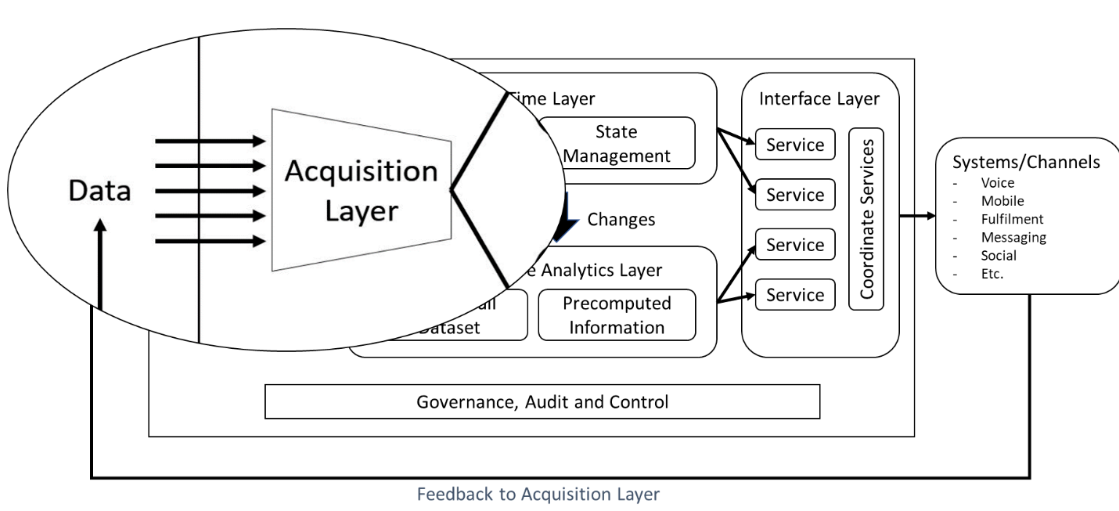
**Ab Initio Lambda - Acquisition Layer**

n the previous article [(click here)](https://www.linkedin.com/pulse/ab-initio-lambda-overview-chris-day-/) we described the Ab Initio Lambda Architecture at a high-level.

In this article, we go into the detail of the Acquisition Layer.



In the original Lambda Architecture, it implies that the Acquisition Layer is a component of both the Speed and Batch Layers.

Nathan had 8 fundamental principles to adhere to;

Robustness and fault tolerance, Extensibility, Low latency reads and updates, Scalability, Generalization, Minimal Maintenance, Debuggability & Ad-hoc Queries

We need an Acquisition framework that adheres to all of those principles, and we have one now;

## Acquisition means Acquire>It

Today we use Ab Initio's Acquire>It extensible framework to serve any data, at any time, to both the Real-Time and Batch Layers. Acquire>It also adheres to Nathan’s 8 fundamental principles too.

Built upon global patterns for data acquisition at scale and fully configurable, you'll find an order of magnitude increase in productivity and reliability. That’s the benefit of the underlying graph model – metadata drive and parallel from the get-go.

Acquire>It is an Express>It-based application that enables you to read data from a single source dataset in almost any format, process it, and then deliver the resulting data to multiple, independent target datasets.

### **Previous approaches to acquisition followed either;**

1. Writing a graph for each feed
2. Taking time to find the patterns and producing a generic graph, then a parameter set for each feed

Ultimately both approaches can lead to underperforming processing. Multiple graphs increase the administrative burden, and generic graphs often become complicated and unmaintainable.

The outcome is a shed load of technical debt and mounting problems that we sweep under the carpet for the next iteration or SI.

### **Now there's a new approach using Acquire>It**

Acquire>it gives you efficacy with a flexible and efficient framework for the Acquisition Layer out of the box without breaking pipeline parallelism.

### Acquire>It is composed of 3 layers;

* A generic framework
* A library of generic modules
* And last but not least - Metadata

### An Acquire>It feed is configured opposed to writing a graph by specifying;

* The modules a feed will use
* The metadata that drives those modules; Record Format, Key, Mappings, Sources, Targets, etc.

