" autocomplete="off"/>
```

3.1.3. Advanced Example

The following is a snippet of how an AddressHelper Object will be setup within a HTML page and how advanced layout will be achieved:

```
<script type="text/javascript">
    var adHelper = new AddressHelper({
        id: 'addressHelper',
        wrapper: 'field_wrapper',
        displayFieldID: 'heading_field'});
    adHelper.get('suburb').set({id: 'theSuburb'});
    adHelper.get('suburb').set({styleClass: 'theSuburb_class'});

    //...etc all fields can be set

    window.onload=function(){
        if (adHelper.sanityCheck()){
            adHelper.create();
        }
    }
</script>

<input type="text" style="myCustomBlue" id="heading_field"
onkeydown="adHelper.showSuggestions(this, event)" autocomplete="off"/>

<div id="adHelper_unitNumber" />Unit Number&nbsp;
<div id="adHelper_propertyNumber" />Property Number<br />
<div id="adHelper_suburb" />Suburb<br />

</div>
```

Which will output:

```
<input type="text" style="myCustomBlue" id="heading_field"
onkeydown="adHelper.showSuggestions(this, event)" autocomplete="off"/>

<div class="field_wrapper">

    <input type="text" id="unitNumber"
onkeydown="adHelper.showSuggestions(this, event)"
autocomplete="off"/> Unit Number <input type="text" id="unitNumber"
onkeydown="adHelper.showSuggestions(this, event)"
autocomplete="off"/> Property Number

    <input type="text" id="suburb" name="theSuburb"
class="theSuburb_class" onkeydown="adHelper.showSuggestions(this,
event)" autocomplete="off"/> Suburb

</div>
```

3.2. HTML ID's/NAME's

As shown above the ID's will be prefixed (if set) by the prefix option. This will allow easy integration into websites and applications.

The current architecture has all fields hardcoded so "suburb", etc, which is the only name the suburb field can have. This means if there is another suburb field in the page there will be conflict. To alleviate this ID's should be used and override-able.

3.2.1. Simple Example

```
<script type="text/javascript">
    var hotLocality = new HotLocality({
        id: 'hotlocality',
        prefix: 'hll'
    });

    hotLocality.get('fields').freeFormAddress.set({id:
    'secondFFAddress'});

    window.onload=function(){
        if (hotLocality.sanityCheck()){
            hotLocality.create();
        }
    }
</script>

<div id="hl_hotlocality" />
```

Which will output:

```
<input type="text" style="freeFormAddressField" id="hll_secondFFAddress"
onkeydown="hotLocality.showSuggestions(this, event)" autocomplete="off"/>
```

3.2.2. Advanced Example

```
<script type="text/javascript">
    var adHelper = new AddressHelper({
        id: 'addressHelper',
        prefix: 'ki'
    });

    adHelper.get('fields').freeFormAddress.set({elementId:
    'ki_heading_field'});

    adHelper.get('fields').suburb.set({id: 'ki_suburb'});

    window.onload=function(){
        if (adHelper.sanityCheck()){
            adHelper.create();
        }
    }
</script>

<div id="kl_adhelper" />
```

Which will output:

```
<input type="text" id="ki_suburb" onkeydown="adHelper.showSuggestions(this,
event)" autocomplete="off"/>
```

```
<!--somewhere in the HTML is expected to be an input/div/span with an
id='ki_heading_field' something like the following, this will be updated to
the current closest match when the user starts typing.-->
```

```
<input type="text" id="ki_heading_field"
onkeydown="adhelper.showSuggestions(this, event)"
autocomplete="off"/>&nbsp;
```

3.3. Class styling

Currently style IDs are hard coded to 'suburb'... etc.. what we want is an easily over writable option to give each ELEMENT a style. By default each field will be given a simple style, then if needed a class can be assigned either to the wrapper or to individual ELEMENTS.

3.3.1. Simple Example

```
<script type="text/javascript">
    var hotLocality = new HotLocality({ id: 'hotlocality',
                                        prefix: 'h11'});

    hotLocality.get('fields').freeFormAddress.set({
        id: 'secondFFAddress',
        styleClass: 'secondFFAddress'
    });

    window.onload=function(){
        if (hotLocality.sanityCheck()){
            hotLocality.create();
        }
    }
</script>
```

```
<div id="h11_hotlocality" />
```

Which will output:

```
<input type="text" class="secondFFAddress" id="h11.freeFormAddressField"
name="secondFFAddress" onkeydown="hotLocality.showSuggestions(this, event)"
autocomplete="off"/>
```

3.3.2. Advanced Example

```
<script type="text/javascript">
    var adHelper = new AddressHelper({id: 'addressHelper',
        prefix: 'ki',
        wrapper: 'addressHelper_box',
        wrapperClass: 'doubleLine',
        displayElementId: 'ki_heading_field',
    });

    adHelper.get('fields').suburb.set({
        id: 'ki_suburb',
        wrapper: 'ki_suburb_box',
        wrapperClass: 'light_blue',
        style: 'bold'
    });

    adHelper.get('fields').streetNumber.set({id: 'ki_number'});
```

```
        //...etc all fields can be set

        window.onload=function(){
            if (adHelper.sanityCheck()){
                adHelper.create();
            }
        }
    }
</script>

<div id="ki_addressHelper" />
```

Which will output:

```
<div id="addressHelper_box" class="doubleLine">

    <input type="text" id="ki.propertyNumberField"
    onkeydown="adHelper.showSuggestions(this, event)"
    autocomplete="off"/><br />

