Article
Tani Frankel · Jun 4, 2020 10m read

A Hidden Object Design Consideration “ Journal Killer ” [... And an even more secret Business Process Parameter]

In this article I'd like to share with you a phenomena that is best you avoid - something you should be aware of when designing your data model (or building your Business Processes) in Caché or in Ensemble (or older HealthShare Health Connect Ensemble-based versions).

Background

As a reminder, every (well⋯ almost every) SET and KILL you perform of a global to the database (directly or via some interface like an object %Save() or SQL DDL) gets written to the database Journal.

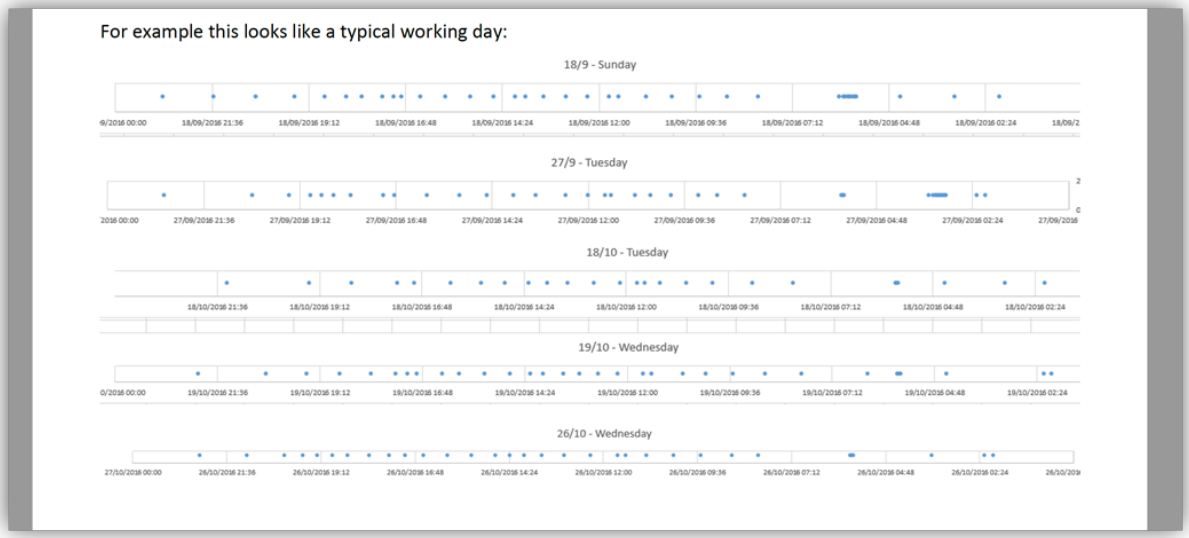
[You can see this article with more background about this]

In this article I ’ d like to highlight a consideration that could cause you (might even already been causing you) serious journaling consumption.

I ’ ll give you a little background of how I got on to this topic (and this could also illustrate the results of this phenomena).

