Article
Chris Stewart · Apr 21, 2017 3m read

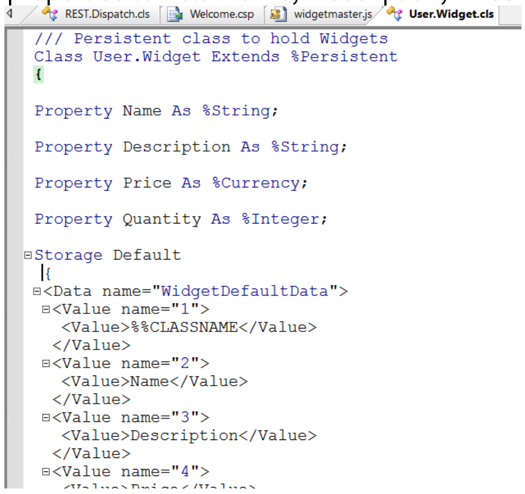
Let's write an Angular 1.x app with a Caché REST backend - Part 6

or *"Didn't you say you would cover Persistent Objects in* Part 5*, Chris?"*

Yes, that was the plan. This is a pretty important topic, so it get's its own Article

Up until now, we've display widget JSON that has been created by a basic loop. Clearly this isn't of much value. Now we have our stack connected together, and we can see that the data is flowing to the Welcome page, it's time to complete the stack and start feeding our service from "real" data.

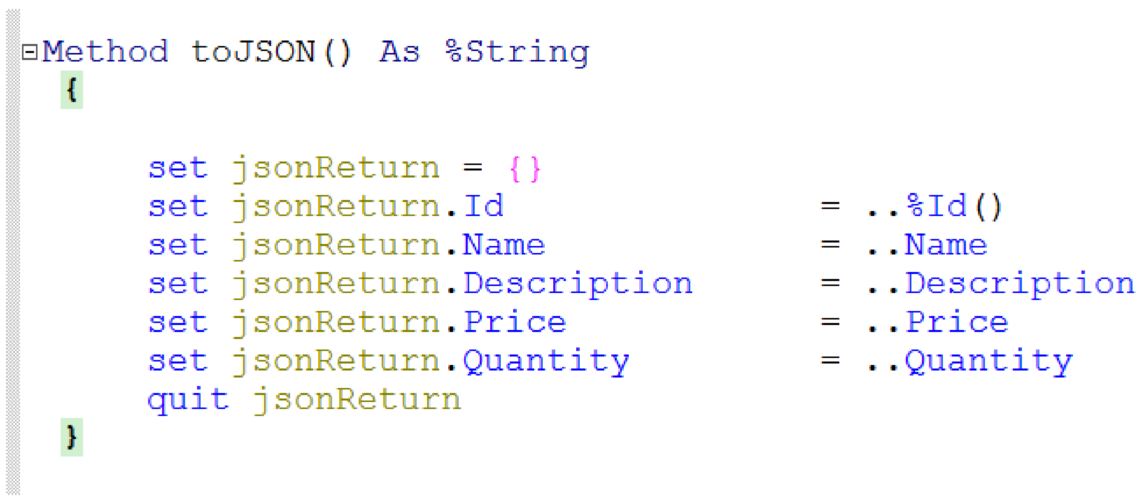
Let's start with our (very basic) Persistent class for Widgets. We have 4 properties to hold Name, Description, Price and current Quantity. We would like to expose all of these on our page.



```
/// Persistent class to hold Widgets
Class User.Widget Extends %Persistent
{

Property Name As %String;

Property Description As %String;

Property Price As %Currency;

Property Quantity As %Integer;

Storage Default
{
<Data name="WidgetDefaultData">
 <Value name="1">
   <Value>%%CLASSNAME</Value>
  </Value>
 <Value name="2">
   <Value>Name</Value>
  </Value>
 <Value name="3">
   <Value>Description</Value>
  </Value>
 <Value name="4">
```

