

Developing in Service Portal

Bringing Bootstrap to life with AngularJS: Widgets, Widgets, Widgets!

Frank Schuster
Platform Architect

Introductions

- Name, Company, Role, Portal/CMS (Yes/No), Which category of the below do you fit in?

No-Code:

Configuration

Low-Code:

ServiceNow Administrator

HTML, CSS

Page and Portal creation

Pro-Code:

AngularJS (front-end)

ServiceNow API (back-end)

- What are you hoping to get out of this session today?

Agenda

- Widget Basics and Options
- **Lab 1:** Low-Code Custom Widget
- Data vs Input object & Data flow
- Common Angular directives and SP APIs
- The Art of Stealing: Knowing your resources
- **Lab 2:** Building a contact form (pure HTML)
- **Lab 3:** Building a table of Incidents (load data)
- Widget Advanced: Extended Options, Dependencies, Angular Providers and ng-Templates
- **(Optional) Lab 4:** Modifying our contact form - adding form validation, loading and writing data from/to the server and leveraging custom CSS
- Wrap up / Resources / Q&A
- **Running free:** Build your own widget, bring in 3rd-party libraries, team-up, discuss, ask, innovate 😊

Widget Basics and Options

What is this **Widget** thing?

- A Widget is *technically* an AngularJS directive
- It consists of 4 parts:
 - HTML Template (*Body or view*)
 - Client Script (Angular Controller, already linked 😊)
 - CSS (SASS)
 - Server Script (GlideRecord, GlideSystem, REST Messages etc.)
- Reusable in multiple (or the same....) pages & portals
 - A new widget instance is created each time
- Can make use of so called “Options” for specific configuration
- Widgets are *read-only*
 - If you need to make changes, it's best practice to clone it first to benefit from future upgrades
- Some components are only available in Platform view

Widget Editor (Anatomy)

- You can preview a widget directly in the Widget Editor
- Demo data can be applied via JSON
- Why reinvent the wheel? Clone FTW!

Widget

Hello World 3

Show


☒ HTML Template

☒ CSS - SCSS

☒ Client Script

☒ Server Script

☐ Link Function



Save (⌘ + s)



This widget is **read only** and cannot be edited except for the **Public** checkbox. You can clone this widget by clicking the  button in the top right corner of the editor

HTML Template	CSS - SCSS	Client Script	Server Script
<pre>1 <div> 2 Enter your name: 3 <input type="text" ng-model=" 4 <h1>{{ c.data.message }}</h1> 5 </div></pre>	<pre>1 h1 { 2 color: #428bca; 3 }</pre>	<pre>1 function() { 2 var c = this; 3 c.display = function() { 4 c.server.update().then(5 console.log("message", 6) 7 } 8 9 c.display(); 10 }</pre>	<pre>1 (function() { 2 if (input) { 3 data.message = (input.s 4 } 5 })();</pre>

Widget Options vs Instance Options

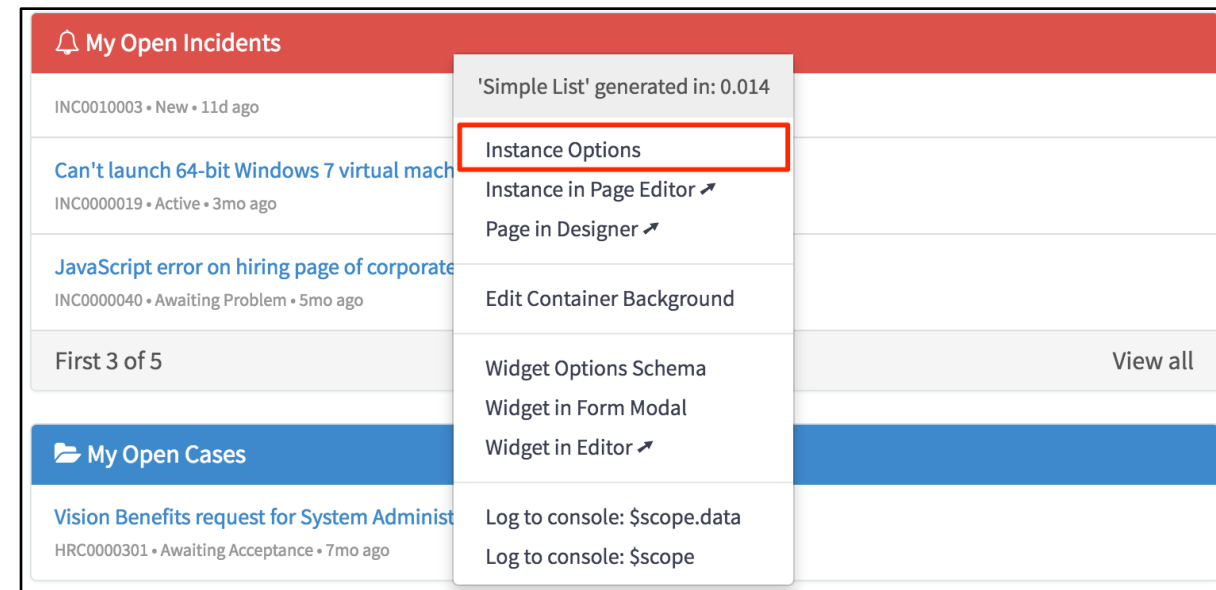
- Options define essentially what properties can be configured for the widget
- Configurable by editing the Widget's Option Schema
- Limited selection of Options field types
 - Reference Qualifiers do NOT apply to the Option Schema (as of now...)
 - No UI policies / Client scripts
 - Fear not! (advanced) method available 😊
- You can change options on demand directly on the page!
 - Ctrl + right click on widget instance
 - **sp_admin** roled users only, of course!