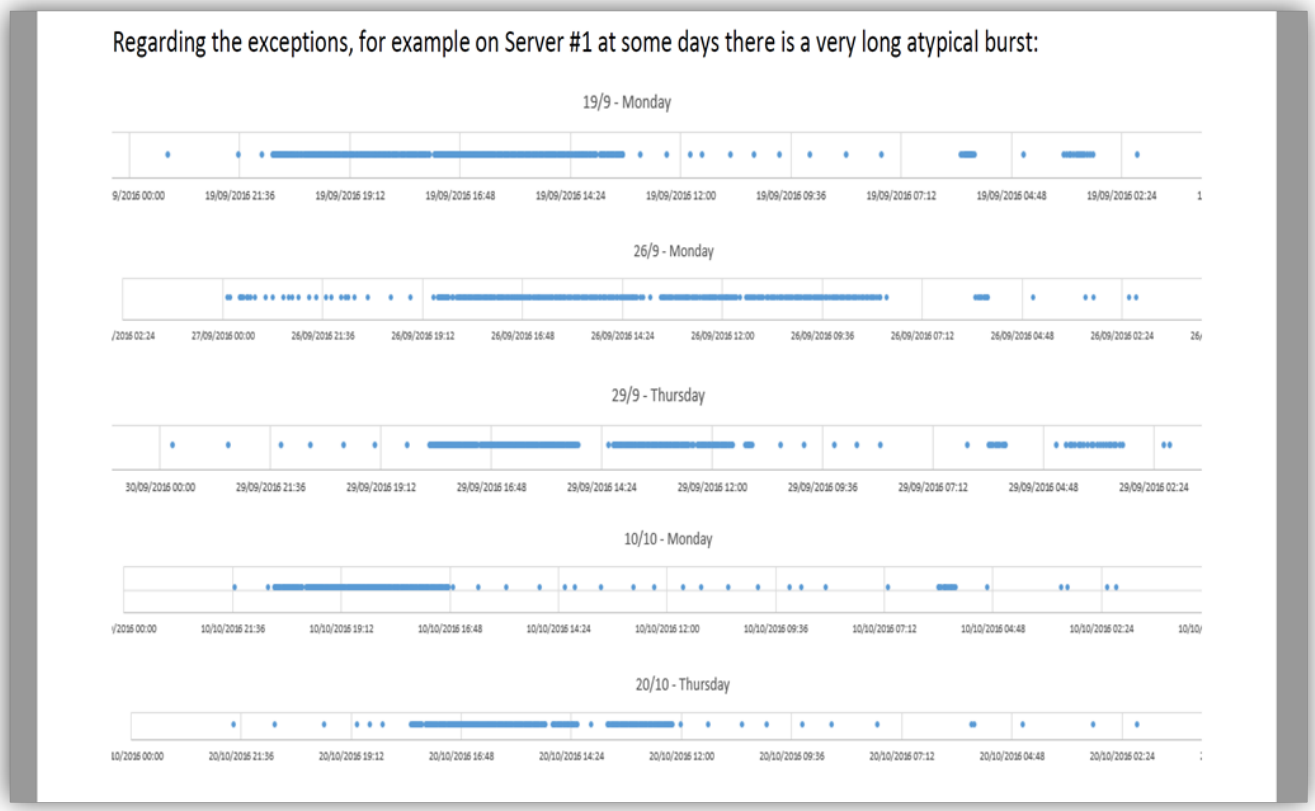
A customer was complaining of extreme journal growth.

We examined the journal usage, here ’ s a certain visualization of “ regular days ” journal consumption at this customer –



Each “ dot ” signifies a 1GB-sized journal file switch.

And this is what the “ bad days ” looked like –



You can see the " lines " of " connected dots " that signify consecutive heavy journal consumption.

Looking closer (see this article with tips of how to analyze the Journal) we learned the reason for this was the phenomena described below.

Example

The scenario in which this happens is when you have an object with an Array Collection (or some other " Collection " that " behaves " like an Array storage-wise).

Each time, you save the object, regardless of how many items in the collection you changed (or even if you changed them at), the code behind the scenes KILLs the whole Collection global subscript node, and re-SETs.

For example, assume you have the following class –

```
Class Demo.Data.Test Extends %Persistent
{
/// test type
Property TypeCode As %String;
/// related runs
Property RelatedRuns As array Of %String;
Storage Default
{
<Data name="RelatedRuns">
<Attribute>RelatedRuns</Attribute>
<Structure>subnode</Structure>
<Subscript>"RelatedRuns"</Subscript>
</Data>
<Data name="TestDefaultData">
<Value name="1">
```

```
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>TypeCode</Value>
</Value>
</Data>
<DataLocation>^Demo.Data.TestD</DataLocation>
<DefaultData>TestDefaultData</DefaultData>
<IdLocation>^Demo.Data.TestD</IdLocation>
<IndexLocation>^Demo.Data.TestI</IndexLocation>
<StreamLocation>^Demo.Data.TestS</StreamLocation>
<Type>%Library.CacheStorage</Type>
}
```

Then in the %SaveData label of the generated routine you will find this, with regard to updating the Collection, in case of updating an existing object instance:

```
kill ^Demo.Data.TestD(id,"RelatedRuns")
set zzc40v1=$Order(i%RelatedRuns(""),1,val)
While zzc40v1'="" {
    Set ^Demo.Data.TestD(id,"RelatedRuns",zzc40v1)=val,zzc40v1 = $Order
(i%RelatedRuns(zzc40v1),1,val)
}
```

You can see we KILL the whole " previous " node, and then loop over the " current " items and SET them.

Journal-wise this will lead to an entry for each time in the Collection – twice – once during this KILL and once for the SET (assuming it ' s still there).

So assume a scenario where we have a loop adding items to a collection, each item each iteration, with a save in each iteration.

The first iteration would have 1 Journal entry (for the Collection, we ' ll focus just on this aspect).

The 2nd iteration will have 3 entries – 1 for killing the 1st item, then another 2 for setting the 1st (again) and the 2nd.

