



Building better components: reusable and generic

Christian Menzinger
Product Developer
cmenzinger@appero.com
[@appero_com](https://www.appero.com)

“Please allow me to introduce myself”

Sympathy for the Devil - Rolling Stones



Christian Menzinger

Working for ISV partner appero

3,5 years Salesforce Developer

Munich Developer Group Co-Leader



<http://www.appero.com>

 @appero_com

 appero-com



 salesforce developers
Developer Group Leader - Munich, Germany



Objectives

What can you expect

Climbing up!

Leveraging the framework

Fundamental thinking for Awesome Admins and Users

Code for Developers



Example from a Developer's Life

Calling a server side action using Apex

How to? [Google!](#)



Example from a Developer's Life

Google Search Result

Lightning Components Developer Documentation

https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/controllers_server_actions_call.htm

Ever copy & paste this code?

Calling a Server-Side Action

Call a server-side controller action from a client-side controller. In the client-side controller, you set a callback, which is called after the server-side action is completed. A server-side action can return any object containing serializable JSON data.

A client-side controller is a JavaScript object in object-literal notation containing a map of name-value pairs.

Let's say that you want to trigger a server-call from a component. The following component contains a button that's wired to a client-side controller echo action. `simpleServerSideController` contains a method that returns a string passed in from the client-side controller.

```
1 <aura:component controller="SimpleServerSideController">
2   <aura:attribute name="firstName" type="String" default="world"/>
3   <lightning:button label="Call server" onclick="{!c.echo}"/>
4 </aura:component>
```

This client-side controller includes an echo action that executes a `serverEcho` method on a server-side controller.



Use unique names for client-side and server-side actions in a component. A JavaScript function (client-side action) with the same name as an Apex method (server-side action) can lead to hard-to-debug issues. In debug mode, the framework logs a browser console warning about the clashing client-side and server-side action names.

```
01  ({
02    "echo" : function(cmp) {
03      // Create a one-time use instance of the serverEcho action
04      // in the server-side controller
05      var action = cmp.get("c.serverEcho");
06      action.setParams({ firstName : cmp.get("v.firstName") });
07
08      // Create a callback that is executed after
09      // the server-side action returns
10      action.setCallback(this, function(response) {
11        var state = response.getState();
12        if (state === "SUCCESS") {
13          // Alert the user with the value returned
14          // from the server
15          alert("From server: " + responseReturnValue());
16
17          // You would typically fire a event here to trigger
18          // client-side notification that the server-side
19          // action is complete
20        }
21        else if (state === "INCOMPLETE") {
22          // do something
23        }
24        else if (state === "ERROR") {
25          var errors = response.getError();
26          if (errors) {
27            if (errors[0] && errors[0].message) {
28              console.log("Error message: " +
29                          errors[0].message);
29            }
30            else {
31              console.log("Unknown error");
32            }
33          }
34        });
35
36        // optionally set storables, abortable, background flag here
37
38        // A client-side action could cause multiple events,
39        // which could trigger other events and
40        // other server-side action calls.
41        // $A.enqueueAction adds the server-side action to the queue.
42        // $A.enqueueAction(action);
43      }
44    })
45 })
```

Is this productional code?

Still some work to do

```
01  ({
02    "echo" : function(cmp) {
03      // create a one-time use instance of the serverEcho action
04      // in the server-side controller
05      var action = cmp.get("c.serverEcho");
06      action.setParams({ firstName : cmp.get("v.firstName") });
07
08      // Create a callback that is executed after
09      // the server-side action returns
10      action.setCallback(this, function(response) {
11        var state = response.getState();
12        if (state === "SUCCESS") {
13          // Alert the user with the value returned
14          // from the server
15          alert("From server: " + response.getReturnValue());
16
17          // You would typically fire a event here to trigger
18          // client-side notification that the server-side
19          // action is complete
20        }
21        else if (state === "INCOMPLETE") {
22          // do something
23        }
24        else if (state === "ERROR") {
25          var errors = response.getError();
26          if (errors) {
27            if (errors[0] && errors[0].message) {
28              console.log("Error message: " +
29                          errors[0].message);
30            }
31          } else {
32            console.log("Unknown error");
33          }
34        }
35      });
36
37      // optionally set storable, abortable, background flag here
38
39      // A client-side action could cause multiple events,
40      // which could trigger other events and
41      // other server-side action calls.
42      // $A.enqueueAction adds the server-side action to the queue.
43      $A.enqueueAction(action);
44    }
45  })
```

