AB INITIO SOFTWARE

One System, Many Applications

AB INITIO SOFTWARE

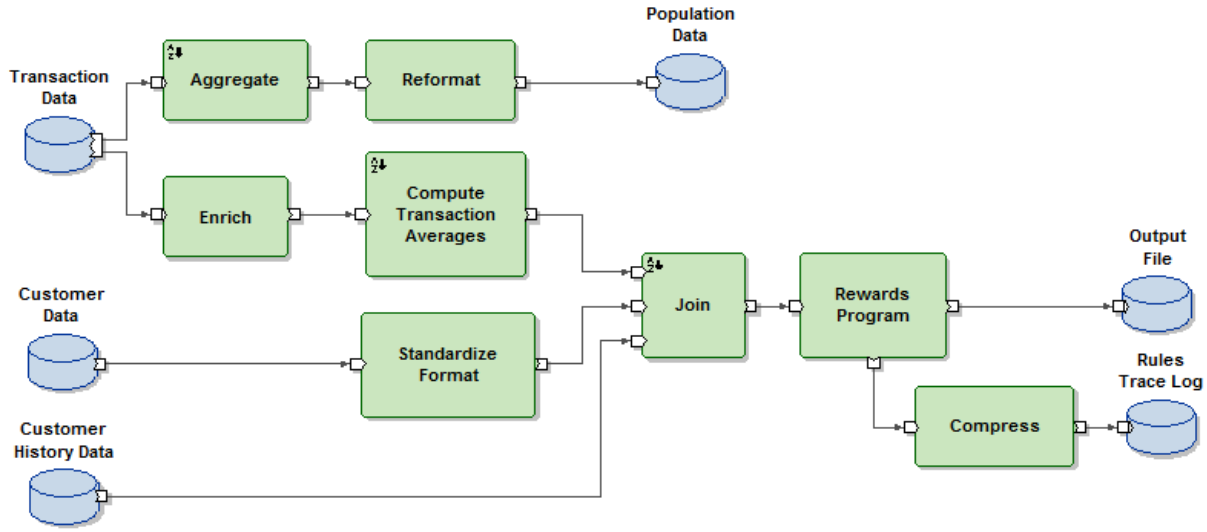
One System, Many Applications

THE CO>OPERATING SYSTEM: Broad and Deep

The Co>Operating System is an environment for building, integrating, and running enterprise business applications. The Co>Operating System is the foundation for all of Ab Initio's technologies, including the Enterprise Meta>Environment, Continuous>Flows, Conduct>It, the Business Rules Environment, and more. These technologies provide a complete and seamless application development and execution environment.

The heart of the Co>Operating System is a "dataflow engine." This engine drives a large library of data processing "components" that manipulate the data flowing through an application. Applications are designed, implemented, and maintained graphically through Ab Initio's Graphical Development Environment (GDE).

The core principle of the Co>Operating System is that applications are designed and developed in the way most people would design a system on a whiteboard (or even on a napkin). Easily recognizable icons are used to represent the input and output sources, which are then combined with processing boxes and arrows to define the overall processing flow. By selecting the appropriate components from an extensive library and "wiring them up," you create an Ab Initio application.



Ab Initio seamlessly integrates the design and execution of applications: the drawing *is* the application. And the resulting application can be batch, near real-time, or real-time in nature, or even a combination of all of these — all united into one consistent and powerful computing environment.

The graphical dataflow approach means that Ab Initio can be used to build the vast majority of business applications – from operational systems, distributed application integration, and complex event processing to data warehousing and data quality management systems. But graphical design and development addresses just one part of the challenge. These applications must also meet significant operational and management requirements.

Historically, graphical programming technologies yielded pretty pictures that fell apart when it came to addressing these real-world requirements. The Co>Operating System is a genuinely different beast — it actually works. Here are some sample deployments and their underlying requirements:

- One of the world's largest stock exchanges converted millions of lines of Cobol code for mission-critical operations into Ab Initio applications. The solution is now a major part of the trade processing pipeline. It connects to the real-time trading bus and processes transactions at over 500,000 messages per second.
- One of the world's largest retailers receives market-basket data in real-time from the cash registers in thousands of stores for inventory control and fraud detection.
- One of the world's largest phone companies processes call detail information for call rating, data usage tracking, and network monitoring. Many billions of call-detail records per day are processed, as are millions of usage queries.
- One of the world's largest chip manufacturers pulls manufacturing quality information in real-time from the fabrication line to increase yields.
- One of the world's largest credit card networks uses Ab Initio as its data backbone, processing and passing all transactions to back-end systems in batch and real-time. They accumulate 1 petabyte of transaction data per year in an Ab Initio data retention system that supports customer service call centers with sub-second response times to queries.
- One of the world's largest Internet companies processes tens of billions of advertising impressions per day for billing and better ad placement.
- One of the world's largest insurance companies does many parts of its claims processing with Ab Initio. This company's re-insurance and treaty processing system contains many tens of thousands of rules, all implemented in Ab Initio.
- One of the world's largest package delivery companies generates all its invoices and computes sales compensation for its account teams using Ab Initio applications.
- One of the world's largest banks consolidates information about all their customers from all lines of business into an enormous data warehouse built with Ab Initio software. The same company uses Ab Initio to customize and process all the SWIFT transaction traffic between its international subsidiaries.

You might have noticed that all these examples say "one of the world's largest." Why have all these major corporations chosen Ab Initio? Because not only is Ab Initio software intuitive and easy to use, it also stands up to the most complex application logic and to huge amounts of data. And it does this with high performance and remarkable robustness. This combination is unique.

These are just some examples of specific deployments at these customers, and these deployments tend to be very broad. Meeting the requirements of all these applications requires many capabilities, including:

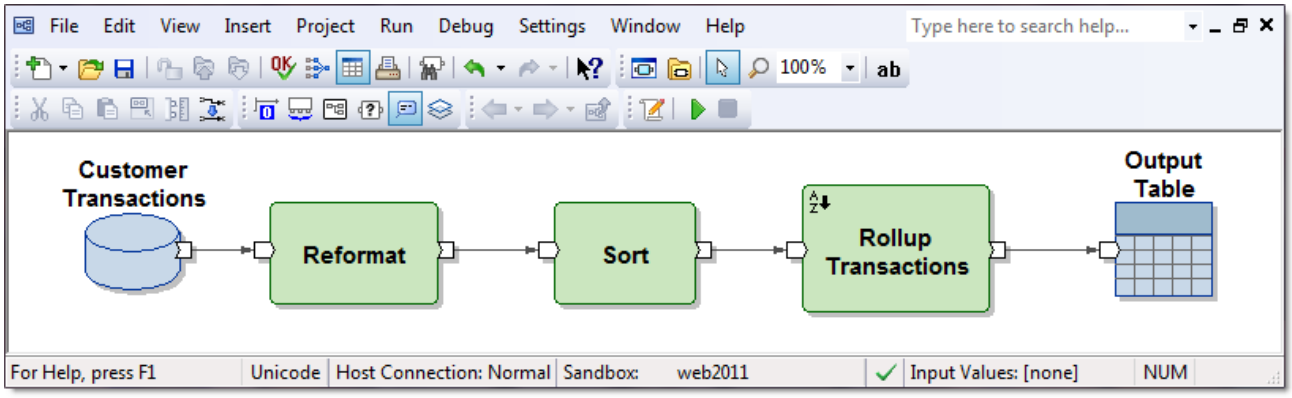
- The ability to easily express large amounts of complex logic graphically.
- Connectivity to practically anything: a wide range of databases, queuing systems, flat files, ERP systems, standard messaging protocols, and more.
- Native support for complex data structures – any kind of data from anywhere. Hierarchical (XML and legacy), international, big objects, variable length, bit-packed data, and more.
- Support for a wide range of operating systems (Unix, Linux, Windows, mainframes) and distributed processing across these platforms.
- Open architecture for rapid integration with legacy and 3rd party applications, data, and environments.
- High efficiency and high scalability, along with the ability to enable "cloud"-like computing across networks of servers.
- High-performance batch and real-time processing (web services and service-oriented architectures (SOA), as well as streaming).
- High degree of reusability of both large and small portions of applications, including business rules.
- High resilience against data issues and system failures. Strong user control over failure response.
- Comprehensive management capabilities. Software development lifecycle support, including version control and release promotion. Execution scheduling, monitoring, and alerting. Application analysis, and more.

The only way to meet these requirements is to have an architecture designed from the beginning to meet them all — at the same time. Once an architecture has been set, it is practically impossible to add fundamental capabilities. The Co>Operating System was designed from the beginning to have all these capabilities. It is a robust, proven technology that is successfully used for a very wide range of complex data processing applications.

WHAT *IS* AN AB INITIO APPLICATION?

The core of an Ab Initio application is a dataflow graph, or "graph" for short. A graph consists of components that are connected together through data "flows."

For example, the following graph is a simple application that reads each record from a flat file containing customer transactions, and then reformats the data (by applying rules) before sorting it and then rolling up (or aggregating) the data. The results of this processing are then written out to a table in a database, such as Oracle.



Through the use of highly configurable components, the Ab Initio Co>Operating System provides all the fundamental building blocks for business data processing, including:

- Data transformations
- Selection/Filtering
- De-duplication
- Joining/Merging
- Normalizing/Denormalizing
- Compressing/Uncompressing
- Aggregating/Scanning
- Sorting
- Data generation and validation
- ... and more

In addition, the Co>Operating System comes with a large library of built-in components for dealing with virtually every type of data source or target.