The 3rd iteration will have already 5 entries – 2 for killing the 2 existing items, and 3 for (re)setting the previous 2, plus adding the 3rd new one.

The 4th iteration will have now 7 entries - killing 3, setting 4.

You can see that in general each iteration n has $2n-1$ journal entries.

And in total after 4 iterations we had: 1 + 3 + 5 + 7 = 16 entries.

And in general, after n iterations we ' d have n^2 Journal entries.

Here ' s what the Journal looks like after running such a loop as per above –

```
USER>set test=##class(Demo.Data.Test).%New() for i=1:1:4 { do test.RelatedRuns.SetAt(
i,i) do test.%Save() }



  Address   Proc ID Op Directory        Global & Value
================================================================================
```

```
364040     22092 BT
364056     22092 S  c:\intersystems+ Demo.Data.TestD = 1
364112     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",1) = 1
364184     22092 ST c:\intersystems+ Demo.Data.TestD(1) = $lb("","")
364240     22092 CT
364256     22092 BT
364272     22092 kT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",1)
364344     22092 KT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns")
364408     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",1) = 1
364480     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",2) = 2
364552     22092 ST c:\intersystems+ Demo.Data.TestD(1) = $lb("","")
364612     22092 CT
364628     22092 BT
364644     22092 kT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",1)
364716     22092 kT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",2)
364788     22092 KT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns")
364852     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",1) = 1
364924     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",2) = 2
364996     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",3) = 3
365068     22092 ST c:\intersystems+ Demo.Data.TestD(1) = $lb("","")
365128     22092 CT
365144     22092 BT
365160     22092 kT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",1)
365232     22092 kT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",2)
365304     22092 kT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",3)
365376     22092 KT c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns")
365440     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",1) = 1
365512     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",2) = 2
365584     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",3) = 3
365656     22092 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",4) = 4
365728     22092 ST c:\intersystems+ Demo.Data.TestD(1) = $lb("","")
365788     22092 CT
```

Now imagine an array with 1000s or 10,000s of items… and say you have 100s or more of such objects… your Journal would get very big very fast.

Note I ' m focusing here on the Journal space aspect here, but of course the writing performance impact (to the Journal file and possibly to the CACHE.DAT file) is also an issue.

The Consideration

So, the general rule of thumb for using Collections (and this has been mentioned in different contexts over the years, for various considerations, not just this Journal one) is to use it for small-sized Collections. This way even if there is this Journal-related overhead it should not be significant.

Alternatives could include (but not limited to) using references with Foreign Keys to connection/relation-tables.

Note for InterSystems IRIS

It is important to note that in InterSystems IRIS (as of version 2019.1; including of course IRIS for Health and the related HealthShare Health Connect based on these versions) this behavior has changed (for internal reference the related change is identified as MAK4939). And updating an object with a Collection will not cause all of the sub-node Collection items to be deleted and set all over again.

Referring to the example we used above, this is the generated code in the routine pertaining to the same area of code:

```
kill nodes merge nodes=i%RelatedRuns
set zzc40v1=""
for {
    set zzc40v1 = $Order(^Demo.Data.TestD(id,"RelatedRuns",zzc40v1),1,data)
    Quit:zzc40v1=""
    if $data(nodes(zzc40v1),val) {
        if data=val kill nodes(zzc40v1)
    } else {
        kill ^Demo.Data.TestD(id,"RelatedRuns",zzc40v1)
    }
}
merge ^Demo.Data.TestD(id,"RelatedRuns")=nodes
```

You can see here we use a temporary local array, and merge it in (and kill only necessary removal of the Collection items).

The same loop above would generate in InterSystems IRIS the following Journal entries –

```
  Address   Proc ID Op Directory        Global & Value
================================================================================
   1471332     28064 BT
   1471348     28064 S  c:\intersystems+ Demo.Data.TestD = 1
   1471400     28064 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",1) = 1
   1471468     28064 ST c:\intersystems+ Demo.Data.TestD(1) = $lb("","")
   1471524     28064 CT
   1471540     28064 BT
   1471556     28064 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns") = ""
   1471620     28064 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",2) = 2
   1471692     28064 ST c:\intersystems+ Demo.Data.TestD(1) = $lb("","")
   1471752     28064 CT
   1471768     28064 BT
   1471784     28064 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns") = ""
   1471848     28064 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",3) = 3
   1471920     28064 ST c:\intersystems+ Demo.Data.TestD(1) = $lb("","")
   1471980     28064 CT
   1471996     28064 BT
   1472012     28064 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns") = ""
   1472076     28064 ST c:\intersystems+ Demo.Data.TestD(1,"RelatedRuns",4) = 4
   1472148     28064 ST c:\intersystems+ Demo.Data.TestD(1) = $lb("","")
   1472208     28064 CT
```

Here you can see the minimum number of entries (*n*), per each item update.