Problem: “I did it my way”

My Way - Frank Sinatra

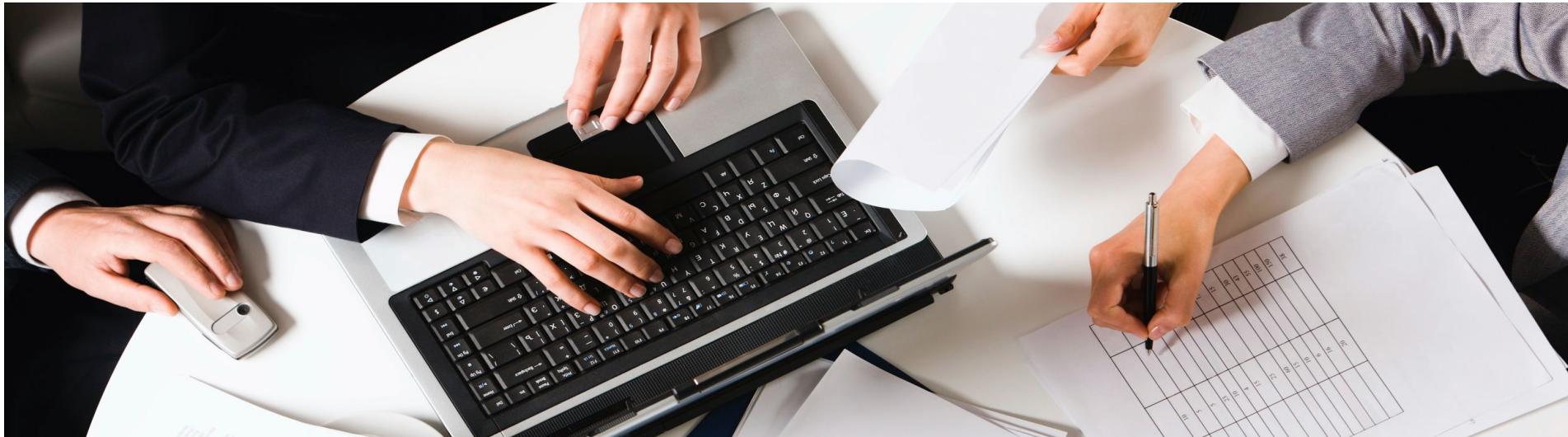
Several developers

Multiple Apex calls

Different code locations

Error-handling: console.log()? alert()? Toast messages? Modal component?

Have to re-Google template code after a while



Idea: “One ring to rule them all”

The Lord of the Rings - J.R.R. Tolkien

Need a consistent behavior

Need one code location

Don't want to fiddle around with the same code again and again!



Thinking about behavior

OOP: behavior = method

Prototyping: using a helper function

What changes in different scenarios?



Identify what changes

What are the essentials?

focus on:

- Apex function name
- calling parameters
- return values

```
01  ({
02    "echo" : function(cmp) {
03      // create a one-time use instance of the serverEcho action
04      // in the server-side controller
05      var action = cmp.get("c.serverEcho");
06      action.setParams({ firstName : cmp.get("v.firstName") });
07
08      // Create a callback that is executed after
09      // the server-side action returns
10      action.setCallback(this, function(response) {
11        var state = response.getState();
12        if (state === "SUCCESS") {
13          // Alert the user with the value returned
14          // from the server
15          alert("From server: " + response.getReturnValue());
16
17          // You would typically fire a event here to trigger
18          // client-side notification that the server-side
19          // action is complete
20        }
21        else if (state === "INCOMPLETE") {
22          // do something
23        }
24        else if (state === "ERROR") {
25          var errors = response.getError();
26          if (errors) {
27            if (errors[0] && errors[0].message) {
28              console.log("Error message: " +
29                          errors[0].message);
30            }
31          } else {
32            console.log("Unknown error");
33          }
34        }
35      });
36
37      // optionally set storable, abortable, background flag here
38
39      // A client-side action could cause multiple events,
40      // which could trigger other events and
41      // other server-side action calls.
42      // $A.enqueueAction adds the server-side action to the queue.
43      $A.enqueueAction(action);
44    }
45  })
```



Prototyping: “v1UsingHelperMethodHelper.js”