This library includes components for:

- Connecting to any kind of flat file on Unix, Windows, and mainframes
- Connecting to all common (and many less common) databases (Oracle, DB2, Teradata, SQL Server, Netezza, Greenplum, mainframe DB2, IMS, IDMS, Adabas, ...)
- Connecting to all standard message queuing infrastructures (IBM MQ, JMS, Microsoft MQ)
- Connecting to web service infrastructures (WebSphere, WebLogic, IIS, Apache/Tomcat, ...)
- Connecting to many 3rd party products (SAP, Siebel, SAS, PeopleSoft, Salesforce.com, ...)
- Parsing and manipulation of hierarchical data structures (XML, ASN.1, Cobol, ...)
- Parsing, manipulation, and generation of domain-specific data formats (SWIFT, FIX, EDIFACT, ...)
- Content-based message routing
- Metadata connectivity to a wide range of data definition and business intelligence products (ERwin, ERStudio, MicroStrategy, Business Objects, Cognos, ...)

Small applications may have 3 to 5 components. Large applications may have a hundred components. Very large applications may have many thousands of components. Applications can also consist of many graphs, each of which may have many components.

Throughout the spectrum – from the smallest applications to the largest — Ab Initio applications exploit reusable rules and components, enabling rapid response to changing business needs.

BUSINESS LOGIC IS SPECIFIED COMPLETELY GRAPHICALLY

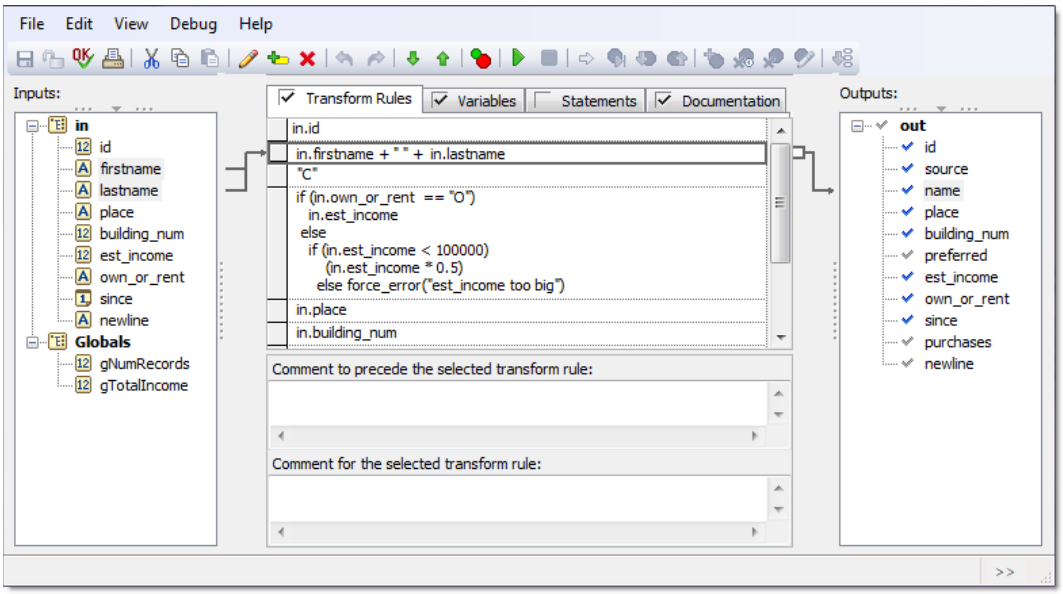
Below the component level, the Co>Operating System can apply practically any type of user-specified rules and logic to any type of data. And, those rules are specified in a graphical manner. This is a significant departure from other technologies. To the extent that these technologies provide graphical specification of rules, they apply only to simple rules. When rules get complex, the user quickly runs into walls that cannot be overcome. In the end, the user frequently has to leave the technology for anything complex and instead use traditional programming methods (Java, C++, scripting, stored procedures, Perl, ...).

Not so with the Co>Operating System. Once users have the Co>Operating System, they will find that it is easier to specify complex logic from within the Co>Operating System than from without. This has tremendous implications for productivity, ease of use, and transparency — it is easier and faster to specify rules graphically, and it is easier for business people to understand them.

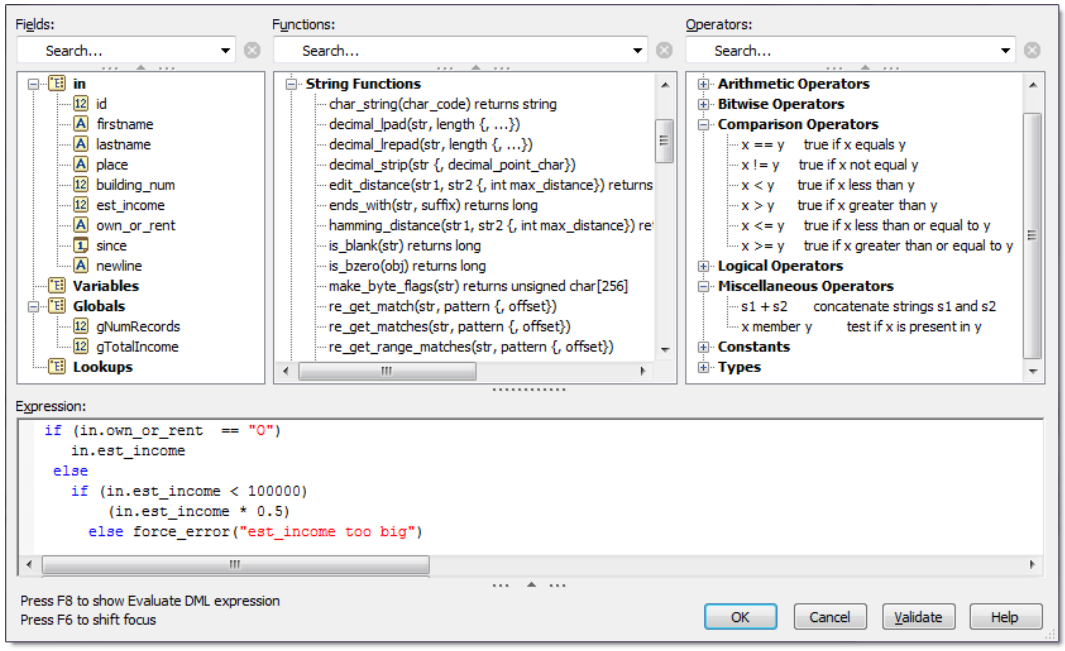
In the Co>Operating System, you specify rules using Ab Initio's Data Manipulation Language (DML), and these rules go inside "transformation" components, which are the basic building

blocks for processing data. There are transformation components for mapping one kind of record structure to another, for aggregating data, for filtering data, for normalizing and denormalizing structures, for joining multiple record streams, and more. Each of these components can apply rules specified in DML to the records as they are processed.

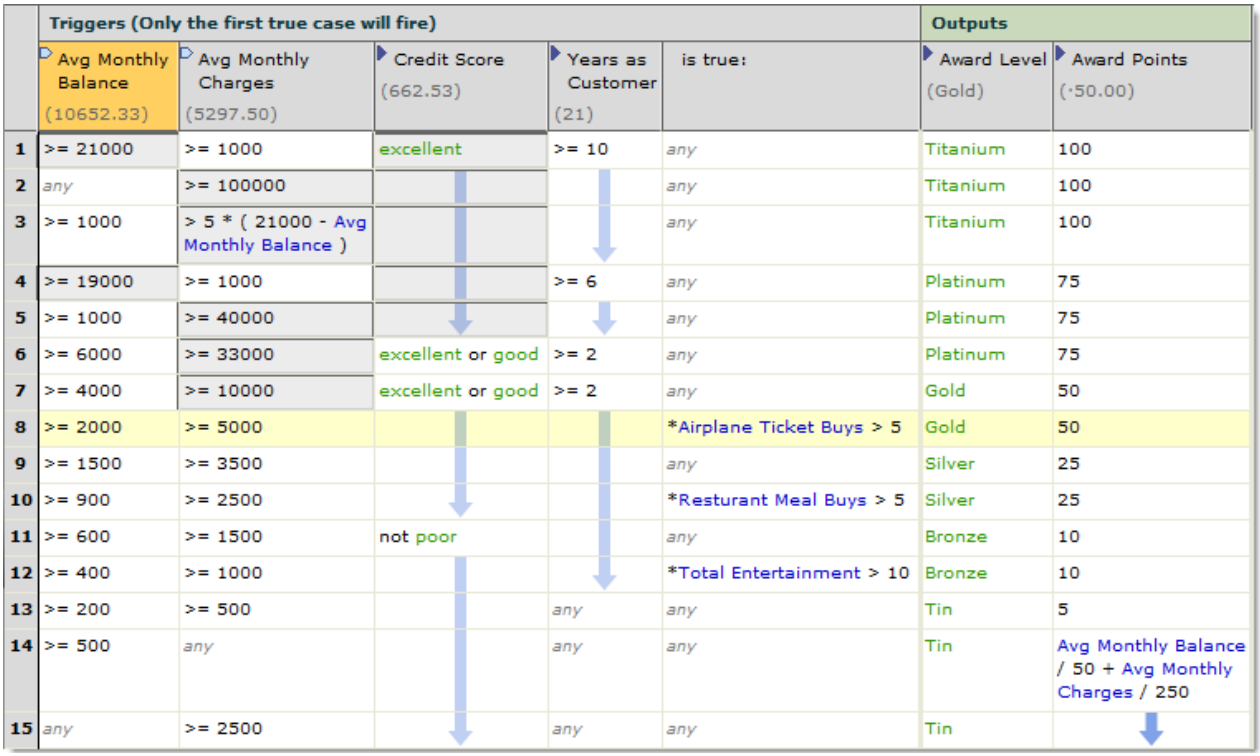
Below is a screen shot of simple rules that compute the output of a mapping component. On the left are the fields coming into the component, on the right are the fields coming out, and in the middle are the rules.



