Configuring

The UI Router is configured in the module `config` function. This can be done two different ways.

Module `config` Option #1

```
var app = angular.module('app', ['ui.router'], function($stateProvider,
$urlRouterProvider) {

}
```

Module `config` Option #2

```
var app = angular.module('app', ['ui.router']);

app.config(function($stateProvider, $urlRouterProvider) {

});
```

Default Otherwise Route

Defines a path that is used when an invalid route is requested.

```
$urlRouterProvider.otherwise('/');
```

States

Basic State

```
$stateProvider
  .state('state-name', {
    url: 'some/url',
    templateUrl: 'path/to/template'
  });
```

Basic State with Controller

```
$stateProvider
  .state('state-name', {
    url: 'some/url',
    controller: 'ControllerName as vm',
    templateUrl: 'path/to/template'
  });
```

Child State

NOTE: the `url` here is *appended* to the parent's `url` property.

```
$stateProvider
  .state('parent-state.child-state', {
    url: 'some/url',
    templateUrl: 'path/to/template'
  });
```

Named View

```
$stateProvider
  .state('state-name', {
    url: 'some/url',
    views: {
      'view-name': {
        templateUrl: 'path/to/template'
      }
    }
  });
```

Named View with Controller

```
$stateProvider
  .state('state-name', {
    url: 'some/url',
    views: {
      'view-name': {
        controller: 'ControllerName as vm',
        templateUrl: 'path/to/template'
      }
    }
  });
```

Named View on Named State

```
$stateProvider
  .state('parent-state.child-state.grandchild-state', {
    url: '/enter-pin',
    views: {
      'view-name@state-name': {
        templateUrl: 'path/to/template'
      }
    }
  });
```

Resolvers

As far as what can be injected into a resolver, it's anything you'd be able to inject into something

like a controller **and** previous resolved values. The second example shows how you would inject the first value.

The `valueOne` key name is also what gets injected into the controller.

```
$stateProvider
  .state('state-name', {
    url: 'some/url',
    templateUrl: 'path/to/template',
    resolve: {
      valueOne: function($stateParams, $q) {

      },
      valueTwo: function($stateParams, $q, valueOne) {

      }
    }
  });
```

Components

```
$stateProvider
  .state('state-name', {
    url: 'some/url',
    component: 'component-name'
  });
```

Components with Resolve

```
$stateProvider
  .state('state-name', {
    url: 'some/url',
    templateUrl: 'path/to/template',
    resolve: {
      componentBinding: function() {
        // here you can resolve whatever that component binding needs
      }
    }
  });
```

Usage

Changing States Programmatically

```
function SomeController($state) {
  $state.go('state-name');
}
```

Changing States with Params

```
function SomeController($state) {
  $state.go('state-name', {key: 'value'});
}
```

Getting a State Object

```
function SomeController($state) {
  var stateObject = $state.get('state-name');
}
```

State Change Options

- `location`: If true will update the url in the location bar, if false will not. If string, must be "replace", which will update url and also replace last history record.
- `inherit`: If true will inherit url parameters from current url.
- `relative`: When transitioning with relative path (e.g '^'), defines which state to be relative from. The `$stateObject` can be recovered with `$state.get`.
- `notify`: If true will broadcast $stateChangeStart and $stateChangeSuccess events.
- `reload`: If true will force transition even if no state or params have changed. It will reload the resolves and views of the current state and parent states. If reload is a string (or state object), the state object is fetched (by name, or object reference); and \ the transition reloads the resolves and views for that matched state, and all its children states.

The types and values of the options are shown below in the usage.

```
function SomeController($state) {
  $state.go('state-name', {
    location: true|false|string,
    inherit: true|false,
    relative: $stateObject,
    notify: true|false,
    reload: true|false
  });
}
```

Getting the Link for a State

```
function SomeController($state) {
  var href = $state.href('state-name', {params}, {options});
}
```

Reloading the Current State

```
function SomeController($state) {
```

```
  $state.reload();
}
```

Events

There are four very common events. All of them are hooked up on the `$rootScope`. Those events are:

- `$stateChangeError`: Fired when an error occurs during transition. It's important to note that if you have any errors in your resolve functions (javascript errors, non-existent services, etc) they will not throw traditionally. You must listen for this $stateChangeError event to catch ALL errors.
- `$stateChangeStart`: Fired when the state transition begins. You can use event.preventDefault() to prevent the transition from happening and then the transition promise will be rejected with a 'transition prevented' value.
- `$stateChangeSuccess`: Fired once the state transition is complete.
- `$stateNotFound`: Fired when a requested state cannot be found using the provided state name during transition. The event is broadcast allowing any handlers a single chance to deal with the error (usually by lazy-loading the unfound state). A special unfoundState object is passed to the listener handler, you can see its three properties in the example. You can use event.preventDefault() to abort the transition and the promise returned from go will be rejected with a 'transition aborted' value.

$stateChangeError

```
function SomeController($rootScope) {
  $rootScope.$on('$stateChangeError',
    function(evt, toState, toParams, fromState, fromParams, error) {

    });
}
```

$stateChangeStart

```
function SomeController($rootScope) {
  $rootScope.$on('$stateChangeStart',
    function(evt, toState, toParams, fromState, fromParams) {

    });
}
```

$stateChangeSuccess

```
function SomeController($rootScope) {
  $rootScope.$on('$stateChangeSuccess',
    function(evt, toState, toParams, fromState, fromParams) {

    });
```

```
}
```

$stateNotFound

The `unfoundState` object contains the `to`, `toParams`, and `options` properties.

```
function SomeController($rootScope) {
  $rootScope.$on('$stateNotFound',
    function(evt, unfoundState, fromState, fromParams) {

    });
}
```