Only little changes from original code

```
1  ({
2    getContactList: function(component, filterString) {
3      this.callApex(component, "c.getContactsByFilter", {"filter": filterString}, this.getContactListSuccess);
4    },
5
6    callApex: function(component, controllerMethod, actionParameters, successCallback) {
7      // create a one-time use instance of the serverEcho action
8      // in the server-side controller
9      var action = component.get(controllerMethod);
10     action.setParams(actionParameters);
11
12     // Create a callback that is executed after
13     // the server-side action returns
14     action.setCallback(this, function(response) {
15       var state = response.getState();
16       if (state === "SUCCESS") {
17         //-->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE
18         successCallback(component, response.getReturnValue())
19       }
20     else if (state === "INCOMPLETE") {
```

Prototyping: “v1UsingHelperMethodHelper.js”

Only little changes from original code

```
31             console.log("Unknown error");
32         }
33     }
34 );
35
36 // optionally set storable, abortable, background flag here
37 // A client-side action could cause multiple events,
38 // which could trigger other events and
39 // other server-side action calls.
40
41 // $A.enqueueAction adds the server-side action to the queue.
42 $A.enqueueAction(action);
43 ,
44
45 ▼ getContactListSuccess: function(component,.returnValue) {
46     //process result in some way
47     component.set("v.contactList", returnValue);
48 }
49 })
```

How far do we get?

“callApex” can be reused for a second server call:

```
1  ({
2    getContactList: function(component, filterString) {
3      this.callApex(component, "c.getContactsByFilter", {"filter": filterString}, this.getContactListSuccess);
4    },
5
6    getOpportunityList: function(component, stageFilter) {
7      this.callApex(component, "c.getOpportunityByStage", {"stage": stageFilter}, this.getOpportunityListSuccess);
8    },
9
10   callApex: function(component, controllerMethod, actionParameters, successCallback) {
11     // create a one-time use instance of the serverEcho action
12     // in the server-side controller
13     var action = component.get(controllerMethod);
```

But:

Only possible within the **same** helper

One Apex controller per component

Still different code locations



Reusability

Encapsulate functionality

How can other components use this **abstraction** of an Apex service?

OOP: **class = extensible** program-code-template

Idea:

Create an **abstract base class** where other components can **extend** from!



Prototyping: “v2AbstractComponent.cmp”

Abstract base class

```
1 1 <aura:component extensible="true" abstract="true">
2
3 2
4
5
6
7
8
9
10
11
12 3 </aura:component>
```

Prototyping: “v2AbstractComponent.cmp”

Behavior

```
1  <aura:component extensible="true" abstract="true">
2
3      <aura:method name="callApex" action="{!c.onCallApex}">
4          <aura:attribute name="component" type="Aura.Component" />
5          <aura:attribute name="controllerMethod" type="String" />
6          <aura:attribute name="actionParameters" type="Object" />
7          <aura:attribute name="successCallback" type="Object" />
8      </aura:method>
9
10     { !v.body }
11
12 </aura:component>
```

Prototyping: “v2AbstractComponentController.js”

Controller invocation from method

```
1  ( {
2    onCallApex : function(component, event, helper) {
3      //get the method parameters
4      var params = event.getParams().arguments;
5      var controllerMethod = params.controllerMethod;
6      var actionParameters = params.actionParameters;
7      var successCallback = params.successCallback;
8      helper.callApex(component, controllerMethod, actionParameters, successCallback);
9    }
10 })
```

Prototyping: “v2AbstractComponentHelper.js”

Copy v1-Helper-JavaScript: callApex() did not change

```
1 ▼ ({  
2 ▼   callApex: function(component, controllerMethod, actionParameters, successCallback) {  
3     // create a one-time use instance of the serverEcho action  
4     // in the server-side controller  
5     var action = component.get(controllerMethod);  
6     action.setParams(actionParameters);  
7  
8     // Create a callback that is executed after  
9     // the server-side action returns  
10    action.setCallback(this, function(response) {  
11      var state = response.getState();  
12      if (state === "SUCCESS") {  
13        //-->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE  
14        successCallback(component, response.getReturnValue())  
15      }  
16      else if (state === "INCOMPLETE") {  
17        //-->HERE WE PROCESS THE RESPONSE RATHER CALL THE CALLBACK  
18        var incompleteResponse = response.getReturnValue();  
19        if (incompleteResponse.state === "INCOMPLETE") {  
20          incompleteResponse.setCallback(this, function(incompleteResponse) {  
21            if (incompleteResponse.state === "SUCCESS") {  
22              successCallback(component, incompleteResponse.returnValue);  
23            }  
24          });  
25        }  
26      }  
27    });  
28  }  
29});
```