Individual rules can be built with the "Expression Editor", shown below. Expressions can be arbitrarily large and complex, and DML has a large library of built-in operators and functions.



Ab Initio also supports an alternate set of user interfaces for the specification of business rules. These interfaces are aimed at less technical users, often business analysts and subject matter experts. They use business terms rather than technical terms, and organize rules in a familiar spreadsheet-like manner:

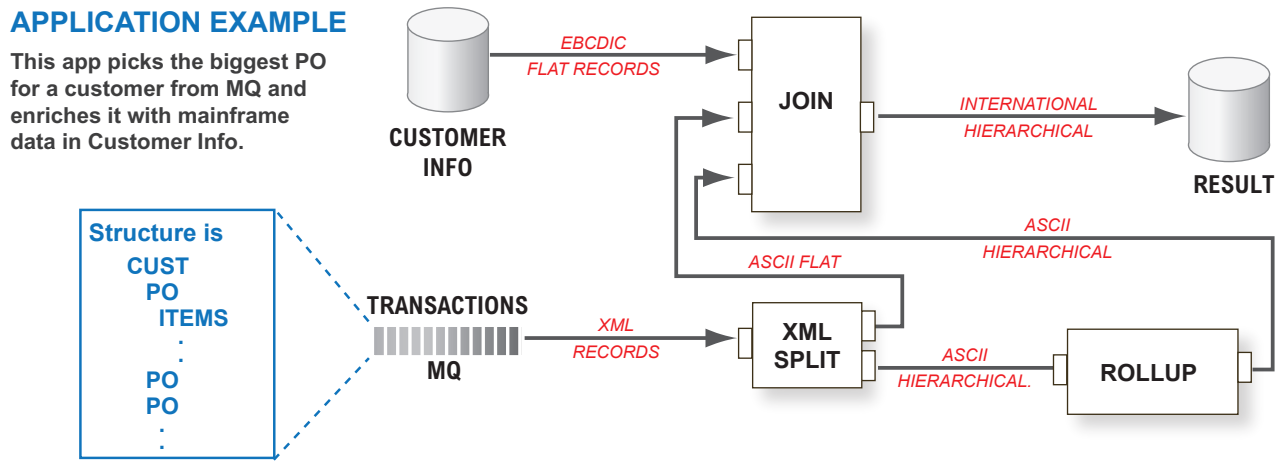


	Triggers (Only the first true case will fire)					Outputs	
	Avg Monthly Balance (10652.33)	Avg Monthly Charges (5297.50)	Credit Score (662.53)	Years as Customer (21)	is true:	Award Level (Gold)	Award Points (·50.00)
1	>= 21000	>= 1000	excellent	>= 10	any	Titanium	100
2	any	>= 100000			any	Titanium	100
3	>= 1000	> 5 * (21000 - Avg Monthly Balance)			any	Titanium	100
4	>= 19000	>= 1000		>= 6	any	Platinum	75
5	>= 1000	>= 40000			any	Platinum	75
6	>= 6000	>= 33000	excellent or good	>= 2	any	Platinum	75
7	>= 4000	>= 10000	excellent or good	>= 2	any	Gold	50
8	>= 2000	>= 5000			*Airplane Ticket Buys > 5	Gold	50
9	>= 1500	>= 3500			any	Silver	25
10	>= 900	>= 2500			*Resturant Meal Buys > 5	Silver	25
11	>= 600	>= 1500	not poor		any	Bronze	10
12	>= 400	>= 1000			*Total Entertainment > 10	Bronze	10
13	>= 200	>= 500		any	any	Tin	5
14	>= 500	any		any	any	Tin	Avg Monthly Balance / 50 + Avg Monthly Charges / 250
15	any	>= 2500		any	any	Tin	

THE CO>OPERATING SYSTEM NATIVELY PROCESSES ANY TYPE OF DATA

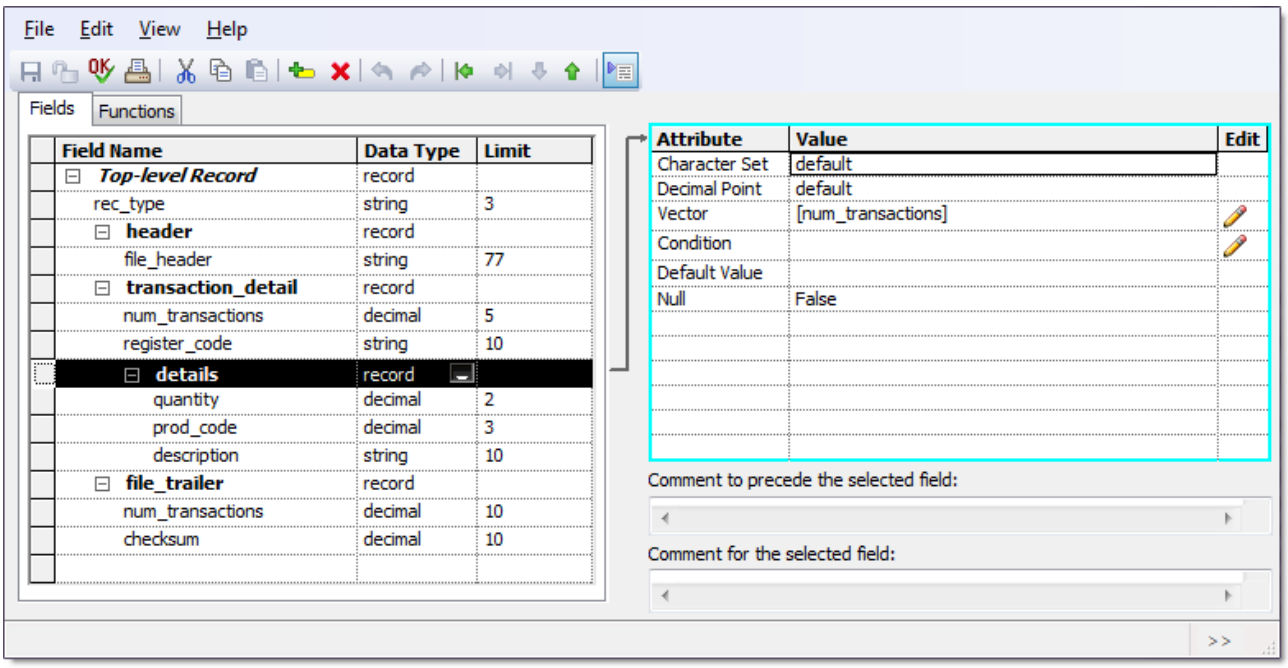
With the Co>Operating System, the data is what it or the user wants it to be — the Co>Operating System does not force data to be translated into a limited set of built-in formats that it understands. Instead, the Co>Operating System processes data in its native format and does this identically on all supported operating systems. So, the Co>Operating System, and all of Ab Initio's components, can natively process mainframe data on Unix and Windows servers, XML on mainframes, complex hierarchical and bit-packed structures everywhere, international data with automatic transcoding, and so on. Given the same data and the same application, the Co>Operating System will produce the same results, regardless of where the application is running.

Applications can read and write from a heterogeneous set of systems, and the data can have very different formats at different points in the application. The example below shows an application that reads mainframe data from a flat file, reads XML data from an MQ queue, processes that data with different intermediate formats, and outputs the results to a flat file using an international codeset.



The Co>Operating System knows how to get data formats from wherever they are stored: database catalogs, schema definition products, Cobol copybooks, XML DTDs or XSDs, WSDL, spreadsheets, or in-house data format systems.

Here is what the record format for the intermediate flow marked "ASCII hierarchical" in the example above might look like:



Finally, the Co>Operating System and its components know how to automatically translate formats as necessary. For example, if EBCDIC and packed decimal data is presented to a component writing to an Oracle database, the component will automatically translate the data to ASCII and decimals if the columns in the database have those formats.

PERFORMANCE AND SCALABILITY

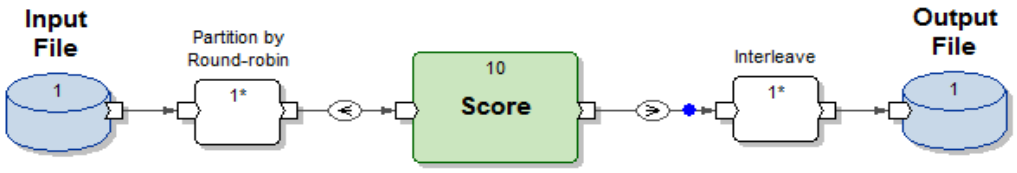
The Co>Operating System was designed from the ground up to achieve maximum performance and scalability. Every aspect of the Co>Operating System has been optimized to get maximum performance from your hardware. And you don't need "cloud" technology because the Co>Operating System naturally distributes processing across farms of servers.

The Co>Operating System is typically at least 4 to 5 times faster than the next fastest technology. This includes programs hand-coded in Java and C++: it is the rare programmer who can write a program that runs as fast as the Ab Initio version! What's more, this special programmer will be the only one in an organization who can do this with Java or C++, and he or she will spend weeks coding something that can be accomplished in just days with Ab Initio. Usually, such talented programmers don't do mere programming for long; they are tapped to do design, architecture, and even project management.

How does Ab Initio achieve both scalability and performance? What is Ab Initio's "secret sauce"? There are many ingredients, but the most important ones are architecture and fanatical attention to all details. The Co>Operating System's architecture was designed from "first principles" to enable scalable computing.