Widget Options Schema

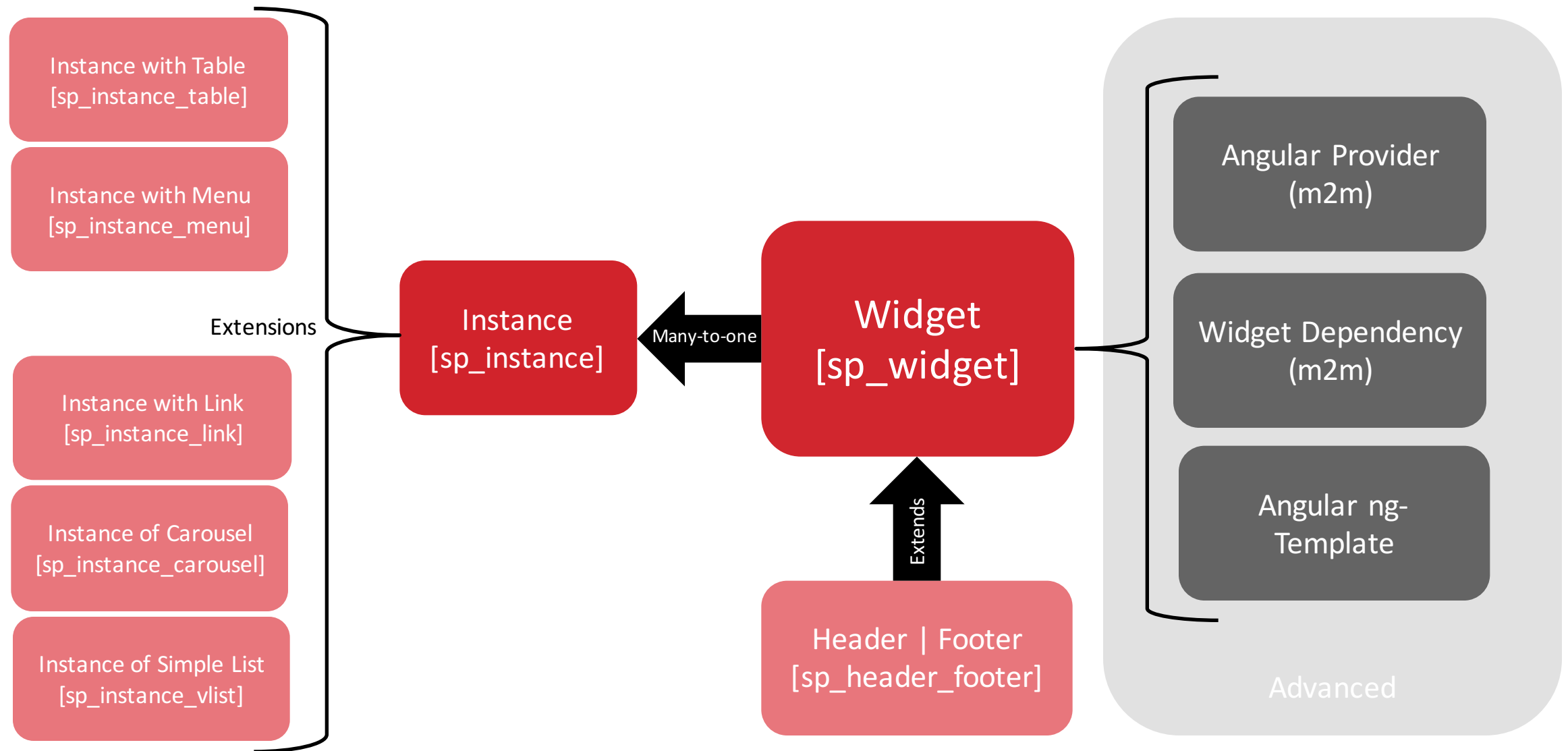
✕	My Option	my_option	string	▼	Hint (optional)
---	-----------	-----------	--------	---	-----------------

