Part II

Integration Use Cases

Achieving “One Version of the Truth” requires integration between the data synchronization application environment (especially the local trade item catalog) and enterprise applications that create and maintain product data. The primary context for understanding how this integration works is the product lifecycle. As business events occur – new products are introduced, product-packaging changes, new pricing terms are negotiated – data synchronization activities must occur to support them. For example, introduction of a packaging change requires that the packaging description in the local catalog be updated from some back-end system of record (application integration), and that customers who carry the associated product be notified of the change (data synchronization).

Implementing integration for data synchronization requires an understanding of the product lifecycle process context – chiefly, which business events drive data synchronization content changes and notifications, and what process steps must be executed when each event type occurs. It also requires a thorough understanding of the data in both product data management and data synchronization systems of record, to ensure that data transformations and validity rules are applied correctly in each context.

The figures in this section illustrate the most important integration use cases for data synchronization:

- Initially loading enterprise application data into the local trade item catalog
- Periodically updating the local catalog with accumulated application system changes
- Triggering an immediate local catalog update when an application data change occurs
- Triggering application updates and business processes when synchronization events occur

In each case, supply-side companies can choose from a variety of integration implementation approaches. For example, a business professional could manually re-key information from existing applications into the local catalog to implement the initial data load. Or s/he could write program code to extract, transform, and validate data from each source application and update the local catalog. Or s/he could employ general-purpose integration technology – the approach we will examine in this document. The key insights here are (1) you must address integration somehow, and (2) the approach you take to integration will determine, to a significant degree, the costs and benefits you realize from your data synchronization investment.

Figure 7 shows the main components of an integration process that loads data from back-end systems of record into the local trade item catalog. For now, we’ll focus on the functional requirements for this and the other integration use cases, rather than the implementation details.

The main components of the initial catalog load process are:

1. Manual process launch: The timing of the initial load process is determined primarily by the readiness of the source data, not by a business event or schedule. So all we need in this case is a way to launch the load process manually.

2. Load process script: There are just two steps in our load process use case: (1) extracting / transforming / loading data from a system of record into a staging database; and (2) notifying a business user when the process completes. The purpose of the staging
database is to provide a “halfway house” for source data, so that data from multiple systems of record can be aggregated and reconciled before it is imported into the local catalog.

3. Transformation rules: The focus of the initial catalog load process is a transformation step that reads source data from a system of record, transforms it to comply with local integrity and validity rules, and updates the staging database.

4. Source and target “schema” definitions: In order to interpret application data and produce appropriate result data in the staging database, the transformation step requires “schemas” (document definitions) that describe the content and type information for both sources and targets. Ideally, schemas can be generated from existing metadata, as in this case, from the source and target DBMS catalogs.

5. Resource adapters: Adapters are general-purpose interfaces to persistent data, communications services, database management services, and other resources needed by integration processes. In the initial load process, database or file adapters provide access to source and target databases for the transformation step.

6. Import utility: Upon receiving the email notification that the staging database has been populated by the initial load process, a business user or system administrator invokes an import utility that validates the data and, if valid, loads it into the local catalog.

Figure 8 shows a process that looks very similar to Figure 7, but implements a different function, namely, updating the catalog periodically as source data changes occur in back-end systems of record.
Implementing a catalog update process is important in any supply-side company where changes occur in data synchronization source data. Examples of such changes include:

- Introduction of new products (including seasonal introductions)
- Changes to product attributes (colors, styles, dimensions, weights, etc.)
- Changes to product packaging (including case and pallet configurations)
- Changes to product pricing (including customer-specific terms)
- Withdrawal of obsolete products

Once data that describes such changes is captured in a system of record, corresponding data in the local data synchronization catalog must also change, and, from there, the changes must be synchronized externally with customers. Figure 8 illustrates the components of a process that automates this kind of change process, based on periodic extraction of change data from back-end systems of record.

The main differences between this process and the initial data load process shown in Figure 7 are:

1. Scheduled process launch: On the assumption that product data changes continually in back-end applications, the update process runs on a cyclic schedule. The update frequency may
depend on source data volatility, batch window availability, seasonal product introduction
cycles, or other factors.

2. Adapter configuration: Although the data transformation step is essentially identical to the
one depicted in Figure 7, the source adapter in this case must be configured to access only
data that has changed since the last time the update process ran. This can be accomplished by
either (1) querying or filtering source data using time stamp criteria; or (2) periodically
creating a flat file extract that contains only changed data.

Whether the update process also automates the external synchronization of changes with

*Figure 9: Triggered catalog updating in near-real time*

...customers depends on the business process management policies implemented by the supplier. In
some cases, approval of the external synchronization step may occur prior to exporting change
data to the update process, so the need for a separate synchronization approval step is absent. In
other cases, a separate synchronization approval step may be desirable to ensure that any
exceptions raised by the update are reviewed, prior to synchronization. Note that, due to the
similarity between our initial catalog load process (Figure 7) and catalog update process (Figure
8), there are opportunities here to reuse integration components – transformation rules and
schema definitions, for example.

Figure 9 shows an alternative catalog update mechanism, using triggers to automate the detection
of changes and invocation of the update process in near-real time. The term “near-real time”
means that source data updates and triggered catalog updates occur asynchronously, but with the
appearance of simultaneity. The advantage of this approach is that local catalog data remains
consistent with source data in back-end systems of record, on a continual basis.

The main differences between this process and the processes examined above are:

1. Triggered process launch: In order to maintain near-real time consistency between
   application source data and local catalog data, neither manual nor scheduled process
   launching suffices. Instead, processes are launched automatically by an event monitor, which
   is notified by a database stored procedure that triggers on application data changes.