The Co>Operating System's architecture is known as "shared-nothing." Because the Co>Operating System does not require the CPUs to share anything, they can run completely independently from each other. This allows a single application to span as many CPUs on as many servers as desired. The Co>Operating System provides facilities for distributing workload evenly across many CPUs, so most applications achieve linear scalability. This means that doubling the number of CPUs leads to a doubling of performance. Ten times more CPUs means ten times more performance. The Co>Operating System combines data parallelism and pipeline parallelism to create the maximum number of opportunities to execute pieces of an application concurrently.

The simple example below shows how an application might partition data so that the Score component can run in parallel across many CPUs (and servers). The Partition by Round-robin component splits the customer data into equal streams of data, like someone dealing out a pack of cards. The Co>Operating System then runs multiple instances of the Score component, each instance working on one stream of data. As each record is output from each instance of the Score program, the Interleave component merges the streams back together before writing the result to the output file. It's as simple as that.



Shared-nothing architectures work wonderfully as long as there are no bottlenecks. But a single bottleneck will ruin the whole thing. This is where attention to detail matters. Any part of the system that might cause a serial bottleneck must be carefully designed and implemented to never get in the way. All algorithms in all components must be optimal under many different scenarios. All communication and data transport between partitions and components must use the most efficient channels. Only when all these details, and many more, are attended to, will the system scale.

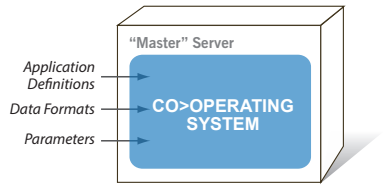
Another critical detail is the execution of business rules. If the system is designed properly, this is where a majority of the CPU time should be spent. The Co>Operating System's transformation engine is what runs the business rules. This engine has a special design called a "just-in-time compiler." This means that the business rules are compiled by the Co>Operating System at the last possible moment, when all the information about what they are supposed to do is finally available. The "just-in-time" aspect yields tremendous flexibility, and the "compiler" is highly optimized to run traditional business logic with maximum performance. This is why it is hard for a programmer using traditional technologies to compete with the Co>Operating System.

THE CO>OPERATING SYSTEM'S PROCESSING MODEL

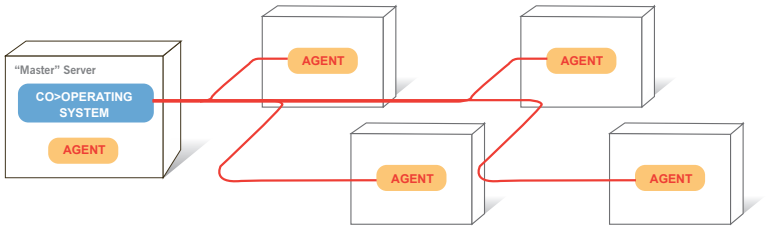
The Co>Operating System is a distributed peer-to-peer processing system. It must be installed on all the servers that will be part of running an application. Each of these servers may be running a different operating system (Unix, Linux and zLinux, Windows, or z/OS).

Here's how the Co>Operating System manages a distributed set of processes across a network of servers:

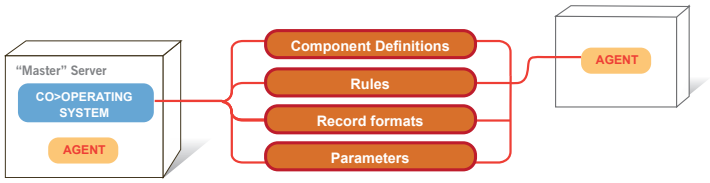
1. The Co>Operating System is started on the "Master" Server, which reads in application definitions, data formats, logic rules, parameters...



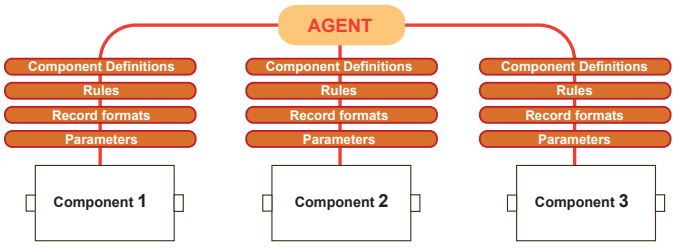
2. The Co>Operating System starts "agents" on other servers.



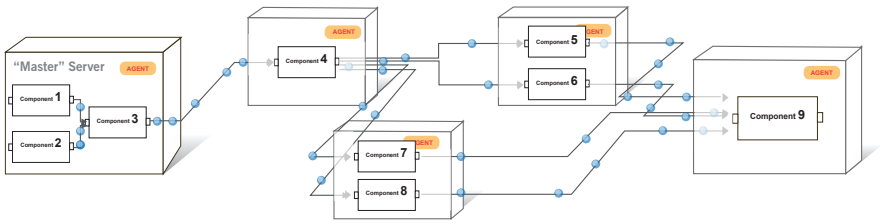
3. The master process informs each agent of components to be executed on that server, along with everything the components need to know to do their work.



4. Agents start components and tell them the rules, record formats, parameters, etc...



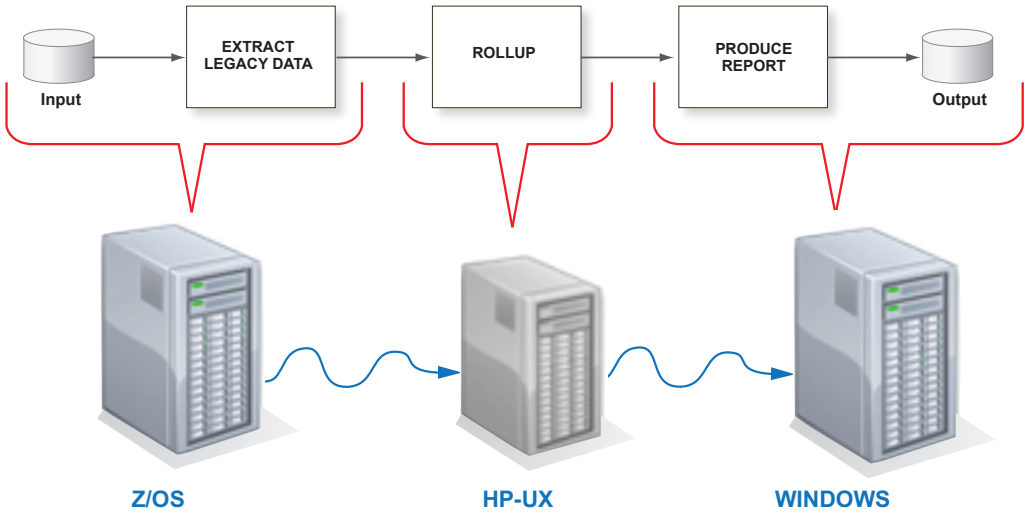
5. Components connect the data flows and begin processing the data while agents monitor the processing.



As you can see in these diagrams, a single Ab Initio application can run inside a single server or across a network of servers. The definition of the application, the graphical diagram specified by the developer, is the same in both cases. An application will run across a different set of servers just by changing the specification of which component runs where – there are no changes to the application itself. An application can therefore be rehosted from mainframes to a Unix server, or from a single server to a farm of servers, for example, with no changes.

The Co>Operating System runs equally well on Unix, Windows, Linux and zLinux, and z/OS. Furthermore, a single application can run on any mix of these platforms. Every component in an Ab Initio application can be run on any platform on which the Co>Operating System has been installed (with the exception of a few platform-specific components, such as VSAM access on mainframes). The Co>Operating System takes care of the complexity of moving data between the machines, thereby providing a seamless middleware and processing capability. Furthermore, the assignment of components to target machines can be changed before every single run of an application.

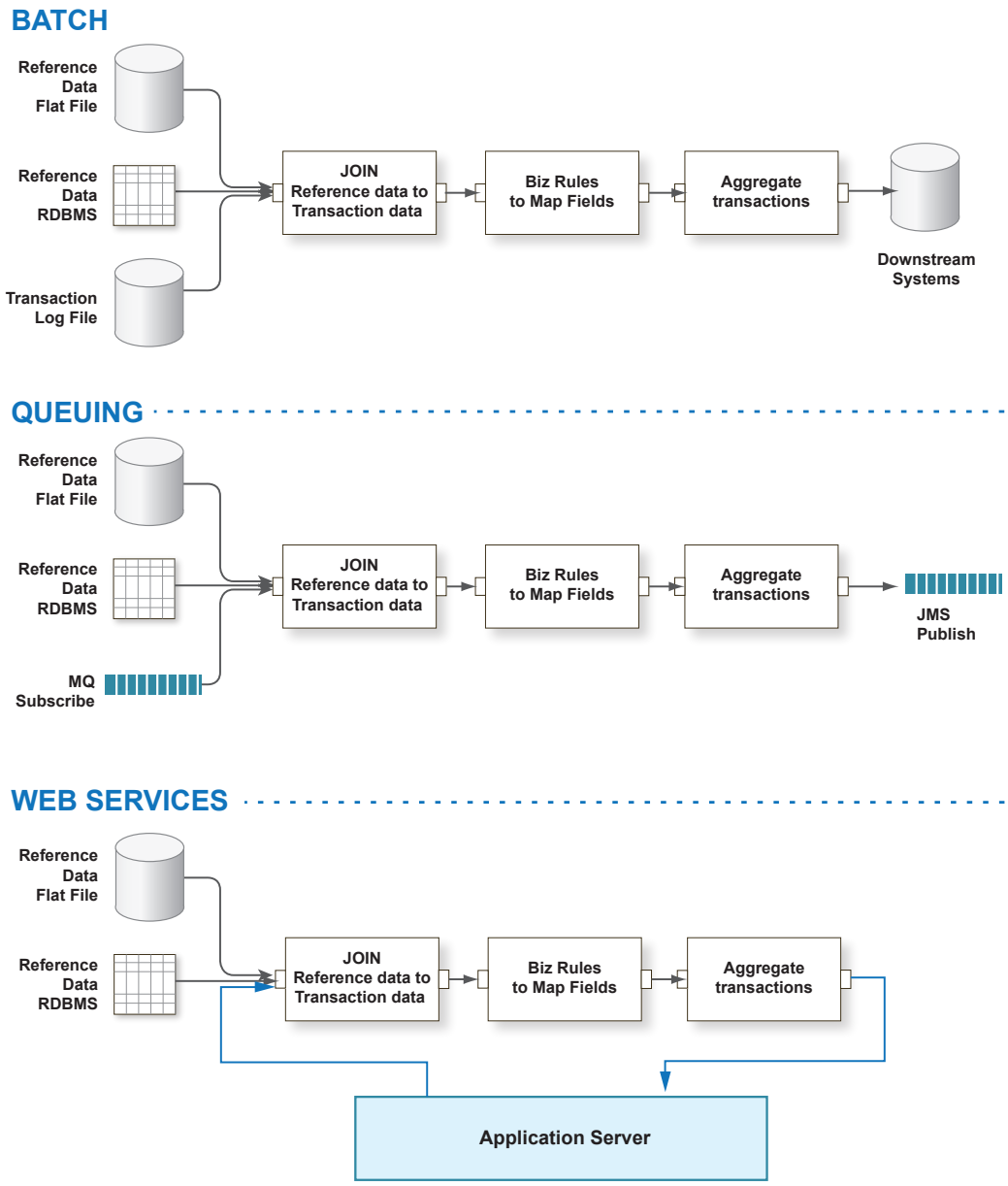
Any component or group of components can be assigned to a different computer resource:



BATCH, REAL-TIME, WEB SERVICES – THE CO>OPERATING SYSTEM DOES IT ALL

The needs for building and operating batch and real-time applications are very different, and their technologies have been different as well. Not so with the Co>Operating System. The Co>Operating System has a single architecture that applies equally well to batch, real-time, and web services (SOA) systems. Instead of requiring you to have a multitude of technologies and different development teams for each system, with potentially multiple implementations of the same business logic, the Co>Operating System, with Continuous>Flows, lets you use one technology, one development team, and one encoding of business logic that can be reused across different systems.

With the Co>Operating System, whether an application is batch or real-time depends on what the application reads and writes. Here is the same application, shown first as a batch system (hooked up to flat files for the input and output), then as a real-time queuing system (MQ for the input and JMS for the output), and finally as a web service (getting service requests from an outside system and returning results to that same system):

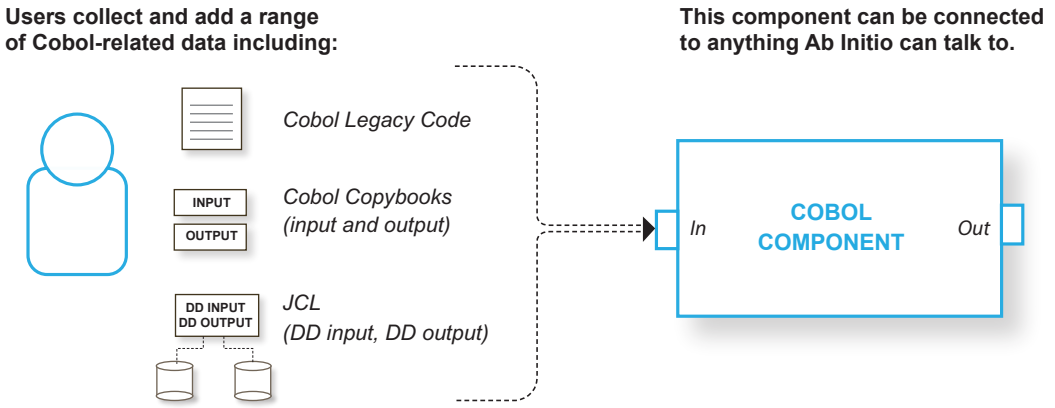


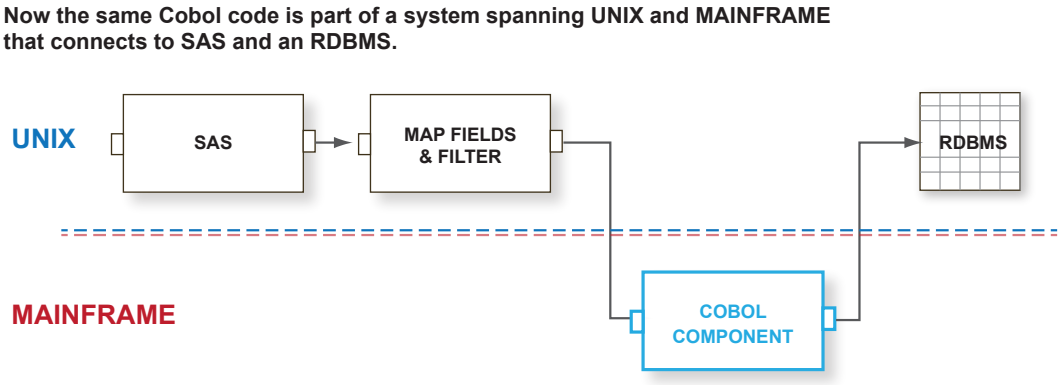
INTEGRATION WITH LEGACY CODES

While Ab Initio enables you to build end-to-end applications completely with the Graphical Development Environment, and run those applications completely within the Co>Operating System, users often have existing applications or 3rd party products that run fine and that are not worth reimplementing. Ab Initio makes it easy to reuse those existing applications, whether they were coded in C, C++, Cobol, Java, shell scripts, or whatever. In fact, the Co>Operating System makes it possible to integrate those applications into environments they were not originally designed for.

Legacy codes are integrated into Ab Initio applications by turning them into components that behave just like all other Ab Initio components. For most legacy codes, this is easy – all that is required is a specification for the program's inputs and outputs, along with command-line parameters. The example below shows how you can take a Cobol program that can read and write flat files and plug it into an Ab Initio application. The Cobol code, along with its Copybooks and JCL, is turned into an Ab Initio component; the component is placed in an Ab Initio application that spans Unix servers and a mainframe; and the application connects to various data sources and targets (SAS and a database). Finally, for performance, the workload is partitioned so that the Cobol code can have multiple instances running in parallel.

LEGACY CODE INTEGRATION





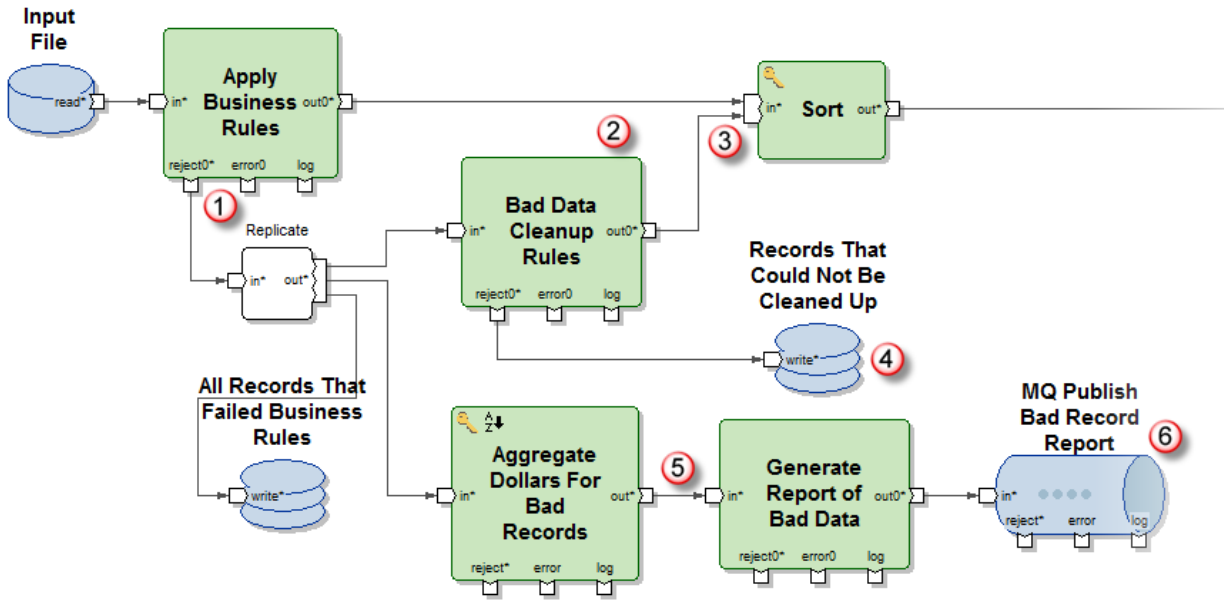
VERY HIGH RESILIENCE IN THE FACE OF BAD DATA

One of the hardest parts of building and maintaining real systems is dealing with unexpected or bad data. It happens all the time, and when fed such data, most applications behave badly — if you're lucky, they just crash. If you're unlucky, these apps may do something strange with the bad data, and it's possible that nobody will find out until the bad results have gone to downstream systems and contaminated them. Cleaning up that kind of mess takes time and really eats into productivity.

