**Usage**

**Require ngStorage and Inject the Services**

angular.module('app', [

'ngStorage'

]).controller('Ctrl', function(

$scope,

$localStorage,

$sessionStorage

){});

**Read and Write |**[**Demo**](http://plnkr.co/edit/3vfRkvG7R9DgQxtWbGHz?p=preview)

Pass $localStorage (or $sessionStorage) by reference to a hook under $scope in plain ol' JavaScript:

$scope.$storage = $localStorage;

And use it like you-already-know:

<body ng-controller="Ctrl">

<button ng-click="$storage.counter = $storage.counter + 1">{{$storage.counter}}</button>

</body>

Optionally, specify default values using the $default() method:

$scope.$storage = $localStorage.$default({

counter: 42

});

With this setup, changes will be automatically sync'd between $scope.$storage, $localStorage, and localStorage - even across different browser tabs!

**Read and Write Alternative (Not Recommended) |**[**Demo**](http://plnkr.co/edit/9ZmkzRkYzS3iZkG8J5IK?p=preview)

If you're not fond of the presence of $scope.$storage, you can always use watchers:

$scope.counter = $localStorage.counter || 42;

$scope.$watch('counter', function() {

$localStorage.counter = $scope.counter;

});

$scope.$watch(function() {

return angular.toJson($localStorage);

}, function() {

$scope.counter = $localStorage.counter;

});

This, however, is not the way ngStorage is designed to be used with. As can be easily seen by comparing the demos, this approach is way more verbose, and may have potential performance implications as the values being watched quickly grow.

**Delete |**[**Demo**](http://plnkr.co/edit/o4w3VGqmp8opfrWzvsJy?p=preview)

Plain ol' JavaScript again, what else could you better expect?

// Both will do

delete $scope.$storage.counter;

delete $localStorage.counter;

This will delete the corresponding entry inside the Web Storage.

**Delete Everything |**[**Demo**](http://plnkr.co/edit/YiG28KTFdkeFXskolZqs?p=preview)

If you wish to clear the Storage in one go, use the $reset() method:

$localStorage.$reset();

Optionally, pass in an object you'd like the Storage to reset to:

$localStorage.$reset({

counter: 42

});

**Permitted Values |**[**Demo**](http://plnkr.co/edit/n0acYLdhk3AeZmPOGY9Z?p=preview)

You can store anything except those [not supported by JSON](http://www.json.org/js.html):

* Infinity, NaN - Will be replaced with null.
* undefined, Function - Will be removed.

**Usage from config phase**

To read and set values during the Angular config phase use the .get/.set functions provided by the provider.

var app = angular.module('app', ['ngStorage'])

.config(['$localStorageProvider',

function ($localStorageProvider) {

$localStorageProvider.get('MyKey');

$localStorageProvider.set('MyKey', { k: 'value' });

}]);

**Prefix**

To change the prefix used by ngStorage use the provider function setKeyPrefix during the config phase.

var app = angular.module('app', ['ngStorage'])

.config(['$localStorageProvider',

function ($localStorageProvider) {

$localStorageProvider.setKeyPrefix('NewPrefix');

}])

**Custom serialization**

To change how ngStorage serializes and deserializes values (uses JSON by default) you can use your own functions.

angular.module('app', ['ngStorage'])

.config(['$localStorageProvider',

function ($localStorageProvider) {

var mySerializer = function (value) {

// Do what you want with the value.

return value;

};

var myDeserializer = function (value) {

return value;

};

$localStorageProvider.setSerializer(mySerializer);

$localStorageProvider.setDeserializer(myDeserializer);

}];)

**Minification**

Just run $ npm install to install dependencies. Then run $ grunt for minification.

**Hints**

**Watch the watch**

ngStorage internally uses an Angular watch to monitor changes to the $storage/$localStorage objects. That means that a digest cycle is required to persist your new values into the browser local storage. Normally this is not a problem, but, for example, if you launch a new window after saving a value...

$scope.$storage.school = theSchool;

$log.debug("launching " + url);

var myWindow = $window.open("", "\_self");

myWindow.document.write(response.data);

the new values will not reliably be saved into the browser local storage. Allow a digest cycle to occur by using a zero-value $timeout as:

$scope.$storage.school = theSchool;

$log.debug("launching and saving the new value" + url);

$timeout(function(){

var myWindow = $window.open("", "\_self");

myWindow.document.write(response.data);

});

or better using $scope.$evalAsync as:

$scope.$storage.school = theSchool;

$log.debug("launching and saving the new value" + url);

$scope.$evalAsync(function(){

var myWindow = $window.open("", "\_self");

myWindow.document.write(response.data);

});

And your new values will be persisted correctly.