Article
Brendan Bannon · Dec 27, 2016 8m read

The Art of Mapping Globals to Classes (5 of 3)

Mapping Examples

Clearly if you have a fourth article in the trilogy you need to go for the money grab and write the fifth, so here it is!

Note: Many years ago Dan Shusman told me that mapping globals is an art form. There is no right or wrong way of doing it. The way you interpret the data leads you to the type of mapping you do. As always there is more than one way to get to a final answer. As you look through my samples you will see there are some examples that map the same type of data in different ways.

At the end of this article there is a zip file with the collection of mapping examples I have written for customers over the years. Below, I will point out a couple that will highlight some things I referenced in my 4 previous mapping articles. As this is a pure money grab there will not be the same level of detail as the last 4. If something doesn ' t make sense please let know and I will go into more detail with you.

Row ID Spec:

Example Class: Mapping.RowIdSpec.xml

I promised this several times in earlier articles. You need to define this only when your subscript expressions are not simple fields. In my example I am taking the value that is stored in the subscript and multiplying it by 100, so when I look at the global I need to have 1.01 but I want the logical value to be 101. So Subscript 2 has an expression of {ClassNumber}/100 and the RowIdSpec is {L2}*100. I always do this backwards the first time. The Subscript Expression is taking the logical value and use that in the $ORDER() of the global so divide by 100. The RowId Spec is taking the value from the global and building the logical value so multiply by 100.

You could also do this by writing Next Code and an Invalid Condition that would handle the multiplying and dividing.

Subscript Expression Type Other / Next Code / Invalid Condition / Data Access:

Example Class: Mapping.TwoNamespacesOneGlobal.xml

Wow what a gold mine this class is! This one class uses half the stuff I want to talk about. This class loops over the same global in 2 different namespaces. We can ' t use subscript level mapping because both namespaces use the same subscript values. Instead we are going to use extended Global Reference syntax, ^|namespace|GlobalName, first looping over the global in the USER Namespace and then looping over the same global in the SAMPLES namespace. The IDKey for this table will be made up of 2 parts: Namespace and Sub1.

Subscript Expression Other: In Subscript Level 1 we are not doing a $ORDER() or $PIECE(), instead we are setting {L1} to 1 of 2 hard coded values: USER or SAMPLES. No global is used at all here so the Type is 'Other'.

Next Code: If you are using a Subscript Expression of Type 'Other' you will need to provide Next Code (OK so you could write a complex expression to do this, but either way you are providing the code). The first time the Next Code gets called {L1} will be set to the 'Start Value', the default is the empty string. Your next code should set {L1} equal to Empty string when you are done looping. For this example {L1} will be set to "USER", "SAMPLES" and "" the three times it is called. The Next Code in Subscript Level 2 will get reset to "" for the 2 good values returned by Subscript Level 1. The Next Code here is just a simple $$ORDER() over the global with the Extended Reference.

Invalid Condition: Both Subscript Levels 1 and 2 have Next Code that means they both need an Invalid Condition. Given a value for this Subscript Level the condition should return 1 if it is a bad value. For {L1} if the value is not "USER" or "SAMPLES" it will return 1. For example:

SELECT * FROM Mapping.RowIdSpec WHERE NS = "%SYS"

will return no rows because of the Invalid Condition for {L1}

Data Access: Say you have a global with an IdKey based on 2 properties in a global like ^glo({sub1},{Sub2}) If you provide a value for {Sub1} before we start loop on {Sub2} we will check to see if there is data at the previous level by doing a $DATA(^glo({Sub1}). In this example we do not have a global in Subscript Level 1 so we need to be told what to test. This is the Data Access Expression: ^|{L1}|Facility. The next example also needs a Data Access expression and it might be easier to follow. If this one is confusing, look at the next example.

Data Access / Full Row Reference:

Example Class: Mapping.TwoNamespacesOneGlobal2.xml

This class is mapping the same data as the previous one. The difference is in Subscript Level 1. Instead of hard coding the values for the Namespace, this class has a second global containing the Namespaces we want to loop over: ^NS("Claims",{NS}). Not only does this simplify the mapping, it make thing more flexible as you can add a new namespace just by setting a global instead of modifying the class mapping.