+ Add a new option with label, name, type and optional hint

Save (⌘ + s)



Widget Table(s) Structure



Portal 4 Ways

{yourInstance}.service-now.com/

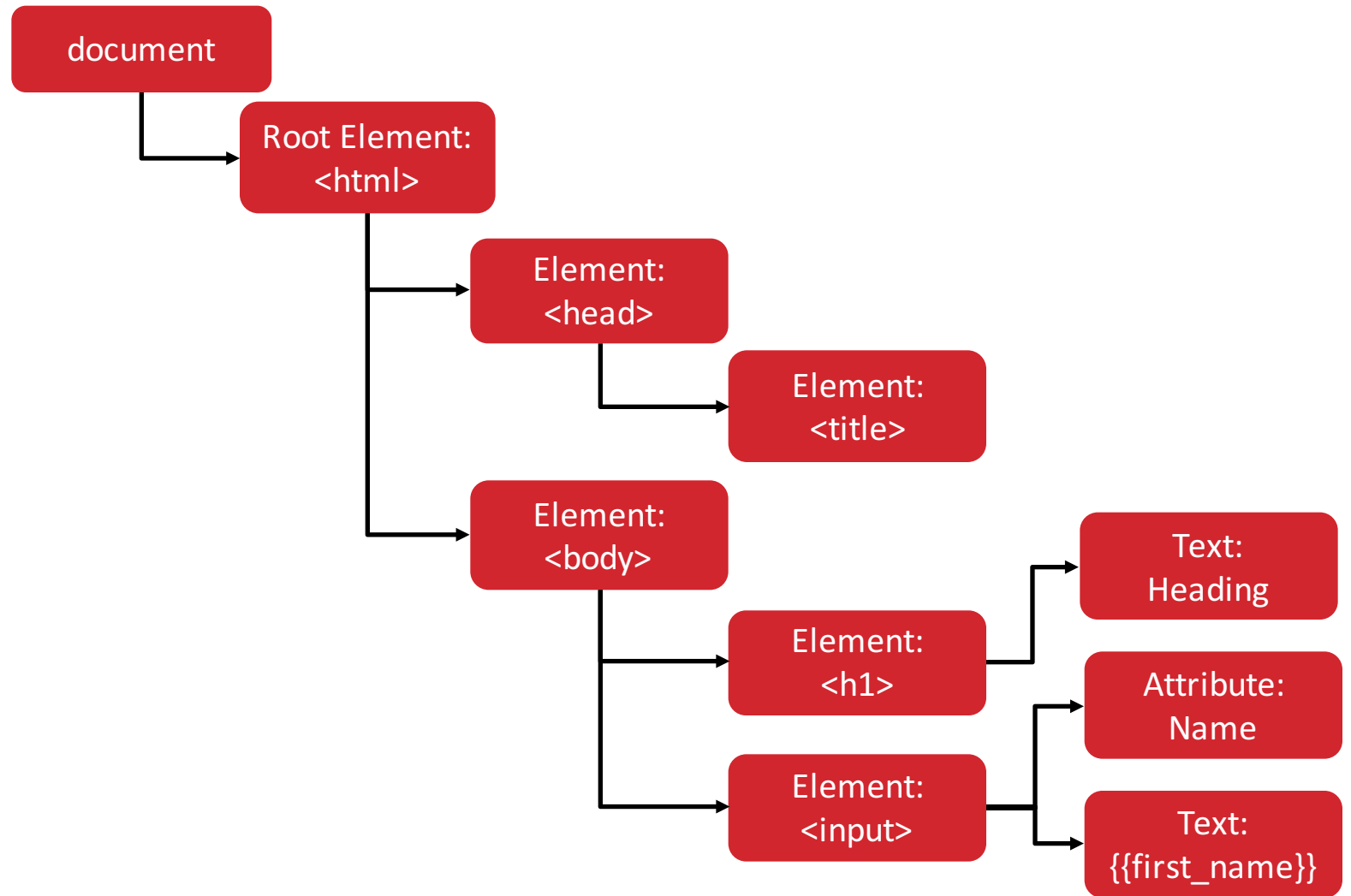
1. {URLSuffix}
 - With Theme
2. \$sp.do?id={pageID}
 - Without Theme
3. /nav_to.do?uri=/{URLSuffix}
 - Within Platform View and including Theme
4. /nav_to.do?uri=\$sp.do?id={pageID}
 - Within Platform View excluding Theme

Data vs Input object

- DOM & \$scope
- Data flow
- Common AngularJS Directives
- SP API Utilities

DOM – the Document Object Model

- The W3C (World Wide Web Consortium) **Document Object Model (DOM)** is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.

