Article
Alessandro Marin · Aug 14, 2018 5m read

Triggering DeepSee synchronization when source class properties reference a different class

The source class of a DeepSee cube has a property referencing a different class:

```
Class ClassA Extends %Persistent {
     Property P1 As ClassB;
}
```

When records in class B change, the ^OBJ.DSTIME global for Class A will not be automatically updated. This means that synchronization of cubes based on source class A will not reflect the changes occurred to property P1. This post will help you determine the best way to achieve synchronization of properties referencing a different class

Example problem

Let's describe the problem using the PATIENTS cube in SAMPLES. The cube's "Patient" source class is defined in DeepSee.Study.Patient.cls and has a PrimaryCarePhysician property. This property is defined as an object of a "Doctor" class DeepSee.Study.Doctors.cls:

```
Class DeepSee.Study.Patient Extends %Persistent {
    Parameter DSTIME = "AUTO";
    ..
    Property PrimaryCarePhysician As DeepSee.Study.Doctor;
    ..
}
```

We can edit a Doctor instance by changing a doctors' group from "II" to "III":

```
SAMPLES>Set doc = ##class(DeepSee.Study.Doctor).%OpenId(15)

SAMPLES>Write doc.DoctorGroup II

SAMPLES>Set doc.DoctorGroup="III"

SAMPLES>Do doc.%Save()
```

The problem we want to address is that **changes in Doctors instances will not be reflected in the PATIENTS cube after cube synchronization**. This can be shown by the following code.
First get the number of facts in PATIENTS with doctors in the "II" group before synchronization:

```
SAMPLES>Set MDX="SELECT  FROM [PATIENTS] %FILTER [DOCD].[H1].[DOCTOR GROUP].&[II]"
SAMPLES>Set rs=##class(%DeepSee.ResultSet).%ExecuteDirect(MDX)
```

```
Do rs.%Print()
Result:                 322
```

Manually synchronize the Patients cube after the change to the Doctors class above:

```
SAMPLES>Do ##class(%DeepSee.Utils).%SynchronizeCube("PATIENTS")
No changes detected.
```

Notice that the no changes were detected in the source class, and in fact the number of facts in PATIENTS with doctors in the "II" group does not change:

```
SAMPLES>Set rs=##class(%DeepSee.ResultSet).%ExecuteDirect(MDX)  Do rs.%Print()
Result:                 322
```

The reason for this behavior is that when the PATIENTS cube is synchronized, DeepSee reads the Patient IDs stored in ^OBJ.DSTIME("DeepSee.Study.Patient"). When we changed DoctorGroup for a Doctor instance, no Patient record was modified and no entries were added to ^OBJ.DSTIME("DeepSee.Study.Patient"). As a result, changes in Doctors cannot be detected by the synchronization feature (see the documentation).

One option to work around this behavior is to run a full cube build. However, building cubes can take relatively long and during a build DeepSee queries cannot run on the cube.

Another option would be to use a property using the Get value at run time setting for the Doctor Group level in Patients. This allows the facts in different Doctor Groups to be calculated at runtime based on the source data (see also this question) . This would probably be the best approach with the cubes used in this example, but let's assume we cannot modify the Patients cube or we prefer to minimize computation time for DeepSee queries.

Another option is to use the Pre-Synchronize code or a Task to get the newly updated instances, and then manually update ^OBJ.DSTIME for the Patients class (we will see how to do this below).
This post shows how to use triggers to enhance the default synchronization feature.

The trigger approach

The following approach enables a cube to get synchronized when tables other than the source table get updated. This solution uses triggers in Doctors to update the synchronization globals for the PATIENTS cube.

```
Class DeepSee.Study.Doctor Extends %Persistent {
..

/// Trigger allowing the PATIENTS cube to detect updates on Doctors
Trigger OnUpdateDelete [ Event = UPDATE/DELETE, Foreach = row/object ]
{
    // Get row id of inserted row
    New id
    Set id = {ID}
    // Get the Patient IDs affected by changes of one record in the Doctor table
    Set query = "SELECT ID FROM DeepSee_Study.Patient WHERE PrimaryCarePhysician="_id
    Set tStatement = ##class(%SQL.Statement).%New()
    Set qStatus = tStatement.%Prepare(query)
    Set rset = tStatement.%Execute()
    // Loop through Patient IDs and update ^OBJ.DSTIME for Patients
    While rset.%Next() {
        Set ID = rset.%GetData(1)
        Do ##class(%DeepSee.Utils).%SetDSTimeIndex("DeepSee.Study.Patient",ID,0)
```

```
    }
}
}
```

The code in the trigger runs and loops through a SQL statement, which selects all Patient IDs affected by the change in a single Doctor record. On each Patient ID, the %DeepSee.Utils:%SetDSTimeIndex method is run. This method sets an entry in the synchronization global ^OBJ.DSTIME for Patients, flagging the appropriate Patient records for synchronization.
Note that the trigger runs on UPDATE and DELETE events. We are not concerned about INSERT events because there should be no Patients records yet pointing to newly created Doctors. The Foreach option allows the trigger to fire when a record is changed using both SQL and objects.

Testing the trigger approach

Let's see what happens when we edit another Doctor record:

```
SAMPLES>Set doc = ##class(DeepSee.Study.Doctor).%OpenId(33)
SAMPLES>Write doc.DoctorGroup
II
SAMPLES>Set doc.DoctorGroup="III"
SAMPLES>Do doc.%Save()
SAMPLES>zw ^OBJ.DSTIME
^OBJ.DSTIME("DeepSee.Study.Patient",0,84)=0
^OBJ.DSTIME("DeepSee.Study.Patient",0,97)=0
….
^OBJ.DSTIME("DeepSee.Study.Patient",0,987)=0
```

After changing DoctorGroup from "II" to "I" for the Doctor instance with ID=33, the trigger computed all Patient IDs that refer to that Doctor instance. The trigger uses the %DeepSee.Utils:%SetDSTimeIndex method to add an entry in ^OBJ.DSTIME("DeepSee.Study.Patient") for each Patient ID.

Now DeepSee synchronization updates the PATIENTS facts whose doctor instance has been changed:

```
SAMPLES>Set rs=##class(%DeepSee.ResultSet).%ExecuteDirect(MDX)  Do rs.%Print()
Result:                 322
SAMPLES>Do ##class(%DeepSee.Utils).%SynchronizeCube("PATIENTS")
32 fact(s) updated
SAMPLES>Set rs=##class(%DeepSee.ResultSet).%ExecuteDirect(MDX)  Do rs.%Print()
Result:                 290
```

As you can see, 32 Patients facts got updated and the result returned by the MDX query changes accordingly after synchronization.

Two questions to the readers:

1. Would it be acceptable to run %ProcessFact instead of %SynchronizeCube in the trigger?
2. Does this approach have any limitation? For example, if our cubes had much more facts and Doctor records were subject to frequent changes, would the trigger approach still be acceptable?

#Analytics #Object Data Model #InterSystems IRIS Analytics (DeepSee)

Source URL: https://community.intersystems.com/post/triggering-deepsee-synchronization-when-source-class-properties-reference-different-class