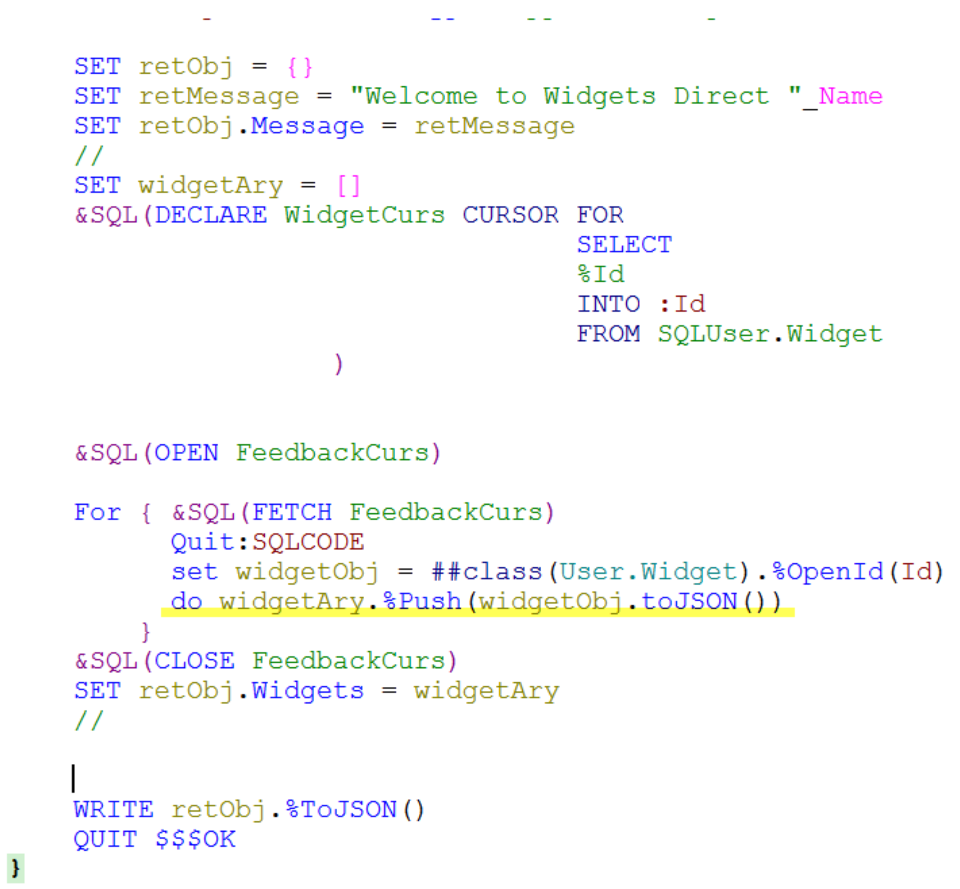
So, we could open up the object on our REST Service class and start spooling off properties into JSON objects. However, that will clutter up the Services and decouple the JSON generation from the source class. Instead, let's make a rule that all Persistent classes should have a method to represent themselves in a JSON form that we want (*what an oddly specific way to end that sentence, I wonder if it will become relevant again in the future?).*

So, we have decided that our Persistent classes should implement a Method toJSON, which will output a JSON representation of itself, as a %String. This is very straightforward on our very simple Widget class, we open a JSON object, and assign the object properties to the corresponding property on the JSON object (*though these property names do not* ***have*** *to match as in %Id).* After we have spooled all properties to the JSON object, we return it with our QUIT.