Real-life Scenarios

At this stage you might be asking yourself what is the probability of running into this scenario, when would I have such a loop (or similar).

I ' ll give you two examples, both real-life ones.

The first relates to the customer use-case I mentioned above (with the " dots " visualization of the Journal switches).

In their solution they had an entity that was created and updated according to incoming messages (specifically HL7 and DICOM messages), and so, in order to keep track (including display in a UI) the messages pertaining to each entity, these entities had an array of Message IDs that were related to it.

Typically, this number of " Relates Messages " would be less than 10 and no more than 20 or 30. But occasionally one of the sending systems would get into a " sending spree " and send continuously many messages, for several entities, causing the Collection to grow to 10,000s.

Here's a small snippet of what the Jounral looked like:

17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",255)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",256)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",257)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",258)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",259)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",260)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",261)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",262)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",263)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",264)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",265)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",266)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",267)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",268)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",269)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",270)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",271)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",272)	o:\ensemblenwa\db\
17688	KILLdes	Yes		.Data.PatientD(100185,"RelatedMessages",273)	o:\ensemblenwa\db\
17688	KILL	Yes		.Data.PatientD(100185,"RelatedMessages")	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",1)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",2)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",3)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",4)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",5)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",6)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",7)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",8)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",9)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",10)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",11)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",12)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",13)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",14)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",15)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",16)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",17)	o:\ensemblenwa\db\
17688	SET	Yes		.Data.PatientD(100185,"RelatedMessages",18)	o:\ensemblenwa\db\

Business Process Use-case

The second real-life use-case is a much more common one, one that probably anyone who uses Ensemble Business Processes runs into (knowingly or unknowingly). And will bring us to the sub title of this article – "An even

more secret Business Process Parameter " .

It might be little unknown that a Business Process is a Persistent entity. I assume many are aware of this fact (as it has an ID for example which is sometimes references, and there is a page in the Management Portal: View -> " Business Process Log/Instances " that shows the records).

But I assume much less are aware that this entity holds two Collections (with a sub-node storage):

%MessagesRecieved & %MessagesSent

As their names hint these Collections hold all the messages a Business Process has sent or received during its life span. They are intended to make it easier for developers to perform searches on what has been already sent or received for a given BP in a performant manner, without having to traverse all BPs.

But as per above this is prone to the issue described above.

Imagine a BP that makes a call to a BO to select rows from a database, then inside the BP there is a loop over the rows returned, and it sends each row ' s data as a request to another BO to perform some action on it. Quite a common scenario for various use-cases. Each iteration will add message IDs to the Collections mentioned above, and these will grow per the numbers of rows processed. In some cases of course this could be 100s or more.

Here a snippet from a Journal of a customer system where this happened:

Process	Type	InTransaction	GlobalNode	Database
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11153)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11154)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11155)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11156)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11157)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11158)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11159)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11160)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11161)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11162)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11163)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11164)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11165)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11166)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11167)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11168)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11169)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11170)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11171)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11172)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11173)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11174)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11175)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11176)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11177)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11178)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11179)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11180)	c:\intersystems\ensemble\
6384	SET	Yes	^Ens.BusinessProcessD(23260,"received",11181)	c:\intersystems\ensemble\

Now, unlike the general case mentioned above, where you can control your own data model and address this issue, keeping this consideration in mind – what you can do about the Business Process Collections…?

Here comes the Class Parameter, from the Ens.BusinessProcess class reference –

parameter **SKIPMESSAGEHISTORY** = 0;

If this parameter is TRUE, then arrays %MessagesSent and %MessagesReceived will not be populated.

So if you want to avoid these Collections from being populated (and accumulating and bloating your Journals) set this parameter to the value of 1.

Note that if you use the graphic BPL Designer for your Business Process, you will need to open the class definition (in your favorite IDE) and add this parameter.

And a reminder – in InterSystems IRIS (and IRIS for Health, and IRIS-based HealthShare Health Connect) this is less of a concern as per above, still if you don ’ t need these Collections, you can use this parameter as well.

Now you can't say you were not aware...

#Business Process (BPL) #Databases #Data Model #Interoperability #Journaling #Object Data Model #Tips & Tricks #Health Connect #InterSystems IRIS #InterSystems IRIS for Health