Prototyping: “v2ConcreteComponent.cmp”

Extend the abstract component

```
1 ▼ <aura:component extends="c:v2AbstractComponent"
2     [...]
3 </aura:component>
```

Prototyping: “v2ConcreteComponentHelper.js”

Invoking the method

```
1 ▾ ({  
2 ▾   getContactList: function(component, filterString) {  
3     //call aura method from abstract base component:  
4     component.callApex(component, "c.getContactsByFilter", {"filter": filterString}, this.getContactListSuccess);  
5   },  
6  
7 ▾   getContactListSuccess: function(component, returnValue) {  
8     //process result in some way  
9     component.set("v.contactList", returnValue);  
10  }  
11 })
```

How far do we get?

“v2AbstractComponent” is **generic** - no hardcoded dependencies, all information is passed in

“v2AbstractComponent” is **reusable** - even on other projects



Problem:



Problem: One does not simply extend multiple classes

Inheritance is one to one

Sometimes already “used”: `extends="force:sls"`

OOP: “Favor composition over inheritance”!

Now: “My XYZ-component **IS** a service component”

Better: “My XYZ-component **HAS** a service component”

Use as many other components as you like - that’s how the framework is designed for!

Prototyping: “v3ServiceComponent.cmp”

Copy & modify V2-Component-markup

```
1  ▼ <aura:component extensible="true" abstract="true">
2
3    ▼   <aura:method name="callApex" action="{!c.onCallApex}">
4      <aura:attribute name="component" type="Aura.Component" />
5      <aura:attribute name="controllerMethod" type="String" />
6      <aura:attribute name="actionParameters" type="Object" />
7      <aura:attribute name="successCallback" type="Object" />
8    </aura:method>
9
10   {-v.body}
11
12 </aura:component>
```

Prototyping: “v3ServiceComponentController.js”

Copy & modify V2-Component-markup

```
1  ( {
2    onCallApex : function(component, event, helper) {
3      //get the method parameters
4      var params = event.getParams().arguments;
5      var callerComponent = params.component;
6      var controllerMethod = params.controllerMethod;
7      var actionParameters = params.actionParameters;
8      var successCallback = params.successCallback;
9      helper.callApex(callerComponent, controllerMethod, actionParameters, successCallback);
10    }
11  })
```

Prototyping: “v2AbstractComponentHelper.js”