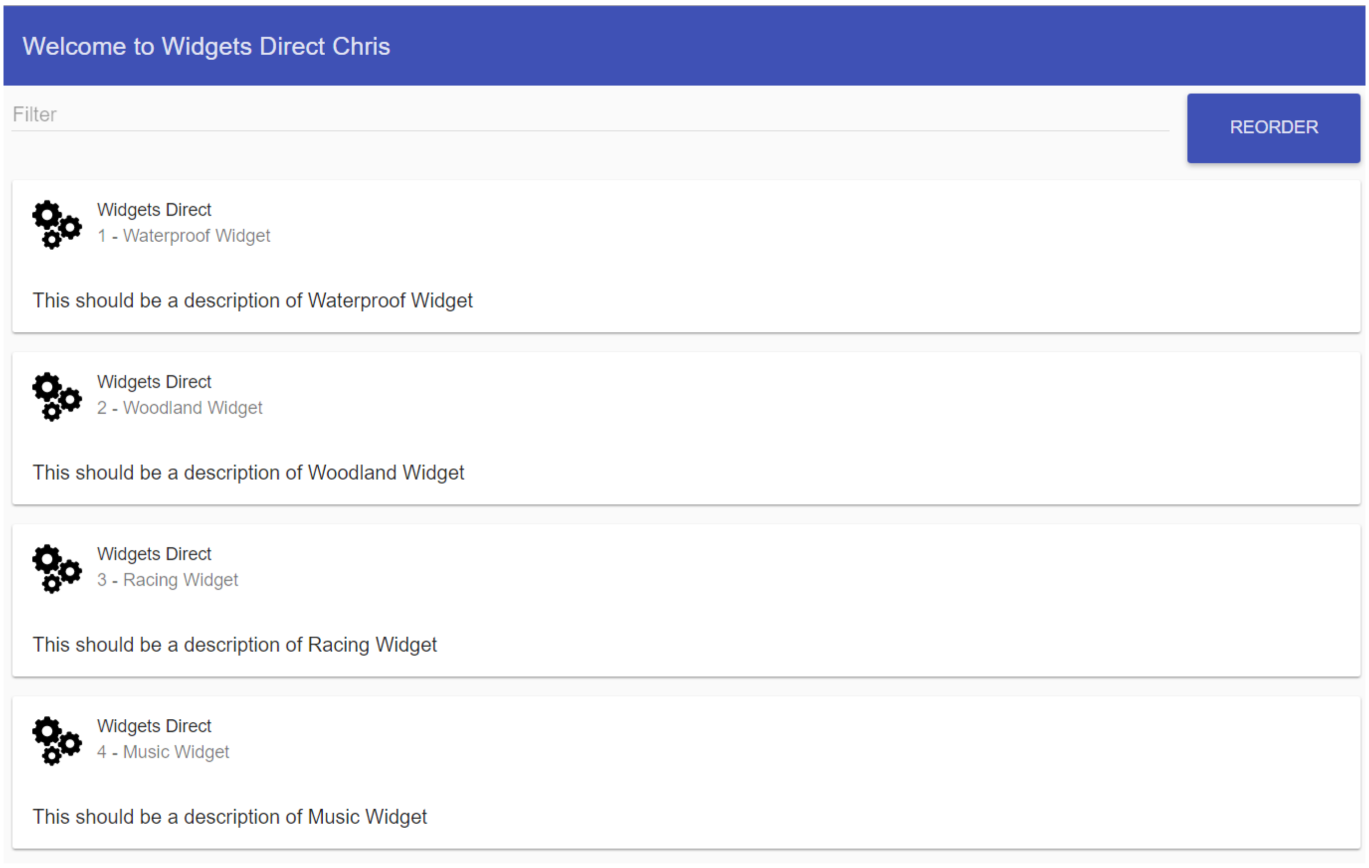
```
Method toJSON() As %String
{

    set jsonReturn = {}
    set jsonReturn.Id                  = ..%Id()
    set jsonReturn.Name                = ..Name
    set jsonReturn.Description         = ..Description
    set jsonReturn.Price               = ..Price
    set jsonReturn.Quantity            = ..Quantity
    quit jsonReturn
}
```