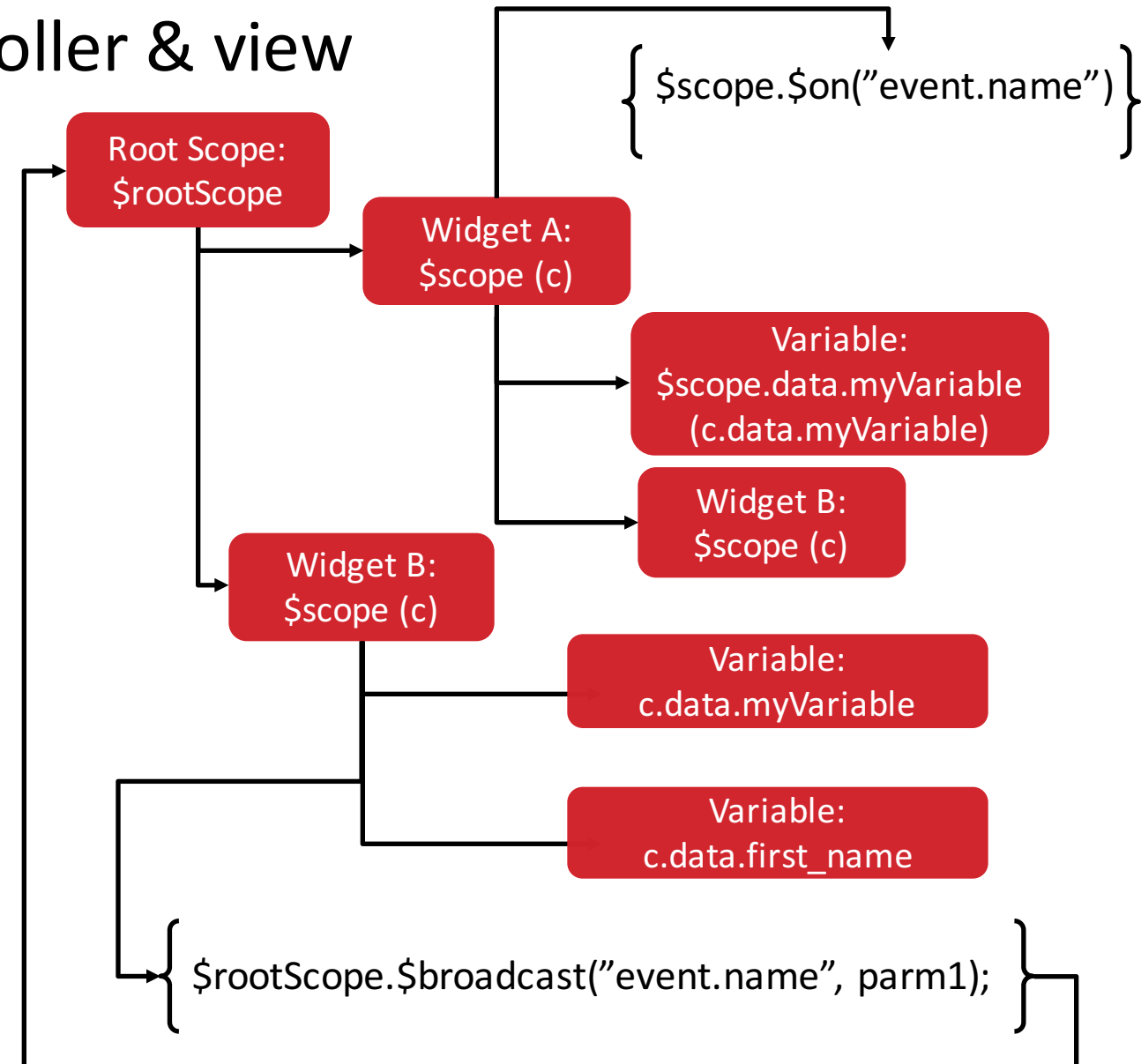


DOM – Tilt3D Model



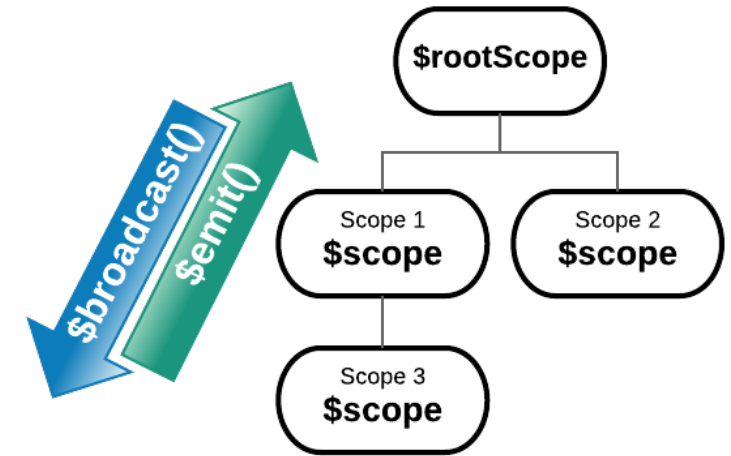
\$scope – the glue between controller & view

- Scope is a special Javascript object that refers to the application model and which plays the role of joining the controller with the views
- Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.
- Scopes provide context against which expressions are evaluated. For example `{{c.data.first_name}}` expression is meaningless, unless it is evaluated against a specific scope which defines the first name property.



Cross-Scope Communication

- Events can be announced and listened by any scope
- The announcement method determines, which scope can listen to it
- With **\$broadcast()**, the event will travel **down** in the scope tree from wherever it was announced, until it hits the bottom. The event will not be heard by sibling scopes. See more on the [official AngularJS documentation](#).
- Using **\$emit()**, the event will travel **up** in the scope tree from wherever it was announced, until it hits the \$rootScope. This event will not be heard by sibling scopes. See more on the [official AngularJS documentation](#).
- Listen to events by using **\$on()**



Lab 1 – Low-Code Custom Widget (20 minutes)

<https://servicenow.box.com/v/sp-dev-workshop-v2>

Instructions

- Download this 1 file only:
 - **Lab Guide:** Service Portal Lab Guide - Lab 1

Goals

- Cloning an existing widget
- Applying simple HTML/CSS changes
- Abstracting the widget functionality with configurable options