The Co>Operating System is inherently resilient to bad data and constantly checks the data to verify that it meets a variety of user- and system-specified criteria. If the data does not conform to those criteria, it is not allowed to move forward without some kind of intervention. And it is easy for the developer to build into the application automated intervention and reporting. The intervention can be designed to correct erroneous data and feed it back into the system, or it can be designed to segregate all related data entities so that they can be processed later in a consistent manner.

Below is an example of how a developer builds an application in Ab Initio that (1) captures problem data, (2) runs the problem data through a "cleanup" set of rules, (3) merges the cleansed data back into the original process, (4) sequesters data that cannot be cleansed, and finally (5) generates a report about the problem data that is (6) sent via an MQ queue.

All Ab Initio components that process data in interesting ways have optional "Reject", "Error", and "Log" ports. The Reject port gets all problem data; the Error port gets a corresponding record that describes each problem; and the Log port gets debugging information. This makes it possible to build extremely robust data processing pipelines.



VERY HIGH RESILIENCE TO SYSTEM FAILURES

Servers fail. Networks go down. Databases refuse to load. The more servers participating in an application, the higher the probability that there will be a system failure of some type. It is hard to build applications that can reliably recover from such failures. While many developers think they can, in practice you don't discover whether the architecture is robust until it has survived a variety of failures. There can be much pain while climbing this learning curve.

The Co>Operating System, on the other hand, was designed from the beginning to reliably recover from all these types of failures. As long as the environment has been configured so that crucial data is not lost, the Co>Operating System's checkpoint/restart facility will be able to restart an application where it stopped before the failure, even if the application spans networks, multiple servers, and multiple databases, and regardless of whether the application runs in batch or real-time. The Co>Operating System uses a two-phase-commit-like mechanism that has much higher performance than the industry-standard XA protocol. For environments that require the XA protocol, the Co>Operating System supports this as well, even across multiple databases and queuing technologies from different vendors.

THE CO>OPERATING SYSTEM DRIVES PRODUCTIVITY WITH REUSE

Because of the nature of the problems they are solving, Ab Initio applications can become very large and complex. However, most systems include many pieces that are very similar to other pieces, or for which there may be many variations on a theme. With traditional programming technologies, developers frequently end up making many copies of these pieces and then making minor modifications to each copy. The result is a maintenance and productivity nightmare.

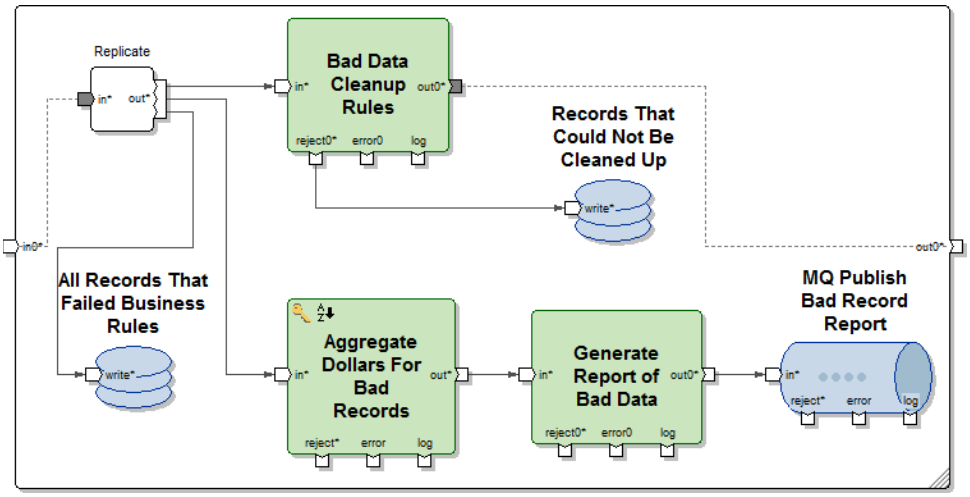
Ab Initio provides a number of mechanisms to radically increase reuse of these application pieces. Application pieces of all types can be stored in the Ab Initio Enterprise Meta>Environment (EME), a centralized repository, and reused both within and across applications. Here are examples of what you can centrally store, manage, and reuse:

- Record formats
- Business and logic rules
- Sections of applications (applications are called "graphs" and the sections are called "subgraphs")
- Orchestrations of applications ("plans" in Ab Initio's Conduct>It)

Ab Initio's reuse capability is very powerful. Reused application pieces can link back to and track changes to the centralized version from which they came. Furthermore, these pieces can support locally applied customizations while still tracking the centralized version.

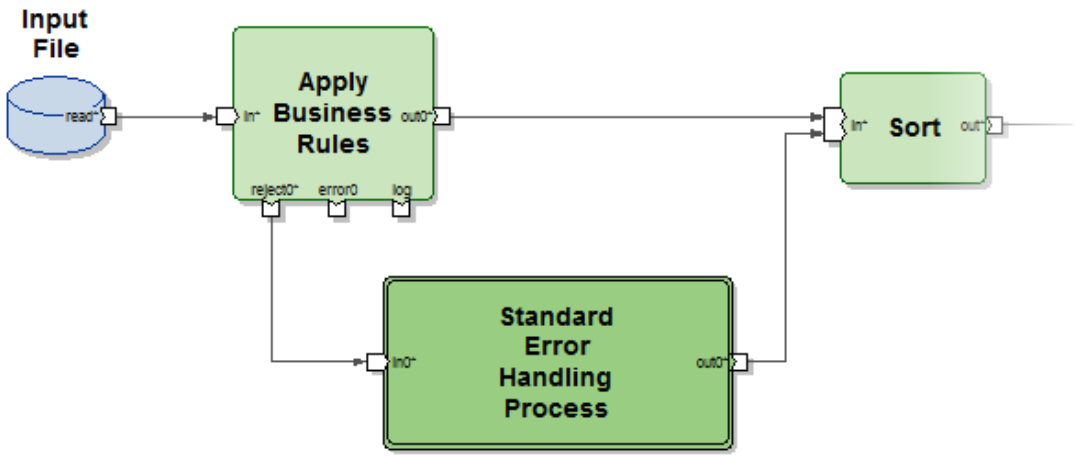
Below is an example of just one of these reusable application modules: the subgraph. A subgraph can contain the components of a subsection of an application (graph). There is no limit to the size or complexity of a subgraph, and subgraphs can be nested. Subgraphs behave in all the ways that normal components do. And they can be saved in a library for reuse across many applications.

This subgraph corresponds to the error-handling portion of the application described earlier:



Notice that the components in this subgraph do not explicitly connect to any input or output components. Instead, they connect to the input and output ports of the "Standard Error Handling Process" subgraph.

Here is the same application as before, now built with the new, reusable error-handling subgraph (subgraphs are visually identified in the GDE by the extra border line surrounding them):



CONCLUSION

The Co>Operating System is the base software for the entire Ab Initio architecture. All the key Ab Initio architectural concepts manifest themselves in the Co>Operating System. And because all of Ab Initio's technologies incorporate the Co>Operating System in one way or another, they incorporate those concepts in a consistent and seamless manner.

The results of this architecture are that:

- Entire application systems can be built graphically, and without traditional coding.
- Technical development and maintenance staff is far more productive with a graphical paradigm than with traditional coding.
- Technical staff can be far more responsive to changing business needs.
- Applications are far more transparent and therefore easier for business users and analysts to understand.
- Application systems are far more robust in every sense: they have unlimited scalability, they are portable across many platforms, they can deal with any kind of complex data, they can implement very complex business rules, they can withstand bad data and system failures, and so on.
- Applications can reuse the business logic across batch, real-time, and web services applications.

These things, and more, are possible because the Co>Operating System was designed from the beginning with a single architecture to achieve these results.

ENTERPRISE META>ENVIRONMENT:
The Ab Initio Metadata System

The IT infrastructure is the central nervous system of modern businesses, and management needs to know everything about it. What information flows through it, what does that information represent, how accurate is it, how does it flow from one place to another, how is it processed, and where is it stored? This is what **metadata** is about — it is "information about information."

But getting that metadata is not so simple. While there are products that claim to address this need, they have taken a very academic approach. Indeed, the concept of "information about information" raises the question of what "information" is in the first place. So these metadata products have focused on defining concepts and how the concepts relate to each other. While these concepts eventually connect with actual information, the connections are tenuous. This metadata must be manually entered by humans and is therefore subjective, incomplete, subject to human error, and inevitably obsolete, since it trails the real systems, which are changing constantly.

Ab Initio has taken a very different approach by focusing on operational metadata. **Operational** metadata is actionable by business and IT management. It is about the systems that process data, the applications in those systems, and the rules in those applications. It is about the datasets throughout the enterprise, what is in them, how they got there, and who uses them. It is about the quality of the data, and how that quality has changed over time. It is about all the many things in all the IT systems.

Ab Initio also ties this operational metadata together with **business** metadata — business definitions, created by business people, of the various pieces of information throughout the enterprise. The result is a true enterprise metadata management system — the Ab Initio Enterprise Meta>Environment, or EME.

An enterprise metadata management system must be many things to many people:

- The CFO needs to be able to tell regulators what a field in a report means and what the source of the data in it is.
- The CIO wants to know about the IT systems — hardware and software — in the company. Who owns this system? What systems does it depend on? What systems depend on it? What is the level of data quality across these systems, and how does it change from one system to the next?
- The business analyst who is helping a division head manage her business needs a business glossary that will help her find the pieces of data she needs to pull together for an analysis by 5 PM today.
- The operations staff wants to know what happened in production, today and in the past. What jobs ran successfully? How long did they take? How much data was processed? How much spare capacity is available? How good is the quality of the data?
- The systems architect is concerned with the inventory of applications, data tables, files, and messages that make up the company's systems. How do they all connect? What produces what? What reads what? What depends on what?
- Application developers want to know the history of changes to their code. What does the data look like now? Who fixed what? When? How? Why? What got released? What work is still in progress?