Now, we need to spool off our Records in our REST class. We will loop over all Widgets using &SQL in this instance, but your choice of iterator and search method is equally valid. We will loop over the %Id values, open the corresponding object, then push the output of that object's toJSON onto our widgetAry (the one we created in part 2).



```
    SET retObj = {}
    SET retMessage = "Welcome to Widgets Direct "_Name
    SET retObj.Message = retMessage
    //
    SET widgetAry = []
    &SQL(DECLARE WidgetCurs CURSOR FOR
                                 SELECT
                                 %Id
                                 INTO :Id
                                 FROM SQLUser.Widget
                  )


    &SQL(OPEN FeedbackCurs)

    For { &SQL(FETCH FeedbackCurs)
          Quit:SQLCODE
          set widgetObj  = ##class(User.Widget).%OpenId(Id)
          do widgetAry.%Push(widgetObj.toJSON())
        }
    &SQL(CLOSE FeedbackCurs)
    SET retObj.Widgets = widgetAry
    //

    WRITE retObj.%ToJSON()
    QUIT $$$OK
}
```

After a compile, we should now be feeding our array of Widgets from our Persistent class. Let's load up our page and see what is being displayed.


Welcome to Widgets Direct Chris

Widgets Direct
1 - Waterproof Widget

This should be a description of Waterproof Widget

Widgets Direct