    <div id="ki_suburb_box" class="light_blue">

        <input type="text" style="bold" id="ki.suburb"
        onkeydown="adHelper.showSuggestions(this, event)"
        autocomplete="off"/><br />

    </div>
</div>
```

3.4. Callback / Event Handlers

The following Events will be registerable for callback functions for both high level objects (HotLocality) and for lower level Objects (fields).

- result
- geocodeResult
- unsuccessfulGeocode

3.4.1. Example:

```
<script type="text/javascript">
    var adHelper = new AddressHelper({id:'addressHelper',
                                     geocode: true,
                                     });

    adHelper.get('fields').set({result: updateStateField });

    window.onload=function(){
        if (adHelper.sanityCheck()){
            adHelper.create();
        }
    }
</script>
```

```
<script type="text/javascript">
    var adHelper = new AddressHelper({id:'addressHelper',
                                     geocode: true, geocodeResult:
                                     success, unsuccessfulGeocode: failed,
                                     });

    function failed(){
        alert('Could not geocode address');
    }

    function success(){
        alert('Successfully geocoded address');
    }

</script>
```

3.5. Simple Data Access

Data access presented in a standard way will allow for further development of external APIs. Two levels of access will be presented.

3.5.1. Top Level data access methods

These methods will be used for other plugin applications such as OpenLayers or Google Maps where by adding the location information to the page customers can integrate the location data into a mapping application (OpenLayers/Google/Yahoo). Mapping functionality is not included in this project only the methods to access the data for mapping, this is where we charge to add the mapping applications to the customer's app also.

- `get('current');`
 - Gets the current closest match Address String
- `get('data')`
 - Gets the current list of address's (and Geocode if option is set) that is available in the JavaScript Objects, HotLocality or AddressHelper
- `get('longitude')`
 - Gets the current closest matched Addresses longitude
- `get('latitude')`
 - Gets the current closest matched Addresses latitude

3.5.2. JSON Examples

Each data has the following structure

current : String Object

"30 Currie Street Adelaide, SA, 5000"

data : JSON Array

```
[ {address:"30 Curries Street Adelaide, SA, 5000", longitude:138.62, latitude:-34.93},  
{address:"Curries Street Adelaide", longitude:138.34, latitude:-34.87} ]
```

longitude : String Object

"138.62"

latitude : String Object

"-34.87"

3.6. Redundant code removal

Removal of mapping code opens the doors for easy integration of any of the earlier mentioned services (OpenLayers/Google/Yahoo) simply by adding the following example JavaScript code to a customer's page:

```
<script type="text/javascript">  
    var adHelper = new AddressHelper({  
        id: 'addressHelper',  
        geocode: true,  
        geocodeResult: updateMap  
    });  
  
    window.onload=function(){  
        if (adHelper.sanityCheck()){  
            adHelper.create();  
            createMap();  
        }  
    }  
</script>  
  
<div id="addressHelper" />  
<div id="map" />
```

3.7. Testing

JsUnit from <http://jsunit.net/> will be used to test all methods from both Objects HotLocality and AddressHelper.

JsUnit is essentially a port of JUnit to JavaScript. It runs a server that, like the JUnit Test Runner, runs a set of online live tests on your JavaScript.

HTML Test pages are similar to JUnit Test Classes. Each Test page will contain JavaScript Test Functions where are equivalent to JUnit test methods.

A full suite of assertions are available in JsUnit:

- `assert([comment], booleanValue)`
- `assertTrue([comment], booleanValue)`
- `assertFalse([comment], booleanValue)`
- `assertEquals([comment], value1, value2)`

- `assertNotEquals([comment], value1, value2)`
- `assertNull([comment], value)`
- `assertNotNull([comment], value)`
- `assertUndefined([comment], value)`
- `assertNotUndefined([comment], value)`
- `assertNaN([comment], value)`
- `assertNotNaN([comment], value)`
- `fail(comment)`

All methods will be tested for:

- Upper bounds
- Middle bounds
- Outer bounds
- NULL bounds
- Correct bounds
- Expected output

3.7.1. Example Test Page:

```
<html>
<head>
  <title>Test Page for getGeocode(value)</title>
  <script language="javascript" src="jsUnitCore.js"></script>
  <script language="javascript" src="myJsScripts.js"></script>
  <script type="text/javascript">
    var adHelper = new AddressHelper('addressHelper');
    adHelper.setCSS('html/css/ki_industries.css');
    adHelper.setDelay(0);
    adHelper.setPrefix('ki');

    adHelper.setDisplayFieldID('ki_heading_field');
    adHelper.getSuburbField().setName('ki_suburb');

    window.onload=function(){
      if (adHelper.sanityCheck()){
        adHelper.create();
      }
    }
  </script>
</head>
<body>

<div id="k1_adhelper" />

<script language="javascript">
  function testWithValidResults() {
    adHelper.setPropertyNumber(30);
    adHelper.setSuburb('Adelaide');
    adHelper.setStreetName('Currie');
    adHelper.setStreetType('Street');
    adHelper.setState('SA');
    adHelper.setPostcode(5000);

    assertEquals("0 results", null, adHelper.getGeocodeList());
  }
</script>
</body>
</html>
```

```
adHelper.setGeocode(true);
adHelper.setPostcode(5000); //force refresh of current address

assertEquals("1 result", 1, adHelper.getGeocodeList().size());
}
</script>
</body>
</html>
```

3.8. Backend Changes

Any changes to the services particularly any backend change that will affect developers will be described on a wiki page available [here](#). This will be a progressive list which will be updated on a regular basis.