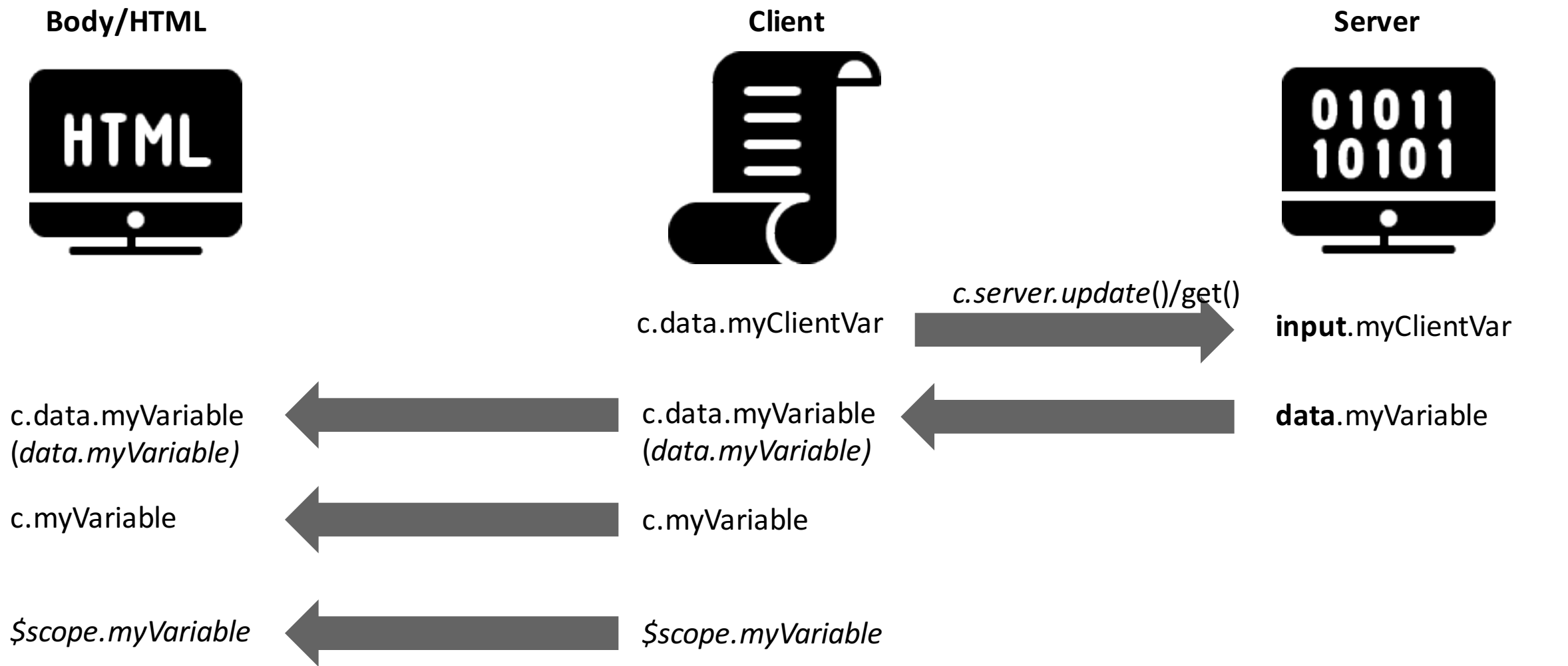
The mighty **data** object

- The data object facilitates any data exchange between HTML/Client & Server
- All widgets use the Angular “Controller As” notation per default
 - See the field “controllerAs” on a widget record – here you will find “c”
 - AngularJS is moving away from the \$scope notation and more towards the “Controller As” notation
 - You **do not** have to use the \$scope notation, be cautious when you e.g. look up tutorials or examples.
 - If you use \$scope you **will have to add \$scope** to the constructor of the client script, otherwise it will not work (*function (\$scope)*)
- To pass data from the client to the server simply write **c.server.update()** in the client script
 - If you want to provide immediate user feedback based on what happened on the server it might be a good idea to use a callback function (a so called **Promise**) – e.g. `c.server.update().then() { ... }`
- There’s also **c.server.get(Object)** where you can send a custom *input*. E.g. `c.server.update({symbol: c.data.symbol})`. **Note**: it also returns a Promise.

The **input** object

- Everything that is written into the **data (c.data)** object on the client is accessible on the server side after the **c.server.update()**, or **c.server.get()**, function has been processed
- **c.data.myVar** can then be accessed as **input.myVar** on server side
- **Important:** The server script is already being executed when a widget is loaded (that means already on page load!)
 - If you have code that should only be executed when specific input variables are present simply add an *if(input)* check to your server script. That will prevent code from running when the page loads.
 - *if(!input)* if you DO want to run on page load (e.g. pre-load options for a select HTML element)

How is data passed between the widget parts?



Common AngularJS directives

Directive	Purpose
ng-model	This directive binds an input, select, textarea (or custom form control) to the Angular Scope
ng-repeat	Iterate through a data array to e.g. build a list of records or a table
ng-show/ng-hide	Evaluate an expression to determine if an element is shown