2 - Woodland Widget

This should be a description of Woodland Widget

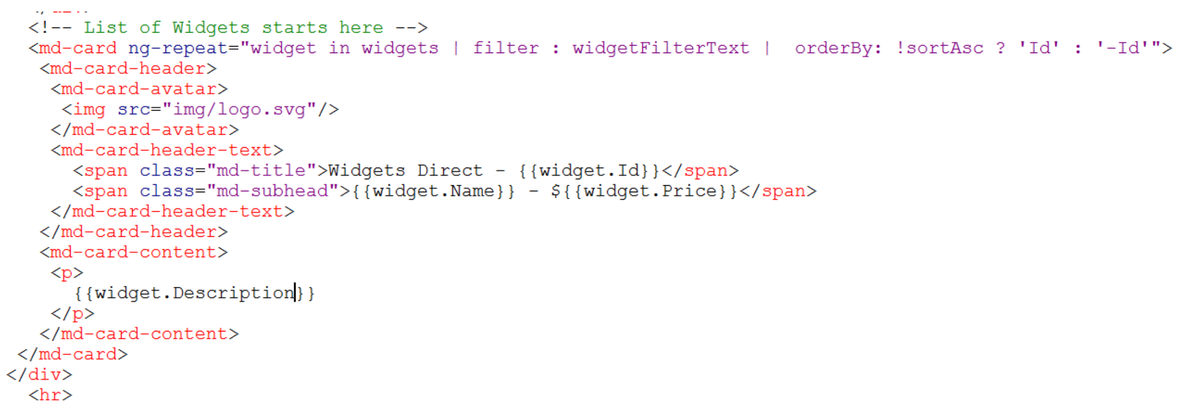
Widgets Direct
3 - Racing Widget

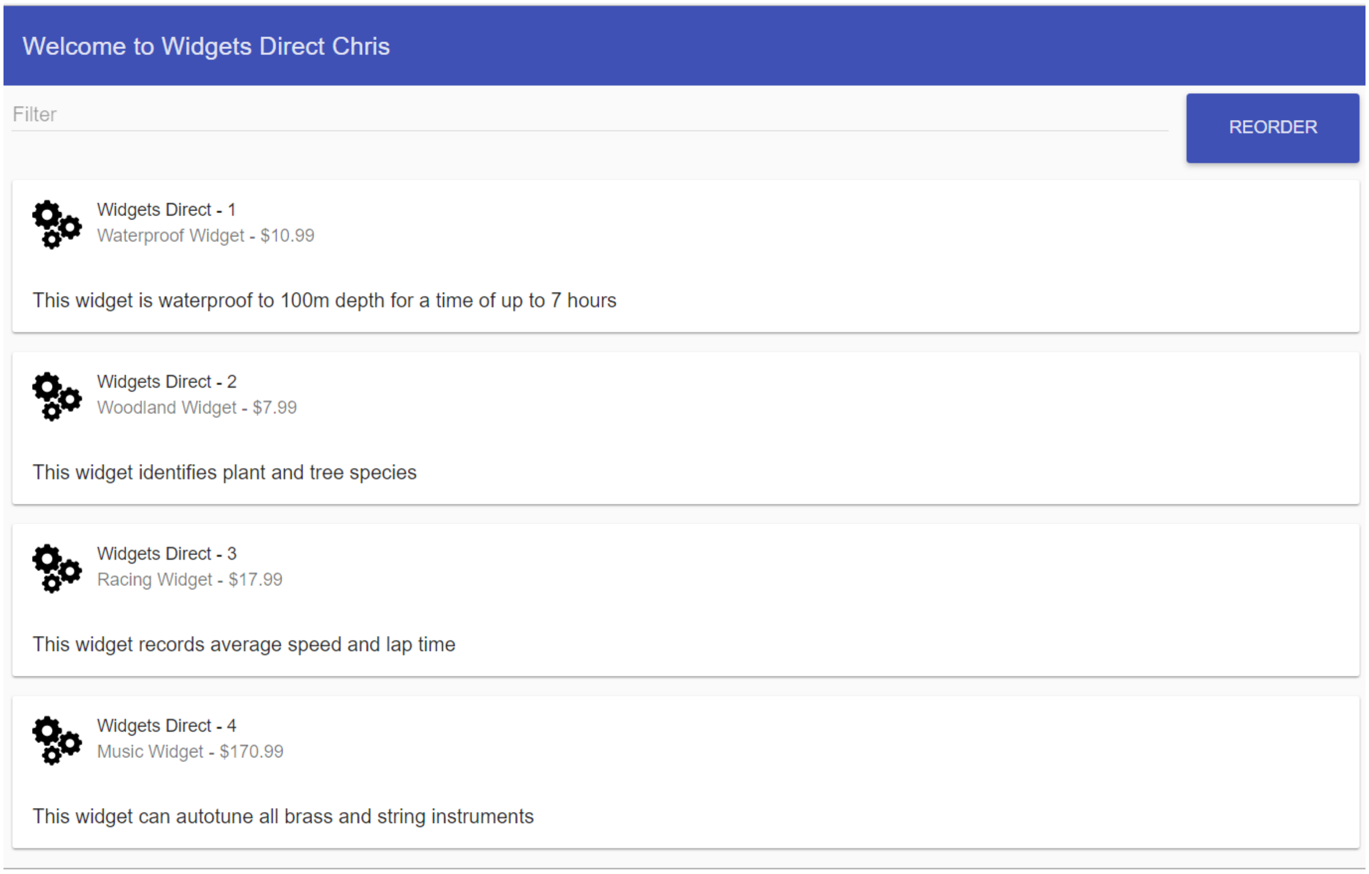
This should be a description of Racing Widget

Widgets Direct
4 - Music Widget

This should be a description of Music Widget

We have real Widgets! The output isn't really taking advantage of the new fields we are providing like Description, Price and Quantity. Let's slot them into the appropriate places on the Card template. Since we are binding the entire widget being returned from the service, we don't have to do anything special to start using the new fields, they are just available to reference


```
      <!-- List of Widgets starts here -->
      <md-card ng-repeat="widget in widgets | filter : widgetFilterText |  orderBy: !sortAsc ? 'Id' : '-Id'">
       <md-card-header>
        <md-card-avatar>
         <img src="img/logo.svg"/>
        </md-card-avatar>
        <md-card-header-text>
          <span class="md-title">Widgets Direct - {{widget.Id}}</span>
          <span class="md-subhead">{{widget.Name}} - ${{widget.Price}}</span>
        </md-card-header-text>
       </md-card-header>
       <md-card-content>
        <p>
          {{widget.Description}}
        </p>
       </md-card-content>
     </md-card>
    </div>
      <hr>
```


