Open source code samples:

- **Trigger program SYNSTATE**
- **Trigger program SYNCUST**

Inuendo is not implemented overnight when existing applications are involved. It must be gradually introduced and retrofitted responsibly. A major component of that process is the series of trigger programs which replicate legacy data into Inuendo virtual tables on a real time basis.

SYNSTATE:

The legacy ERP has a file called STATE that contains a state code and a description. It came pre-loaded with the software and rarely if ever changes. That makes like easier. Since the list of states will be common to all business units, we'll make the STATE class subordinate to the FOUNDER. No properties will necessary for the STATE class because the code can be stored in ENTHEAD as the LEGACYA value while the name can be stored in the DESCRIPTOR value.

The trigger handles inserts, deletes and updates. Note that the newEntity function used by SYNSTATE will test to see if the entity exists already (based on its LEGACYA value), and if it does, will update the DESCRIPTOR.

In order to force an initial load, we simply performed an SQL with a phantom UPDATE (set a field equal to itself). On rare occasions when this file is updated by the legacy ERP, this trigger will kick in, otherwise it will remain dormant.

SYNCUST

This is a different animal because the legacy Customer Master (CUSMS) is frequently updated through maintenance. It also has many “buckets” which are incremented when various business transactions are consummated. Of course, Inuendo’s principles prohibit cumulative buckets. Therefore we will need to introduce “transaction synthesis” to the equation. More on that later, however.

Identity:

Each company will have its own set of customers, therefore the CUSTOMER class will be subordinate to a COMPANY class. Therefore, whenever this trigger fires, we must create a COMPANY entity with the legacy company number if it does not exist already.

If you look at the mapping worksheet, you’ll notice that various addresses and phone numbers, which are made up of multiple fields, have been organized into their own classes. So we will need to instantiate or update entities of those classes while we’re busy replicating data to a CUSTOMER entity.

Links:

In addition, many of the fields contained in the legacy layout have a limited number of possible values. For example, codes, categories and identifiers. While a few of these fields match keys in other tables, the majority exist only in a flat, System/36 style catchall code file. Obviously this is undesirable because the code file cannot be used effectively by SQL due to its hundreds of record layouts. However, we have no plans on converting all these “reference tables” (and I’m being generous here). So in order to replicate these relationships, we will need to create a class for each, such that an entity of that class can be instantiated “on the fly” with the associated legacy value. That entity ID will become the value of a LINK property for the CUSTOMER entity.

Transaction synthesis:

As we mentioned earlier, the CUSMS file is replete with buckets. For example:

- Month to date
- Year to date
- Last year
- All time
- All time prior to this month

Inuendo compliant applications use business rules on transaction entities to produce equivalent results. Each transaction entity records a “delta” – the difference between the before and after image of a legacy bucket. That’s great going forward, but what about previous activity ? Here’s how we’ll handle it…the FIRST time a customer is replicated:

- The BEFORE image of a month-to-date value will become a transaction dated on the first day of the month.
- The BEFORE image of a year-to-date value, less the BEFORE image of the associated month-to-date value, will become a transaction dated on the first day of the year. In layman’s terms, if it’s June, then this value represents the activity from January through May.
- The BEFORE image of a previous year value will become a transaction dated on the first day of the previous year. Since previous year buckets don’t change once they are initialized by year end procedures, the AFTER image could also be used, but we’ll use the BEFORE image for consistency sake.
- The BEFORE image of an all-time history value will become a transaction with an epoch date. On an IBM i system, this could be *LOVAL. Other platforms may use dates such as 1/1/1970 or some Julian date from the 16th century. Some all-time buckets exclude current month activity, however that is handled by the legacy ERP and does not affect our strategy.

In our sample ENTPROP table, we defined a class called INCREMENT for our transactions. We can construct a business rule that performs an SQL sum over the AMOUNT or QUANTITY property (remember, these are represented by GET functions) for a specified entity ID where the NICKNAME property matches an argument and the EFFECTIVE property is between a specified pair of inclusive date arguments. This function could be made Moment sensitive by adding an optional Moment argument and only considering INCREMENT entities that existed within the date range.

In the near future, such a function may be included in the open source download. It would require compliance with the INCREMENT class definition, however.