### The critical point is that the level of reuse moves from a graph to a module which drives the following;

* Minimise complex conditionality
* Minimise re-test from upgrades
* Enable extensibility without future upgrade re-testing costs
* Minimal Maintenance
* Reduction in technical debt

### Extensibility

* You can write modules for Input, Output and Processing
* All modules included at runtime in-flow; No I/O cost, no extra scheduling cost, single phase.
* Low-latency

Acquire>It is designed to be extensible by customers to meet their specific and possibly unique needs. I can’t stress enough that engagement with Ab Initio for module extensibility is vital. You need a sounding boarding, and that’s a call to Ab Initio’s internal consulting. Period.

### Simplicity and Clarity

Within the Mapping modules we use business rules, a spreadsheet-like interface to create and test expressions that describe processing logic to perform. The organisation of these expressions is within rules and rulesets. For many Acquire>It configurations, this is the only metadata that an analyst needs to provide.

The use of rules and rulesets can benefit an organization in the following ways:

* Transparency — Rulesets present business rules in a format that technical and nontechnical people can understand.
* Turnaround time — Business analysts can write and test rules without assistance from developers, increasing the speed and agility of rule development.
* Traceability — Ruleset tests show not only the computed result but also how the rule logic applies to each record.

### Modular Flexibility

•       Each Module is atomic within the framework

–       For a given feed we can reorder Modules

–       Remove Module version impact on existing feeds

–       Testing a Module does not impact

•       Limits regression testing requirements

•       Minimize conditionality –

–       Only the functionality you need is present in the final graph

–       Much simpler to understand and debug if required

### Metadata management

* Fully integrated with the Metadata>Hub
* All data in and out of Acquire>It is configured via Datasets
* Dataset is the definition of ‘some data, somewhere’; Controlled by a data steward, Can be made read-only for feed configurators
* Current Dataset types include: File, Queue, Hive, Database, HDFS, Excel

### As of today, Acquire>It consists of the following Pre-Built Modules

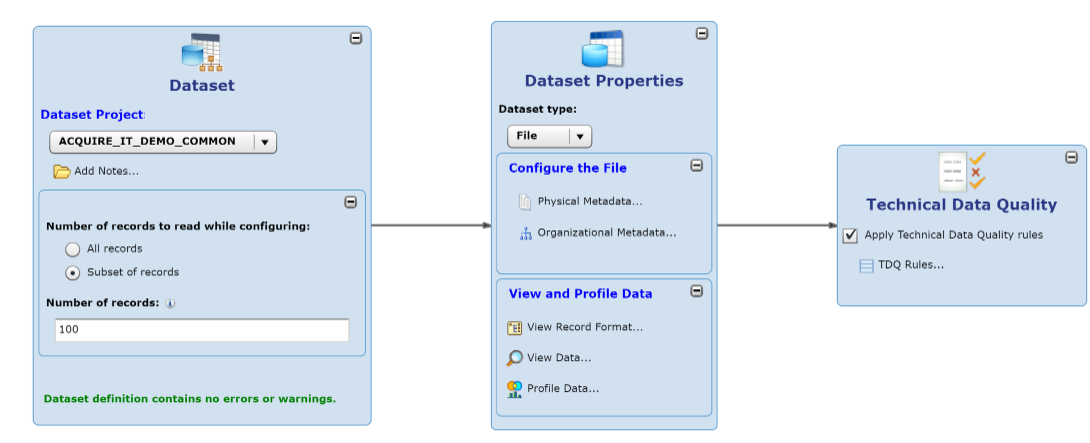
Processing Modules; Filter, Enrich, Join, Replicate, De-duplicate, Rollup, Normalize, Hierarchical Mapping

* Change Data Capture Modules; Delta, Snapshot, Transactional
* Validation Modules; Business Data Quality, Technical Data Quality