<i>ng-if</i>	<i>Evaluate an expression to determine if an element is shown (removes the element from the DOM – might get costly)</i>
ng-class	Equivalent of the CSS element “class” – could e.g. be used to assign a CSS class conditionally
ng-change	Triggered when the according element changes (e.g. ng-change=“c.myChangeFunction()”)
ng-include	Fetches, compiles and includes an external HTML fragment (e.g. an Angular template)
watch	AngularJS record watcher – e.g. used to do something on the client when value changes on the server
rootScope	The root scope element will e.g. let you broadcast an event to the rootScope of the page (allows communication between widgets)

Service Portal API Utilities

API	Purpose
spUtil	Client side only utility. Can be used for example to add Info Messages to the page – also required for recordWatch functionality. See documentation for more functions available.
\$sp	Server side only utility. Can be used for example to get parameters from the URL, get the GlideRecord for the current Portal. See documentation for more functions available.
<i>console (Browser)</i>	<i>Powerful utility available in both client and server. Can be used to “log” debug messages of for example the data object.</i>

Custom Widget Showcase

- Some Widget Examples:
- Reporting, Process Flow Formatter, Extended System Status and more

The Art of Stealing

Knowing your Resources

Lab 2 – Building a bootstrap form (20 minutes)

<https://servicenow.box.com/v/sp-dev-workshop-v2>

Instructions

- Download this file only:
 - **Lab Guide:** Service Portal Lab Guide - Lab 2

Goals

- Creating a custom widget from scratch
- Build a form outside of ServiceNow with Bootstrap form builders
- Bring that form into your widget

Lab 3 – Building a Table of Incidents (45 minutes)

<https://servicenow.box.com/v/sp-dev-workshop-v2>

Instructions

- Download this file only:
 - **Lab Guide:** Service Portal Lab Guide - Lab 3

Goals


- Creating a custom widget from scratch
- Building a custom HTML table to display data
- Basic query of data from the server
- Applying common Angular directives
- Leveraging the SP recordWatcher functionality
- Apply CSS styles conditionally