Copy V2-Helper-JavaScript: callApex() did not change

```
1 ▼ ({  
2 ▼   callApex: function(component, controllerMethod, actionParameters, successCallback) {  
3     // create a one-time use instance of the serverEcho action  
4     // in the server-side controller  
5     var action = component.get(controllerMethod);  
6     action.setParams(actionParameters);  
7  
8     // Create a callback that is executed after  
9     // the server-side action returns  
10    action.setCallback(this, function(response) {  
11        var state = response.getState();  
12        if (state === "SUCCESS") {  
13            //-->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE  
14            successCallback(component, response.getReturnValue())  
15        }  
16        else if (state === "INCOMPLETE") {  
17            //-->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE  
18            successCallback(component, null)  
19        }  
20    })  
21 }  
22 }  
23 }  
24 }  
25 }  
26 }  
27 }  
28 }  
29 }  
30 }  
31 }  
32 }  
33 }  
34 }  
35 }  
36 }  
37 }  
38 }  
39 }  
40 }  
41 }  
42 }  
43 }  
44 }  
45 }  
46 }  
47 }  
48 }  
49 }  
50 }  
51 }  
52 }  
53 }  
54 }  
55 }  
56 }  
57 }  
58 }  
59 }  
60 }  
61 }  
62 }  
63 }  
64 }  
65 }  
66 }  
67 }  
68 }  
69 }  
70 }  
71 }  
72 }  
73 }  
74 }  
75 }  
76 }  
77 }  
78 }  
79 }  
80 }  
81 }  
82 }  
83 }  
84 }  
85 }  
86 }  
87 }  
88 }  
89 }  
90 }  
91 }  
92 }  
93 }  
94 }  
95 }  
96 }  
97 }  
98 }  
99 }  
100 }  
101 }  
102 }  
103 }  
104 }  
105 }  
106 }  
107 }  
108 }  
109 }  
110 }  
111 }  
112 }  
113 }  
114 }  
115 }  
116 }  
117 }  
118 }  
119 }  
120 }  
121 }  
122 }  
123 }  
124 }  
125 }  
126 }  
127 }  
128 }  
129 }  
130 }  
131 }  
132 }  
133 }  
134 }  
135 }  
136 }  
137 }  
138 }  
139 }  
140 }  
141 }  
142 }  
143 }  
144 }  
145 }  
146 }  
147 }  
148 }  
149 }  
150 }  
151 }  
152 }  
153 }  
154 }  
155 }  
156 }  
157 }  
158 }  
159 }  
160 }  
161 }  
162 }  
163 }  
164 }  
165 }  
166 }  
167 }  
168 }  
169 }  
170 }  
171 }  
172 }  
173 }  
174 }  
175 }  
176 }  
177 }  
178 }  
179 }  
180 }  
181 }  
182 }  
183 }  
184 }  
185 }  
186 }  
187 }  
188 }  
189 }  
190 }  
191 }  
192 }  
193 }  
194 }  
195 }  
196 }  
197 }  
198 }  
199 }  
200 }  
201 }  
202 }  
203 }  
204 }  
205 }  
206 }  
207 }  
208 }  
209 }  
210 }  
211 }  
212 }  
213 }  
214 }  
215 }  
216 }  
217 }  
218 }  
219 }  
220 }  
221 }  
222 }  
223 }  
224 }  
225 }  
226 }  
227 }  
228 }  
229 }  
230 }  
231 }  
232 }  
233 }  
234 }  
235 }  
236 }  
237 }  
238 }  
239 }  
240 }  
241 }  
242 }  
243 }  
244 }  
245 }  
246 }  
247 }  
248 }  
249 }  
250 }  
251 }  
252 }  
253 }  
254 }  
255 }  
256 }  
257 }  
258 }  
259 }  
260 }  
261 }  
262 }  
263 }  
264 }  
265 }  
266 }  
267 }  
268 }  
269 }  
270 }  
271 }  
272 }  
273 }  
274 }  
275 }  
276 }  
277 }  
278 }  
279 }  
280 }  
281 }  
282 }  
283 }  
284 }  
285 }  
286 }  
287 }  
288 }  
289 }  
290 }  
291 }  
292 }  
293 }  
294 }  
295 }  
296 }  
297 }  
298 }  
299 }  
300 }  
301 }  
302 }  
303 }  
304 }  
305 }  
306 }  
307 }  
308 }  
309 }  
310 }  
311 }  
312 }  
313 }  
314 }  
315 }  
316 }  
317 }  
318 }  
319 }  
320 }  
321 }  
322 }  
323 }  
324 }  
325 }  
326 }  
327 }  
328 }  
329 }  
330 }  
331 }  
332 }  
333 }  
334 }  
335 }  
336 }  
337 }  
338 }  
339 }  
340 }  
341 }  
342 }  
343 }  
344 }  
345 }  
346 }  
347 }  
348 }  
349 }  
350 }  
351 }  
352 }  
353 }  
354 }  
355 }  
356 }  
357 }  
358 }  
359 }  
360 }  
361 }  
362 }  
363 }  
364 }  
365 }  
366 }  
367 }  
368 }  
369 }  
370 }  
371 }  
372 }  
373 }  
374 }  
375 }  
376 }  
377 }  
378 }  
379 }  
380 }  
381 }  
382 }  
383 }  
384 }  
385 }  
386 }  
387 }  
388 }  
389 }  
390 }  
391 }  
392 }  
393 }  
394 }  
395 }  
396 }  
397 }  
398 }  
399 }  
400 }  
401 }  
402 }  
403 }  
404 }  
405 }  
406 }  
407 }  
408 }  
409 }  
410 }  
411 }  
412 }  
413 }  
414 }  
415 }  
416 }  
417 }  
418 }  
419 }  
420 }  
421 }  
422 }  
423 }  
424 }  
425 }  
426 }  
427 }  
428 }  
429 }  
430 }  
431 }  
432 }  
433 }  
434 }  
435 }  
436 }  
437 }  
438 }  
439 }  
440 }  
441 }  
442 }  
443 }  
444 }  
445 }  
446 }  
447 }  
448 }  
449 }  
450 }  
451 }  
452 }  
453 }  
454 }  
455 }  
