Making the most of $Query

Article
Timothy Leavitt · Mar 17 3m read

Making the most of $Query

I ran into an interesting ObjectScript use case today with a general solution that I wanted to share.

Use case:

I have a JSON array (specifically, in my case, an array of issues from Jira) that I want to aggregate over a few fields - say, category, priority, and issue type. I then want to flatten the aggregates into a simple list with the total for each of the groups. Of course, for the aggregation, it makes sense to use a local array in the form:

```
agg(category, priority, type) = total
```

Such that for each record in the input array I can just:

```
Do $increment(agg(category, priority, type))
```

But once I've done the aggregation, I want to get that into an easier form to iterate over, like an integer-subscripted array:

```
summary = n
summary(1) = $listbuild(total1, category1, priority1, type1)
...
summary(n) = $listbuild(totalN, categoryN, priorityN, typeN)
```

Basic Solution:

The simple approach is just to have three nested For loops with $Order - for example:

```
Set category = ""
For {
    Set category = $Order(agg(category))
    Quit:category=""

    Set priority = ""
    For {
        Set priority = $Order(agg(category,priority))
        Quit:priority=""

        Set type = ""
        For {
            Set type = $Order(agg(category,priority,type),1,total)
            Quit:type=""
```

```
            Set summary($i(summary)) = $listbuild(total,category,priority,type)
        }
    }


}
```

That's what I started out with, but it's a lot of code, and if I had more dimensions to aggregate over it'd get unwieldy quickly. It made me wonder - is there a general solution to accomplish the same thing? It turns out, there is!

Better Solution with $Query:

I decided that using $query would help. Note that this solution assumes uniform depth of subscripts/values across the whole local array; weird things would happen if this assumption was violated.

```
ClassMethod Flatten(ByRef deep, Output flat) [ PublicList = deep ]
{
    Set reference = "deep"
    For {
        Set reference = $query(@reference)
        Quit:reference=""
        Set value = $listbuild(@reference)
        For i=1:1:$qlength(reference) {
            Set value = value_$listbuild($qsubscript(reference,i))
        }
        Set flat($i(flat)) = value
    }
}
```

So the above snippet is replaced with:

```
Do ..Flatten(.agg,.summary)
```

A few things to note about this solution:

- *deep* needs to be in the PublicList for $query to be able to operate on it
- in each iteration, *reference* is changed to reference the next set of subscripts in *deep* that has a value - e.g., the value might be: *deep("foo","bar")*
- $qlength returns the number of subscripts in *reference*
- $qsubscript returns the value of the i'th subscript of *reference*
- When $listbuild lists are concatenated, a valid $listbuild list with the combined lists is the result (this is way better than using any other delimiter!)

Summary

$query, $qlength, and $qsubscript are handy for dealing with globals/local arrays of arbitrary depth.

Further Reading

$Query: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/DocBook.UI...
$QSubscript: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.c...
$QLength: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.c...

#Best Practices #Code Snippet #ObjectScript #Caché #InterSystems IRIS
100 2 0 9 463

Log in or sign up to continue
Add reply

Source URL: https://community.intersystems.com/post/making-most-query