## What does this look like?

### **Dataset Configuration**

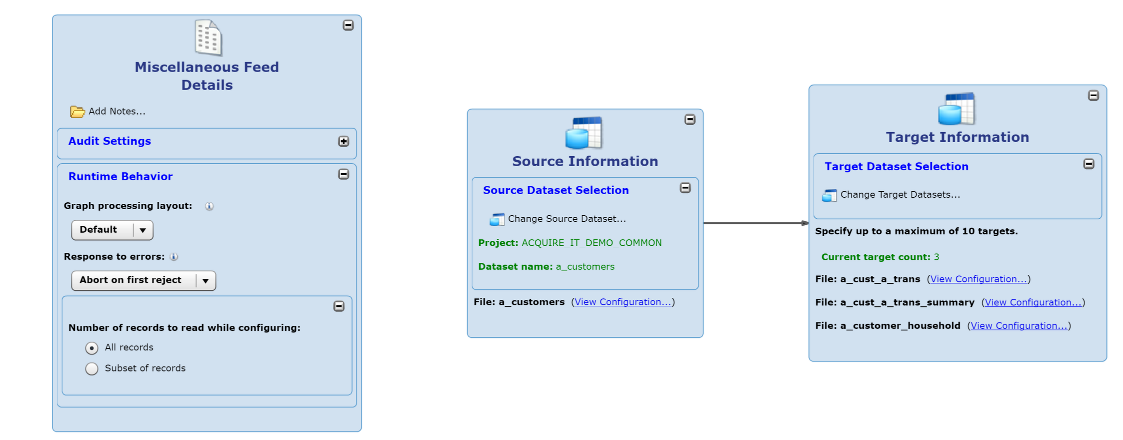
A Dataset configuration adheres to Nathan’s principle for Generalization with a clear and consistent interface. Of note, is putting the Technical Data Quality next to the data, keeping in line with the policy of only processing good data – why waste CPU cycles on the bad?



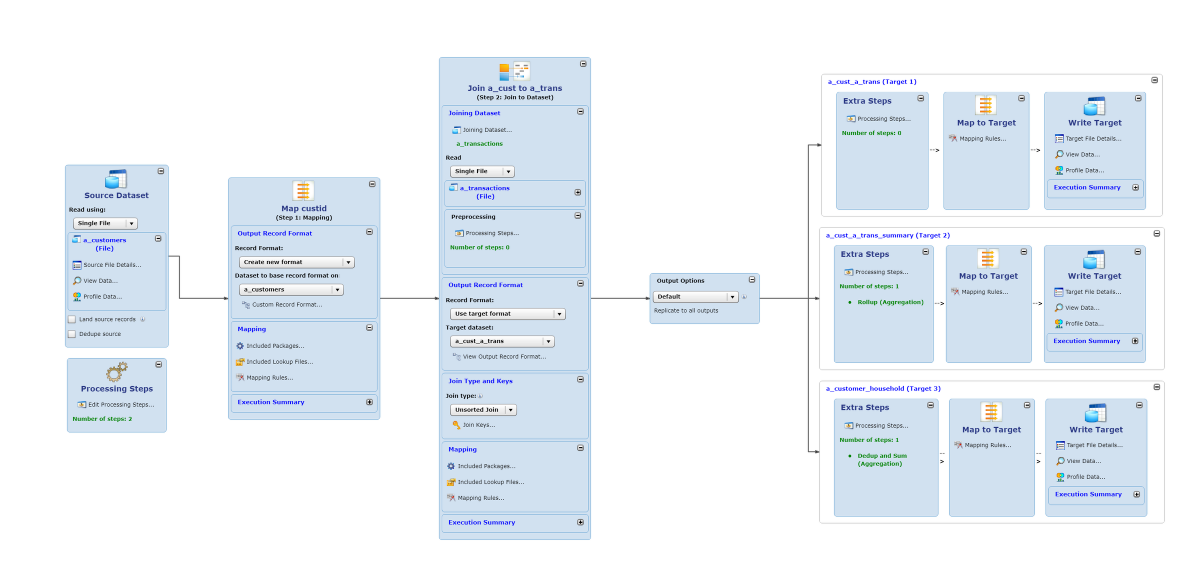
### **Feed Configuration**

A Feed configuration shows support for all of Nathan’s principles.

At a summary level, we see the source and targets along with Audit and Control information.



At a detail level, we see our modular configuration;



## **Want more on Acquire>It ?**

If you want a full featured demo, then engage with your Ab Initio Account Manager. If you are in a knowledge repression culture, then drop me a line and I’ll help you connect.

**Footnote on extensibility**

More often than not, the very mention of customer-driven extensibility widens the eyes of disabling consulting companies. The rationale is that Extensibility means mass customisation and equates to more consulting hours, more technical debt. Wasn’t that the very problem we are trying to address?