456 }  
457 }  
458 }  
459 }  
460 }  
461 }  
462 }  
463 }  
464 }  
465 }  
466 }  
467 }  
468 }  
469 }  
470 }  
471 }  
472 }  
473 }  
474 }  
475 }  
476 }  
477 }  
478 }  
479 }  
480 }  
481 }  
482 }  
483 }  
484 }  
485 }  
486 }  
487 }  
488 }  
489 }  
490 }  
491 }  
492 }  
493 }  
494 }  
495 }  
496 }  
497 }  
498 }  
499 }  
500 }  
501 }  
502 }  
503 }  
504 }  
505 }  
506 }  
507 }  
508 }  
509 }  
510 }  
511 }  
512 }  
513 }  
514 }  
515 }  
516 }  
517 }  
518 }  
519 }  
520 }  
521 }  
522 }  
523 }  
524 }  
525 }  
526 }  
527 }  
528 }  
529 }  
530 }  
531 }  
532 }  
533 }  
534 }  
535 }  
536 }  
537 }  
538 }  
539 }  
540 }  
541 }  
542 }  
543 }  
544 }  
545 }  
546 }  
547 }  
548 }  
549 }  
550 }  
551 }  
552 }  
553 }  
554 }  
555 }  
556 }  
557 }  
558 }  
559 }  
560 }  
561 }  
562 }  
563 }  
564 }  
565 }  
566 }  
567 }  
568 }  
569 }  
570 }  
571 }  
572 }  
573 }  
574 }  
575 }  
576 }  
577 }  
578 }  
579 }  
580 }  
581 }  
582 }  
583 }  
584 }  
585 }  
586 }  
587 }  
588 }  
589 }  
590 }  
591 }  
592 }  
593 }  
594 }  
595 }  
596 }  
597 }  
598 }  
599 }  
600 }  
601 }  
602 }  
603 }  
604 }  
605 }  
606 }  
607 }  
608 }  
609 }  
610 }  
611 }  
612 }  
613 }  
614 }  
615 }  
616 }  
617 }  
618 }  
619 }  
620 }  
621 }  
622 }  
623 }  
624 }  
625 }  
626 }  
627 }  
628 }  
629 }  
630 }  
631 }  
632 }  
633 }  
634 }  
635 }  
636 }  
637 }  
638 }  
639 }  
640 }  
641 }  
642 }  
643 }  
644 }  
645 }  
646 }  
647 }  
648 }  
649 }  
650 }  
651 }  
652 }  
653 }  
654 }  
655 }  
656 }  
657 }  
658 }  
659 }  
660 }  
661 }  
662 }  
663 }  
664 }  
665 }  
666 }  
667 }  
668 }  
669 }  
670 }  
671 }  
672 }  
673 }  
674 }  
675 }  
676 }  
677 }  
678 }  
679 }  
680 }  
681 }  
682 }  
683 }  
684 }  
685 }  
686 }  
687 }  
688 }  
689 }  
690 }  
691 }  
692 }  
693 }  
694 }  
695 }  
696 }  
697 }  
698 }  
699 }  
700 }  
701 }  
702 }  
703 }  
704 }  
705 }  
706 }  
707 }  
708 }  
709 }  
710 }  
711 }  
712 }  
713 }  
714 }  
715 }  
716 }  
717 }  
718 }  
719 }  
720 }  
721 }  
722 }  
723 }  
724 }  
725 }  
726 }  
727 }  
728 }  
729 }  
730 }  
731 }  
732 }  
733 }  
734 }  
735 }  
736 }  
737 }  
738 }  
739 }  
740 }  
741 }  
742 }  
743 }  
744 }  
745 }  
746 }  
747 }  
748 }  
749 }  
750 }  
751 }  
752 }  
753 }  
754 }  
755 }  
756 }  
757 }  
758 }  
759 }  
760 }  
761 }  
762 }  
763 }  
764 }  
765 }  
766 }  
767 }  
768 }  
769 }  
770 }  
771 }  
772 }  
773 }  
774 }  
775 }  
776 }  
777 }  
778 }  
779 }  
780 }  
781 }  
782 }  
783 }  
784 }  
785 }  
786 }  
787 }  
788 }  
789 }  
790 }  
791 }  
792 }  
793 }  
794 }  
795 }  
796 }  
797 }  
798 }  
799 }  
800 }  
801 }  
802 }  
803 }  
804 }  
805 }  
806 }  
807 }  
808 }  
809 }  
810 }  
811 }  
812 }  
813 }  
814 }  
815 }  
816 }  
817 }  
818 }  
819 }  
820 }  
821 }  
822 }  
823 }  
824 }  
825 }  
826 }  
827 }  
828 }  
829 }  
830 }  
831 }  
832 }  
833 }  
834 }  
835 }  
836 }  
837 }  
838 }  
839 }  
840 }  
841 }  
842 }  
843 }  
844 }  
845 }  
846 }  
847 }  
848 }  
849 }  
850 }  
851 }  
852 }  
853 }  
854 }  
855 }  
856 }  
857 }  
858 }  
859 }  
860 }  
861 }  
862 }  
863 }  
864 }  
865 }  
866 }  
867 }  
868 }  
869 }  
870 }  
871 }  
872 }  
873 }  
874 }  
875 }  
876 }  
877 }  
878 }  
879 }  
880 }  
881 }  
882 }  
883 }  
884 }  
885 }  
886 }  
887 }  
888 }  
889 }  
890 }  
891 }  
892 }  
893 }  
894 }  
895 }  
896 }  
897 }  
898 }  
899 }  
900 }  
901 }  
902 }  
903 }  
904 }  
905 }  
906 }  
907 }  
908 }  
909 }  
910 }  
911 }  
912 }  
913 }  
914 }  
915 }  
916 }  
917 }  
918 }  
919 }  
920 }  
921 }  
922 }  
923 }  
924 }  
925 }  
926 }  
927 }  
928 }  
929 }  
930 }  
931 }  
932 }  
933 }  
934 }  
935 }  
936 }  
937 }  
938 }  
939 }  
940 }  
941 }  
942 }  
943 }  
944 }  
945 }  
946 }  
947 }  
948 }  
949 }  
950 }  
951 }  
952 }  
953 }  
954 }  
955 }  
956 }  
957 }  
958 }  
959 }  
960 }  
961 }  
962 }  
963 }  
964 }  
965 }  
966 }  
967 }  
968 }  
969 }  
970 }  
971 }  
972 }  
973 }  
974 }  
975 }  
976 }  
977 }  
978 }  
979 }  
980 }  
981 }  
982 }  
983 }  
984 }  
985 }  
986 }  
987 }  
988 }  
989 }  
990 }  
991 }  
992 }  
993 }  
994 }  
995 }  
996 }  
997 }  
998 }  
999 }  
1000 }
```