Widget Advanced

- Extended Options
- Dependencies
- Angular Providers
- ng-Templates

Extended Options and the **options** object

- **options** object contains an instance of the defined widget schema options (duh...) E.g. `c.options.my_option (Client)` or `options.my_option (Server)`
- You don't necessarily need to "hardcode" all options to the Widget
 - You can specify options via a Data table (extending `sp_instance` table...) - How cool is that?
 - The fields from the table are automatically converted to Options and you can even select which ones are available for configuration 😊
 - Since you're dealing with a record, UI Policies and Client Scripts work too (Mobile/Both)
 - You can mix options defined in Schema too

* Data table	SP Report Instance [u_sp_report_instance]
Fields	<div></div> <div>Report, Report Type, Group By (Field), Show Title, Show Legend, Show Grid, Override Report Title (Service Portal only), Counts (Max. Records per Page), Override Styles, Margin (in px - shorthand), Padding (in px - shorthand), Border Radius (in px), Border Style (px and type - shorthand), Border Color</div>

Widget Dependencies

- Link 3rd party Angular JS and/or CSS libraries to your widget!
- Copy-paste the source into JS/CSS Includes OR embed via CDN/URL
- Angular module name is important!
 - Must match exactly the angular module name. I.e.
`angular.module('ngTable', []);`

The screenshot displays the ServiceNow widget configuration interface for a widget named 'ng-table'. The configuration is set for the 'Global' application. The 'Include on page load' checkbox is checked, and the 'Angular module name' is set to 'ngTable'. There are 'Update' and 'Delete' buttons. Below this, there are two sections: 'JS Includes' and 'CSS Includes'. Both sections have a 'Dependency = ng-table' filter. The 'JS Includes' section shows a table with one entry: '100 ng-Table' with a 'UI script' of 'ng-Table'. The 'CSS Includes' section shows a table with one entry: '100 ng-Table.css'. Both sections have pagination controls showing '1 to 1 of 1'.

Name: Application: ⓘ

Include on page load: ☒

Angular module name:

Go to: <<< 1 to 1 of 1 >>>

Dependency = ng-table

	Order ▲	JS Include	JS file URL	UI script
<input type="checkbox"/>	100	ng-Table		ng-Table

☐ Actions on selected rows... <> <<< 1 to 1 of 1 >>>

Go to: <<< 1 to 1 of 1 >>>

Dependency = ng-table

	Order ▲	CSS Include
<input type="checkbox"/>	100	ng-Table.css

Widget Angular Providers

- Pseudo “Script Includes”... but for Angular 😊
- Can define re-usable Directives, Factories or Services
 - **Directive:** Powerful tool for working with and modifying the DOM
 - **Factory vs Service:** Pretty much equivalent... Essentially, factories are functions that return the object, while services are constructor functions of the object. Capisce?
- Directives can be either element (E) or attribute (A)
- Factories or Services must be injected in controller constructor – e.g. `function($scope, myCoolestAwesomeService) { ...`
- Must be associated to the Widget (m2m related list)
- Providers can be dependent on other providers

Widget ng-Templates

- Define reusable HTML (with Angular/SCSS) markup for a widget
- Works with **ng-include** directive
- Handy for programmatically embedding templates on a page based on conditions
- ID must be unique
- Cloning a widget does **not** clone an ng-Template associated
- Only visible in Platform view

The screenshot shows the configuration interface for a widget named 'Form'. The 'Name' field is 'Form', the 'ID' is 'widget-form', and the 'Application' is 'Global'. The 'Body HTML template' field contains the following Angular template code:

```
<div class="panel-shift">
  <div class="" ng-if="!data.f._view.length && data.hideRelatedLists && data.emptyStateTemplate">
    <div class="empty-state-wrapper panel panel-default" ng-include="data.emptyStateTemplate"></div>
  </div>

  <div class="" ng-if="!data.f._view.length && data.hideRelatedLists && !data.emptyStateTemplate">
    <div class="panel panel-default">
      <div class="panel-heading"><span class="panel-title">{{data.f.title}}</span> <span ng-if="options.showFo
      <div class="panel-body wrapper-lg text-center">
        ${No elements to display}
      </div>
    </div>
  </div>
</div>
```

The `ng-include="data.emptyStateTemplate"` directive is highlighted with a red box in the original image.

