

Adventures in Groovy – Part 18: Real Time Data Movement (Setting The Stage)

Introduction

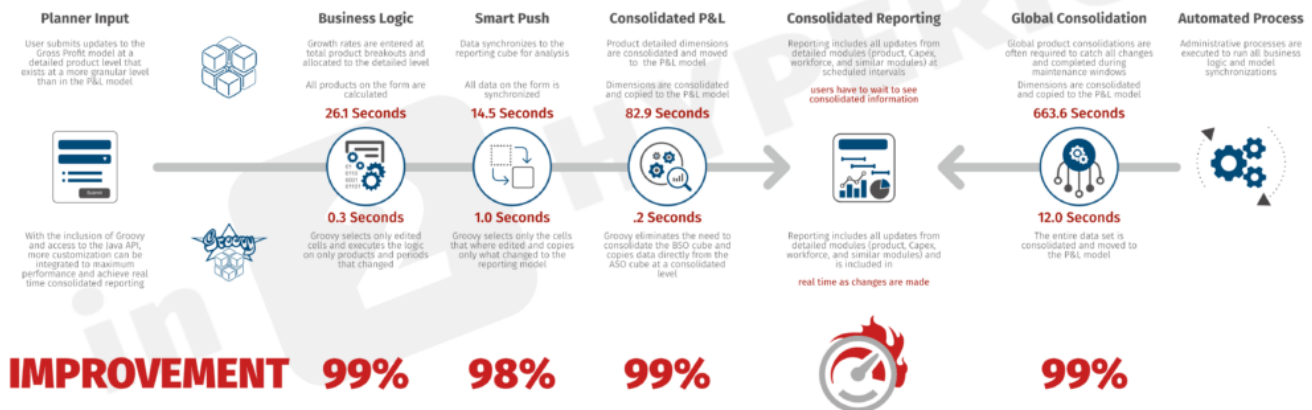
One of the challenges with Hyperion Planning is the ability to move data between applications in real time. A classic example of this is a P&L application with other modules that have greater detail. The following is an example.

- A Gross Profit specific database that includes a product, delivery channel, and product type dimension.
- A CapEx specific database with asset type, asset, and asset category
- A Workforce specific database with job type, union, and employee.
- A P&L application that includes income and expense with information fed from the detailed models at consolidated levels.

In June of 17, with the release of Groovy Calculations, the ability to update any of the detailed models and synchronize the consolidated data in real time to the P&L database became possible. When a user saves data, within seconds, the data can be reflected in a database with different dimensional.


Integrating Groovy Functions Into Calculations Produces Ground Breaking Performance Results

the following are actual results in a production application



Setting The Stage

This is going to be a lengthy, multi part article. Before we begin, the application architecture is going to be laid out so the calculations can be explained in detail. The application will consist of 2 play types. The first is the P&L and the second is a detailed product planning play type. We won't introduce a Capex and Workforce model. It will only complicate the explanation and is redundant in the logic required.

The data flow and architecture looks like this.  GP (Gross Profit Product Detail) databases

The initial plan type is called GP

Although this may not match with your model, the concept is the same.

- It has dimensions that are required to plan at a product level that don't exist in the P&L application.
- It has specific logic that doesn't apply to other databases.
- It has a unique account dimension that doesn't mirror what is in the other applications.

- Consolidation takes a long time and is not optimal to be performed on a data form save.

As previously stated, the same differences will exist in other models, like Capex and Workforce.

Fin (Income Statement / Balance Sheet) databases

The Fin application is a typical consolidated reporting application that excludes details like product level revenue, employee level plans, and assets and their properties needed to calculate capital expense.

Dimensional Summary

For this example, the following shows the application dimensions and database associations

Dimension	GP (BSO)	rGP (ASO)	Fin (BSO)	rFin (ASO)
Years	✓	✓	✓	✓
Period	✓	✓	✓	✓
Scenario	✓	✓	✓	✓
Version	✓	✓	✓	✓
Entity (Company)	✓	✓	✓	✓
Account	✓	✓	✓	✓
Channel	✓	✓		
Material Group	✓	✓		
Vendor	✓	✓		
View		✓		✓

The Synchronization Process

The GP database includes 3 dimensions that don't exist in the Fin model. For this to be moved to the Fin model, 3 dimensions need to be consolidated. The GP model also has a different account structure. A translation between the two

account structures has to occur before the synchronization can be completed. The other piece that is not required, but highly encouraged, is to only work with the data that has changed. So, this will dynamically select the data rows on the form that have been edited by the user. Functionally, the following happens when a user saves a data form.

- Identify the members that need to be included in the synchronization
- Push the level zero data from the GP BSO database to the GP ASO database (only edited data)
- Retrieve the data from the GP ASO database at a total product, channel, and material group
- Submit the data from the above retrieve to the Fin BSO application and the rFin ASO application
- Execute any logic that needs to be completed in the Fin application (taxes, driver-based data, etc.)
- Push the level zero data from the Fin BSO database to the Fin ASO database

Groovy Methods Required

There is a lot going on here, so we are going to summarize and explain the Groovy methods that will be used to accomplish the synchronization.

DataGridIterator

To make this as efficient as possible, it is important to only execute the methods on the data that have been edited. If you haven't read Part 3 of this series, take a look before you continue.

DataMap / SmartPush

Once the POV is identified that needs to be included in the synchronization, the first operation is to push that data to the reporting cube. This will be used a couple of times in this sequence. Part 8 of the Groovy Series covers this in

detail and an understanding is helpful before you continue.

DataGridBuilder / DataGridDefinitionBuilder

This has not been covered yet. These methods give you complete control to simulate a retrieve and submit. These two objects are the major pieces of the puzzle that have never really been exposed in any fashion. These are the methods that really open up the possibilities for real time reporting.

Take A Breath

You may be a little overloaded with new information. We will let this settle in and give you a chance to digest the concepts. The next article will walk you through the code. To satisfy your curiosity, watch this video, which takes you through the above example in a live environment.