Prototyping: “v3JustAnotherComponent.cmp”

Use the service component

```
1 ▼ <aura:component>
2
3     <c:v3ServiceComponent aura:id="service" />
4
5     [ . . . ]
6
7 </aura:component>
```

Prototyping: “v3JustAnotherComponentHelper.js”

Invoke the service component

```
1  ({  
2    getContactList: function(component, filterString) {  
3      //call aura method from service component:  
4      component.find("service").callApex(component, "c.getContactsByFilter", {  
5        },  
6  
7      getContactListSuccess: function(component, returnValue) {  
8        //process result in some way  
9        component.set("v.contactList", returnValue);  
10      }  
11    })
```

“Nobody's gonna slow me down”

Highway to Hell - AC/DC

Improve the service component

Example: show a spinner, lock GUI, use storable actions

Reusable also means: other developer have to know how to use your component - **Documentation!**



Conclusion

What you have to do now

Build more components!

Identify recurring tasks / code!

Be generic to be reusable!

Fiddle around!

Do Trailheads!

RTFM!

Share your problem!

Join a local Developer Group!



Resources

Calling a Server-Side Action

https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/controllers_server_actions_call.htm

Using Object-Oriented Development

https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/oo_intro.htm

Calling Component Methods

https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/js_cmp_methods.htm

Providing Component Documentation

https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_documentation.htm

Code Examples

<https://github.com/appero-com/FTD17-Building-better-components-reusable-and-generic>

Join a local Developer Group

<https://developer.salesforce.com/dugs>





Q & A



Thank You

Remember to tell us what you think in the event survey
www.frenchtouchdreamin.com/survey/

Christian Menzinger
Product Developer
cmenzinger@appero.com
[@appero_com](https://twitter.com/appero_com)