Data Access: The 'Global' in the mapping is defined as ^NS since that is the global we are looping over in Subscript Level 1 and 2. For Subscript Level 3 we want to switch to ^|{NS}|Facility({Sub1}). Before we can use a different Global in the Next Code we need to define it in the 'Data Access'. {L1} was just a constraint in the ^NS global and is not used in the ^Facility global so we just level it out. {L2} now is used as the Namespace Reference in the Extended Global syntax: ^|{L2}|Facility. In this class the 'Next Code' is not defined (so it was not needed in the last example either). The class compiler takes the 'Data Access' and uses that to generate the $ORDER() needed at this level.

Full Row Reference: Similar to the 'Invalid Condition" this is used to reference the row if all parts of the IdKey are used: ^|{L2}|Facility({L3}). I think we could get away without defining the 'Full Row Reference' for this class, but it does not hurt. If you have 'Next Code' that changes the logical value of a subscript before looping over the global then you would need to define the 'Full Row Reference'. For example as I said above, the RowIdSpec class could have been done using 'Next Code'. If you went that route the 'Full Row Reference' would have been ^Mapping({L1},{L2}/100).

Subscript Access Type Global:

Example Class: Mapping.TwoGlobals.xml

This class is displaying data from 2 different globals: ^Member and ^Provider. In the last example we started in one global and then switched to a different global to get at a row. In this case we have two globals that both contain rows. We will loop over the all the rows in the first global and then loop over the rows in the other global.

Subscript Expression Type Global: When you defined the first subscript level as ‘ Global ’ the map needs to have the Global Property set to “ * ”. I bet we could have used this style of mapping to do Mapping.TwoNamespacesOneGlobal. Remember you are an artist when creating these mappings!

In {L1} the Access Type is ‘ Global ’ and the ‘ Next Code ’ will return “ Member ”, “ Provider ” and “ ”. By defining th Access Type as ‘ Global ’, the compiler knows to use {L1} as a global name so {L2} and {L3} are just simple subscripts. {L4} is also a subscript but it has some extra code to deal with the fact that the two globals store the properties in different locations.

This class shows one more interesting thing in the Data section of the mapping. If we were only mapping the ^Member global I would have only defined three subscript levels and the Data section would have had Node values of 4, 5, 6, 7, 8 and 9. To deal with Zip Code being in node 9 or in node 16 we add the base node to the IdKey 4 or 10, and use +6 in the Node to get the offset to the Zip Code.

Access Variables:

Example Class: Mapping.SpecialAccessVariable.xml

Special Access Variables can be defined at any subscript level. There can be more than 1 per level. In this example we have one defined called {3D1}. The 3 means it is defined at subscript level 3 and the 1 means it is the first variable defined at this level. The compiler will generate a unique variable name for this and handle defining it and removing it. The variable gets defined after you have executed the ‘ Next Code ’. In this example I want to use the variable in the ‘ Next Code ’ of level 4 so I have it defined in level 3. In this example I am using {3D1} to “ remember ” where we were in the looping so we can change an invalid date value to “ * ” but still come back to the same place in the looping.

Bitmaps:

Example Class: Mapping.BitMapExample.xml

While Bitmap indices are kind of new, it doesn ’ t mean you would not want to add them to your application that is using Cache SQL Storage. You can add a Bitmap index to any class that has a simple positive %Integer IdKey.

You define the ‘ Type ’ as “ bitmap ” and you do not include the IdKey as a subscript. The one thing you need to remember when you define a bitmap is that you also need a Bitmap Extent. Again nothing special, the Extent is an index of IdKey values, so no subscripts are needed and the ‘ Type ” is “ bitmapextent ”.

The class has methods you can call to maintain the bitmap indices as you modify the data via Sets and Kills. If you can use SQL or Objects to make changes then the indices will be maintained automatically.

So some of those examples are a little involved! Have a look at the classes and see if you can make heads or tails of this. If not let me know: Brendan@interSystems.com, always happy to help a struggling artist.

Get the Examples here.

If you want to jump back and look at the other mapping articles here are the links:

The Art of Mapping Globals to Classes (1 of 3)

The Art of Mapping Globals to Classes (2 of 3)

The Art of Mapping Globals to Classes (3 of 3)

The Art of Mapping Globals to Classes (4 of 3)

#Mapping #Object Data Model #SQL #Caché

Source URL:https://community.intersystems.com/post/art-mapping-globals-classes-5-3