Welcome to Widgets Direct Chris

Filter

Widgets Direct - 1
Waterproof Widget - $10.99

This widget is waterproof to 100m depth for a time of up to 7 hours

Widgets Direct - 2
Woodland Widget - $7.99

This widget identifies plant and tree species

Widgets Direct - 3
Racing Widget - $17.99

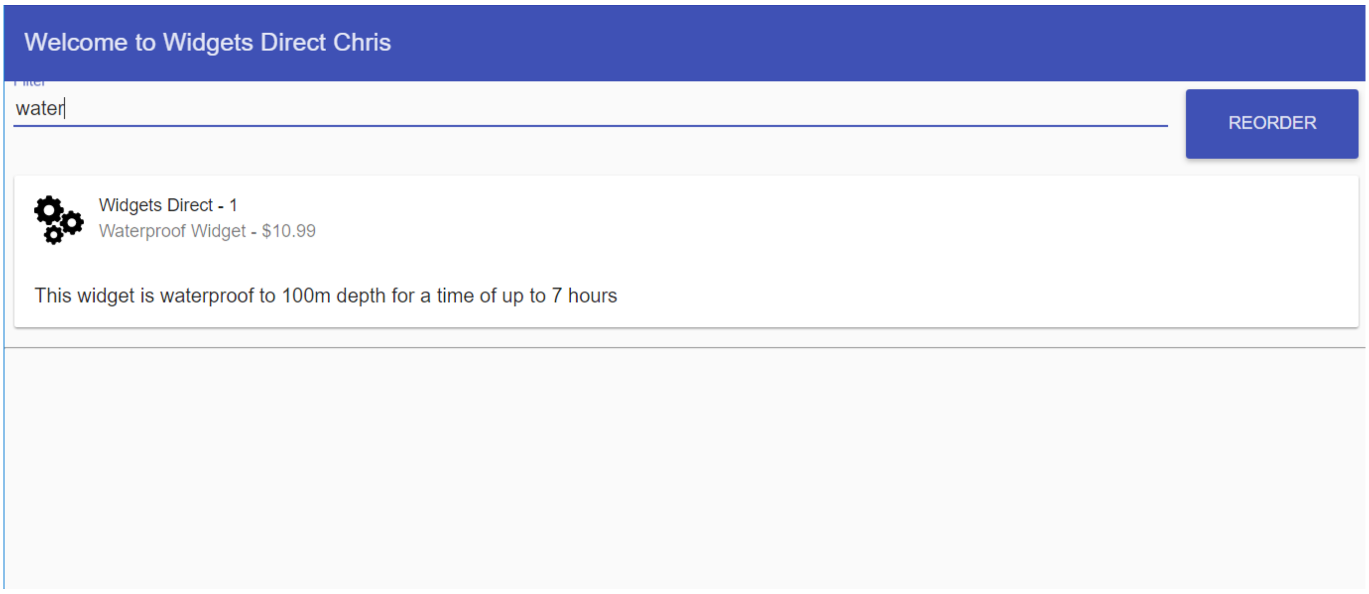
This widget records average speed and lap time

Widgets Direct - 4
Music Widget - $170.99

This widget can autotune all brass and string instruments

"But what about our filtering, Chris?" I hear you ask? Surely we have to rewrite something to get the new fields to be filtered, since we didn't know about these fields when we implemented the filtering?

NOPE! Filtering will automagically work on the new fields without any extra effort. Let's prove it by searching on Description


Welcome to Widgets Direct Chris

Filter
water

Widgets Direct - 1
Waterproof Widget - $10.99

This widget is waterproof to 100m depth for a time of up to 7 hours

Recap

In this lesson we:

1. Implemented our Persistent Widget class
2. Implemented the toJSON() method to provide a JSON representation of the object
3. Adjusted our REST.Dispatch class to output a list of Widgets by calling the toJSON() of each
4. Bound the new fields onto our Welcome page

In our next lesson we will:

- Break something, review the tools we can use to identify problems, then fix it

This article is part of a multi-part series on using Angular on top of Caché REST services. The listing of the full series can be found at the Start Here *page*

#Angular #CSP #HTML #JavaScript #JSON #REST API #Frontend #Caché