Lab 4 – Creating a Contact Request Form (1 hour)

<https://servicenow.box.com/v/sp-dev-workshop-v2>

Instructions

- Download this file only:
 - **Lab Guide:** Service Portal Lab Guide - Lab 3

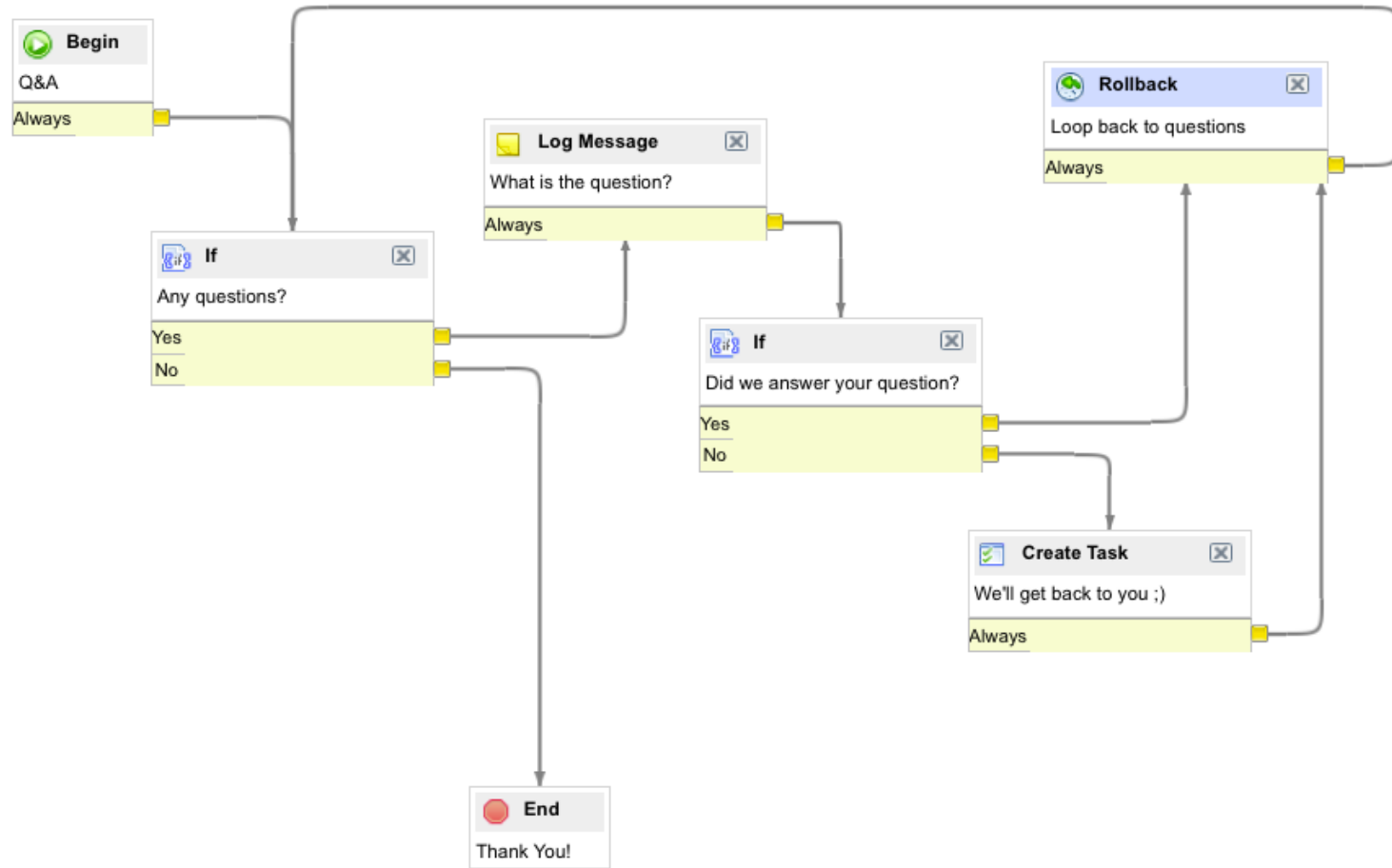
Goals

- Adding AngularJS to a bootstrap form
- Validating a form
- Utilize form data to insert and update records in tables
- Provide direct feedback to the user
- Getting to know some of the AngularJS directives

Development Resources

- API Documentation:
 - <https://github.com/service-portal/documentation>
 - <https://docs.angularjs.org/api>
- caniuse.com: Browser compatibility cross checks
- andrew.hedges.name/experiments/aspect_ratio: Calculate Image ratios for the perfect fit
- <http://tylermcginnis.com/angularjs-factory-vs-service-vs-provider>: Good read on Factory vs Service Provider
- udemy.com: Online Training Courses (\$\$ and free)
- <https://www.codeschool.com/courses/shaping-up-with-angular-js>
- stackoverflow.com (Developer Community)
- **Google** 😊

Q&A / Discussion



We value your feedback...

A short survey regarding this session will soon be available on westnow.ca, look for “My Surveys” in the menu bar.



Hackathon / Free-for-all Widget Creation



- Create a Widget of your own, anything goes! OR
Zero ideas? How about you convert a bootstrap snippet from here: <http://bootsnipp.com/>
- Feel free to continue with previous Lab if you didn't finish OR
- Explore OOB widgets and understand how they work 😊



Thank you