2. Process selection: Based on notification data passed by the stored procedure, the event
   monitor launches an update process that is designed to transform and update the type of data
   maintained by a given system of record.
3. Transformation granularity: Each process instance transforms a single input data instance, rather than a batch of source data changes.

One final difference between the triggered and periodic update processes is that immediate synchronization of changes through the data pool is not practical, in the triggered case. The reason is that synchronization latency – the time it takes for the data pool to process synchronization commands and return responses – is unpredictable. Rather than flood the data pool with a high volume of triggered synchronization requests, it makes more sense to queue synchronization requests locally, and process them on demand, or on a cyclical schedule.

**Figure 10: Triggering application updates from synchronization events**

Figure 10 shows the final application integration use case, in which the processing of inbound customer responses retrieved from the data pool triggers changes to back end application data. When a customer confirms interest in an item you have published, for example, the receipt of that confirmation is an important business event. Although it falls short of an order transaction, it is potentially a buy signal, and should trigger appropriate human and automated responses.

Exactly what business response is appropriate will depend on the policies and processes established by each supplier. Perhaps we might want to update data about the customer in our CRM data base, or update an order management system to allow orders for that product to be accepted from the customer, or notify a salesperson or customer service agent. Or all of the above. At the process automation level, in addition to transforming inbound responses and related catalog data into application inputs and notifying relevant business professionals, we might want to launch application programs or scripts, to exploit existing business logic.

The main differences between this process and the processes examined earlier are:

1. Trigger source: Instead of triggering based on update events in back-end application databases, this process triggers on an inbound communication event.

2. Transformation direction: Instead of transforming application source data into equivalent catalog content, this process transforms inbound confirmation data (and possibly related data from the local catalog) into corresponding application update content.
As with the other use cases examined above, the same integration components and reuse principles apply in this case, as well.

Choosing an Integration Strategy

As observed earlier, integration between the data synchronization environment and enterprise applications is fundamental to achieving data consistency – “One Version of the Truth” – and realizing full benefit from your data synchronization investment. The question is not whether, but how you implement this integration in your company.

Here are some questions to consider before you choose an integration implementation approach:

- Data volume: Is the number of items to be entered, maintained, and synchronized manageable without automation?
- Data quality: Do you need to automate interfaces to ensure that the data you need to synchronize is correct and consistent with internal systems, or do you have the necessary procedures and checks in place to do this manually?
- Process latency: Is the need for visible, rapid response to key business events sufficient to justify automation?
- Training and expertise: For data that is entered manually, do your business professionals understand the source application data and its relationships to data synchronization attributes? For programmed integration, do you have the necessary trained staff to implement all required process and application interfaces, and maintain them, as changes occur?
- Maintainability: For programmed integration, how will you maintain your code when changes occur to business processes, source and target database schemas, data integrity rules, commercial and in-house applications, runtime platforms, etc.?
- Environment dependence: For programmed integration, how will you manage the impact if hardware, operating systems, DBMS, communications, or other factors change?
- Business goals: To what degree is your integration strategy driven by corporate goals to reduce costs, improve customer service, improve quality, or gain competitive advantage? Which business goals and objectives are achievable without automated integration, and which are best realized through a systematic integration approach?

Choosing a Data Synchronization Solution [can be presented as a sidebar with checkboxes]

What first appears to be a very simple idea – synchronizing product, location, pricing, and other trade information with customers – is actually a set of core business processes that nest between systems that maintain enterprise data and systems that use that data in electronic commerce and other e-collaboration applications. Although the simplicity of file transfer and spreadsheet-based offerings may be enticing, the reality is that integration is the key to unlocking the full business benefits of data synchronization.

Here’s a list of capabilities to consider as a starting point, when choosing a data synchronization solution for your company:

Local Trade Item Catalog:

- Full local catalog schema support for EAN.UCC data synchronization message standard
- Ability to manage and operate on large numbers of trade items with good response times
- Extensible to support maintenance of company-standard product identifiers on trade items
- Extensible to support additional customer-defined attributes
- Support for multiple test and production catalogs
- Support for migration of data from test to production
- Support for source data loading from multiple data sources
- Validating importer for loading data from external sources

**Catalog Management:**
- Access controls for all application functions and local catalog contents
- Concurrent, multi-user application support
- Role-based restrictions on data access and modification (e.g., based on data ownership)
- Complete user interface support for local entry, validation, viewing, and editing of trade item, location, pricing, and other data
- Interactive controls for registering, cataloging, publishing, and other data synchronization functions
- Supervisory approval interface for all synchronization activities
- Business-oriented user interface with visual cues for process sequence, data completeness and validity
- Configurable user interface labels for customized terminology
- Global constraints to prevent modification of local catalog attributes maintained in other systems of record
- Instrumentation for notification and automated response to catalog management events

**Synchronization Management:**
- Certified compliance with current UCCnet Synchronization Services release
- Black box automation of all synchronization details (marshalling, communications, retry, error reporting, etc.)
- Configurable synchronization frequency
- Instrumentation for notification and automated response to synchronization events

**Integration Support:**
- Specification of complete integration processes without programming
- Automation of conditional, multi-step business processes
- Scheduler for automated process invocation based on calendar or elapsed time
- Event manager for automated process invocation based on distributed event notifications
- Ability to transform data from any source syntax (file, database, XML, etc.) to any target syntax
- Ability to access data remotely, on other computers and operating system platforms
- Email notification support
- Ability to invoke external applications and scripts / bat files
- Platform-independent architecture to preserve integration requirements.
- Metadata importers for generation of source and target document (“schema”) definitions
- Offline integration modeling and testing.

###

EXTOL International, Inc.
795 Franklin Avenue
Franklin Lakes, NJ 07417
(201) 847-1200
www.extol.com