There is no end to these kinds of questions. Getting useful answers quickly is essential. These are questions that can be answered with the Ab Initio Enterprise Meta>Environment.

DIFFERENT METADATA IN DIFFERENT CONTEXTS

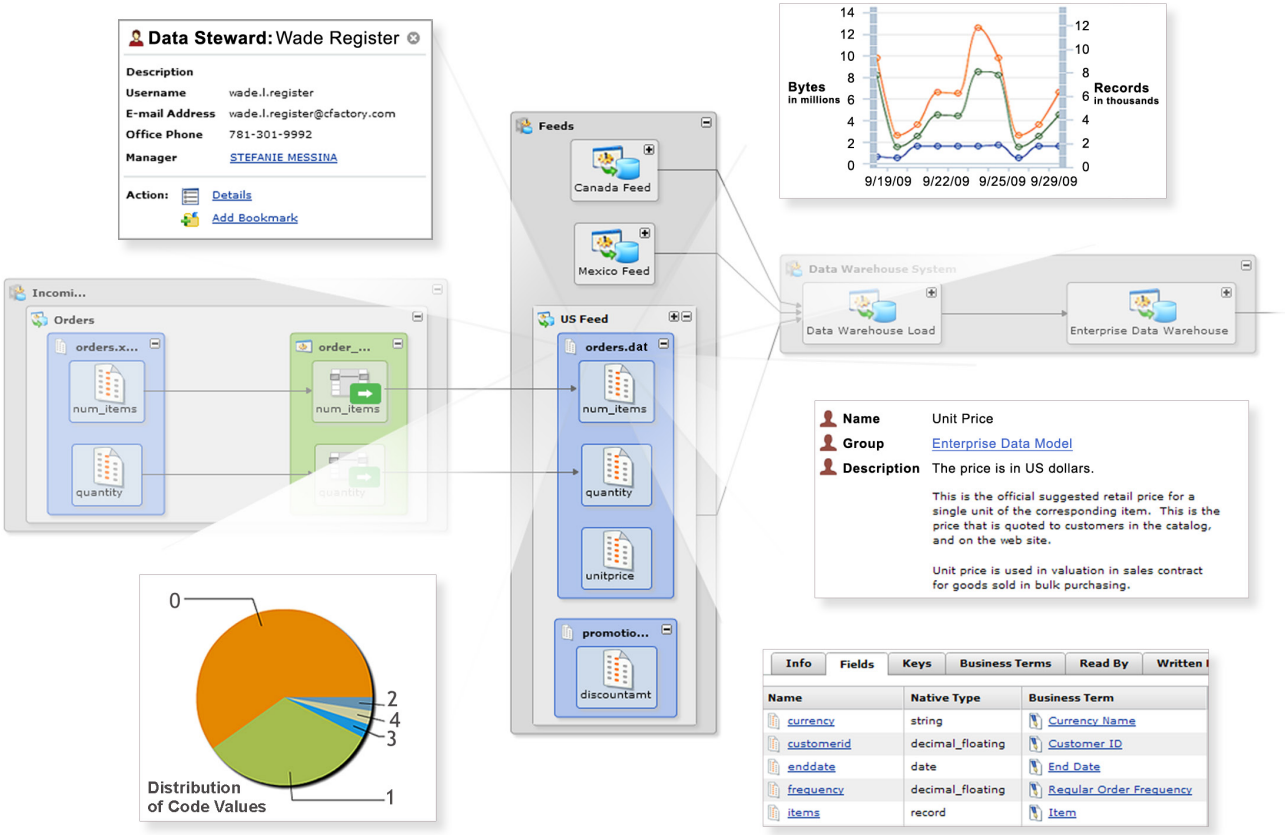
The term "metadata" has different meanings across industries. Ab Initio uses the term "metadata" in the context of the business computing world. In the image processing world, for example, it means something altogether different: information such as when an image was captured, what kind of device took the picture, what the lighting was, and so on. Web pages have metadata too, this being the language the page was written in, the tools used to create it, and how to find more information on this topic.

NAVIGATING AND UNDERSTANDING METADATA

Ab Initio's metadata graphical user interface, the EME Metadata Portal, allows one to start at any point in the system and explore in any direction one chooses. All this metadata is presented at the appropriate level of detail for each audience. Business users are not overwhelmed with technical minutiae when they are trying to answer business questions, while developers and operational staff can easily find the details of interest to them.

The EME can show:

DATA STEWARD INFORMATION, OPERATIONAL STATISTICS, DATA PROFILE RESULTS, CONCEPTUAL DEFINITIONS, DATASET DETAILS, MAPPING SPECIFICATIONS, ENTITY RELATIONSHIPS, DATA QUALITY METRICS, DATA QUALITY HEATMAP & SEMANTIC MODELS



A screen shot of the EME in the process of navigating metadata. The underlying screen is a lineage diagram displaying a number of datasets and their processing relationships. Each of the overlays shows different types of metadata that have all been linked to the same metadata element.

Consider a file that the EME has identified as the ultimate source for a calculation used in a report. What can the EME tell you, a user, about this file? Through Ab Initio's approach of relating elements of metadata, one to the other, you can glean interesting and important information about the file from the intuitive graphical interface, including:

- Which applications use the file
- Its record format
- Its data quality
- Its size over time
- The documented, expected values for each of its fields
- The actual census of values found

- The stewards (and their managers) responsible for its governance
- Documentation about its business meaning and the use of each of its fields
- Its relationship to logical models and similar datasets, including database tables and messages
- A list of programs that read or write the dataset

METADATA INTEGRATION

Capturing so much metadata and storing it in separate buckets would be an accomplishment in and of itself, but the EME does more than that. It establishes relationships between elements of metadata, which effectively enriches their value, revealing deeper meaning about the business to the real-world users of metadata at a company.

The challenge, of course, is how to gather all this metadata in a way that is actually useful. In large, complex organizations with heterogeneous, distributed (even global) environments, this challenge is particularly hard. There are issues of scalability and integration. How to gather metadata from such a disparate set of sources and technologies? How to process so much information? How to store it and display it intelligently, without overwhelming the user or dumbing down the content? How to marry metadata across lines of business, countries, even languages?

The EME integrates all the different kinds of metadata stored in it and, as a result, multiplies the value of each. For example, this integration enables end-to-end data lineage across technologies, consolidated operational statistics for comprehensive capacity planning, and fully linked data profile statistics and data quality metrics.

To begin with, all information about the definition and execution of Ab Initio applications is automatically captured and loaded into the EME. This includes business rules, data structures, application structure, documentation, and run-time statistics. Because users build end-to-end operational applications with the Co>Operating System, everything about those applications is automatically captured.

This metadata is then integrated with external metadata through a combination of the EME's Metadata Importer and sophisticated metadata processing with the Co>Operating System.

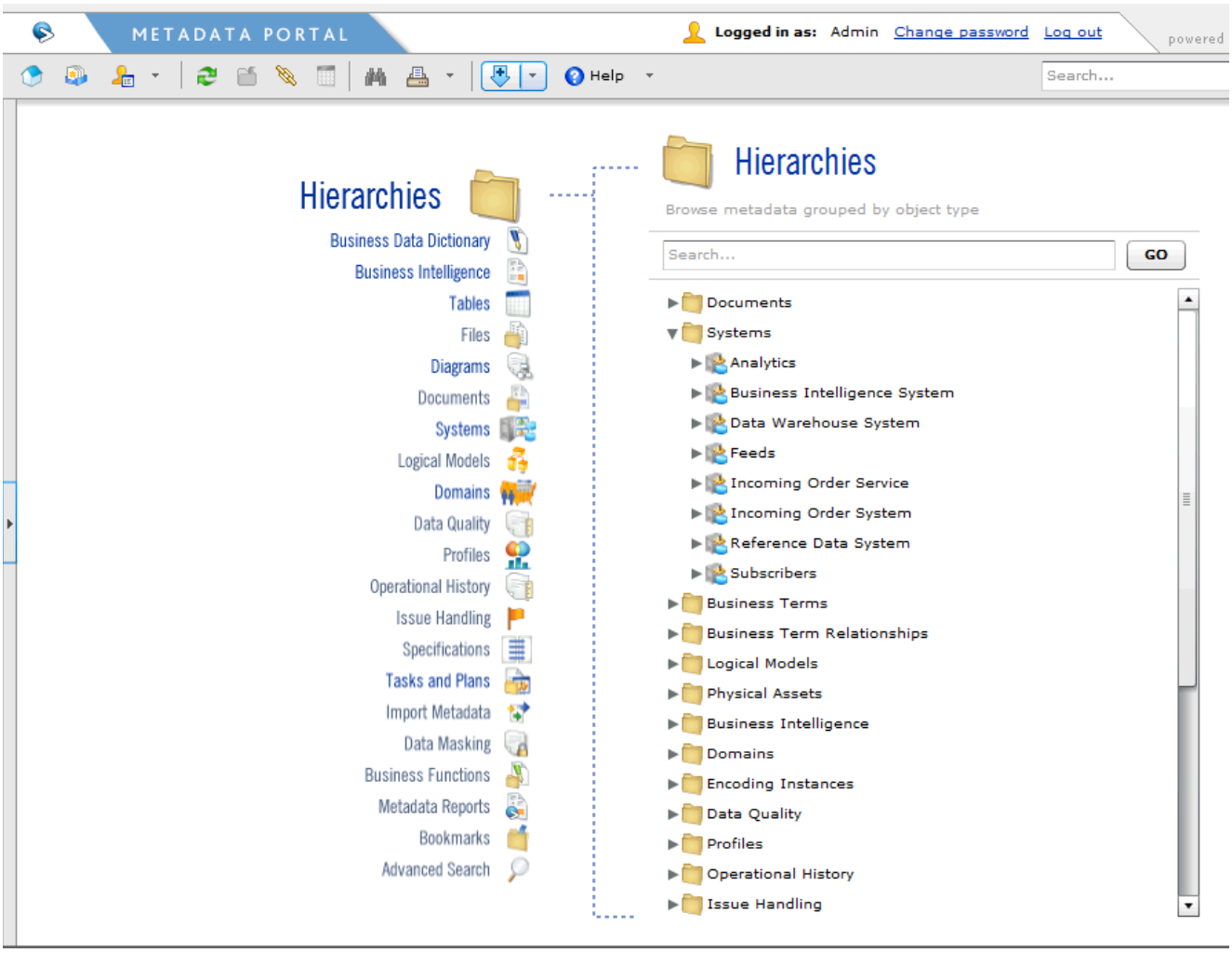
Ab Initio's support for combining metadata from multiple sources allows metadata from one source system to be enriched with metadata from other sources. For example, the Metadata Importer might load the core details of database tables and columns from a database catalog, then enrich the metadata with descriptions and logical links from a modeling tool, and finally link the imported metadata to data quality metrics. The Metadata Importer can load external metadata such as:

- Reporting tools: MicroStrategy, Business Objects, Cognos, ...
- Modeling tools: ERwin, ERstudio, and Rational Architect, ...
- Database system catalogs for all major and most minor relational database management systems
- Tabular metadata, usually stored in spreadsheets using either predefined templates or customer-specific layouts
- Industry-standard protocols for metadata exchanges, including Common Warehouse Model XML Metadata Interchange Format (CWM XMI)

Non-standard and custom metadata sources can also be imported and integrated into the EME. Users can apply the Co>Operating System's powerful data processing capabilities to arbitrarily complex sources of metadata. The Co>Operating System can extract metadata from these non-standard systems, process it as necessary, and load and integrate it with other metadata in the EME.

MANY TYPES OF METADATA

The EME integrates a very wide range of metadata and is fully extensible. The home page of the Metadata Portal allows the user to directly navigate the type of metadata of interest:



From this page you can select an area of interest and dive in to see:

METADATA ABOUT PROJECTS AND APPLICATIONS. The EME stores and manages all information about Ab Initio projects and the applications they contain. Projects are organized in hierarchies and can be shared or kept private. The EME keeps track of which projects reference other projects, as well as tracking all objects within a project.

DETAILS ABOUT APPLICATION VERSIONS. The EME maintains complete version information and history about every detail of Ab Initio applications. Differences between versions of graphs, record formats, and transform rules are displayed graphically. Users can see details about the exact versions that are being used in production.

USERS, GROUPS, LOCKS, AND PERMISSIONS. The EME provides access control management for all metadata. Furthermore, as part of a complete source code management system, the EME's exclusive locking mechanism for whole applications or pieces of applications prevents developers from interfering with each other.

HIERARCHICAL ORGANIZATION OF METADATA. Metadata can be organized into arbitrary hierarchies and folders to help capture business meaning and to provide targeted navigation.

DATA DICTIONARIES. The EME supports the creation of one or more data dictionaries or conceptual data models. Data dictionaries can be a simple hierarchical list of business terms, or a more complex semantic model with complex relationships between business terms.

Enterprise-wide deployments typically have multiple data dictionaries – one for each division or product area, as well as an enterprise model. In the EME, divisional business terms link directly to columns and fields, and have relationships back into the enterprise model. This allows companies to harmonize business concepts across the enterprise without forcing each division to abandon its own data dictionary.

METADATA FROM REPORTING TOOLS. The EME imports metadata from all the major business intelligence (BI) reporting tools, including MicroStrategy, Business Objects, and Cognos. This includes details about reports and report fields, as well as internal reporting objects such as Facts, Metrics, Attributes, and Aggregates. Lineage queries can trace the calculations of various report fields back through the BI tools into the data mart or data warehouse, and from there all the way back to the ultimate sources.

METADATA FROM DATABASE SYSTEMS. The EME imports metadata (schemas, tables, columns, views, keys, indices, and stored procedures) from many database systems. The EME performs lineage analysis through multiple levels of views and stored procedures. For large database systems, the EME is often the only way to understand the interrelationship of database tables, views, and procedures – especially for impact analysis queries, table reuse exercises, and consolidation projects.

METADATA FROM FILES. The EME imports metadata about files, including complex hierarchical record formats such as XML and COBOL copybooks.

END-TO-END DATA LINEAGE. The EME builds complete models of the flow of data through an enterprise by harvesting metadata from a large number of different operational systems, reporting tools, database systems, ETL products, SQL scripts, etc. This integrated model allows users to query the system about data lineage – how data was computed, and what is impacted by a change.

SYSTEM DIAGRAMS. The EME stores graphical pictures that can represent system diagrams or other diagrams of metadata organization. In the Metadata Portal, clicking on a "hot-linked" graphical item within a diagram navigates the user to the connected metadata object.

LOGICAL MODELS. The EME imports logical and physical models from common modeling tools. It models links from logical models to physical models, which are then merged with the schema information in the actual databases.

DOMAINS AND REFERENCE DATA. The EME stores reference data, including domains and reference code values. It can be the primary manager for certain reference data, or can track and maintain a copy of reference data from a different system. It also supports code mappings between logical domain values and multiple physical encodings.

DATA PROFILES. The EME stores data profile results and links them with datasets and individual fields. Many statistics are computed, such as common values and data distributions. These statistics can be computed on demand or automatically as part of an Ab Initio application.

OPERATIONAL STATISTICS. The Co>Operating System produces runtime statistics for every job and for every dataset that is read or written. These statistics can be stored in the EME for trend analysis, capacity planning, and general operational queries.

DATA QUALITY METRICS. To support a comprehensive data quality program, Ab Initio computes data quality statistics and error aggregates and stores them in the EME. The EME can analyze and display data quality metrics for datasets and for collections of datasets. Data quality metrics can also be combined with data lineage to see a "heat map" showing where there are data quality problems in the enterprise.

PRE-DEVELOPMENT SPECIFICATIONS. The EME can capture mapping specifications as part of the development process. The Metadata Portal allows analysts to specify existing or proposed sources and targets along with arbitrary mapping expressions. By using the EME for defining mappings, users can see how the mappings fit into a larger enterprise lineage picture.

These specifications can then be used to guide a development team and to permanently record requirements. After production deployment, the EME will continue to show these specifications in lineage diagrams alongside their actual implementations.

DATA MASKING RULES. The EME stores data masking rules, which can then be applied to data flowing through Ab Initio applications. Ab Initio provides many built-in rules, and users can define their own custom masking algorithms. These rules can be associated with fields or columns, or with business terms in the conceptual model. When linked at the conceptual level, data masking rules are automatically applied to the corresponding physical columns and fields.

DATA STEWARDS AND METADATA ABOUT PEOPLE AND GROUPS. The EME stores metadata about people and groups. This metadata can be linked to other metadata objects to document data governance roles such as data stewardship. Metadata about people and groups can be automatically imported from external systems such as corporate LDAP servers.

BUILT-IN AND CUSTOM METADATA REPORTS. The EME provides many built-in reports. Users can also define custom reports that run against metadata stored in the EME and that are accessible from the Metadata Portal.

CUSTOM METADATA. Users can extend the EME schema to allow a wide variety of additional metadata to be integrated into the EME. Schema extensions include the addition of attributes to existing objects, as well the creation of new metadata objects that can be linked to other existing metadata. Users can easily customize the EME user interface to allow for tabular and graphical views on both standard and custom metadata.

THE EME IS AN OPEN SYSTEM

The EME is an open system based on industry-standard technologies:

- A published, extensible relational schema. The EME comes preconfigured with a rich metaschema that contains a wide variety of types of metadata. The metaschema can be customized and extended with custom tables and columns to support a variety of user-defined metadata. The EME manages these extensions and customizations in concert with the built-in metadata objects, and provides full customization of screens and reports.
- A standard commercial relational database (currently Oracle, DB2, or Microsoft SQL Server), which holds all the business metadata and summaries of the operational and technical metadata. Technical metadata is stored in an object data store accessible via ODBC.
- A graphical user interface that can be hosted in any standard web browser. In addition, the EME supports navigation to external repositories of detailed metadata, such as document management systems, image databases, and 3rd party products.
- A three-tier architecture using commonly available application server technology. On top of the database is a standard Java application server (currently WebSphere, WebLogic, JBoss, or Apache Tomcat) that manages security, calculates role-based views, and implements the workflows around metadata maintenance.
- Support for external reporting tools. While the EME supports a wide range of built-in reports through the Metadata Portal, 3rd party reporting products can also directly access the metadata in the database for custom reports. The relational schema is fully documented and comes with preconfigured database views to support these reporting tools.
- Web services APIs to enable service-oriented architecture and metadata as a service. These interfaces allow external systems to query business metadata as well as to submit metadata change requests. External systems can also subscribe to changes to the metadata, thereby enabling the EME to send messages when any approved change occurs. For example, if the EME is managing valid values, the approval workflow (described later) can send messages to external operational systems to update their cached lookups of these valid values.
- Metadata exports. In addition to the data access interfaces, the EME can also export metadata in a number of ways. For example:

- Virtually every EME tabular screen can be converted into an Excel spreadsheet with the click of a mouse.
- The EME can export metadata in the emerging standard for metadata exchange, CWM XMI.
- The EME can generate a Business Objects universe and populate it with metadata.

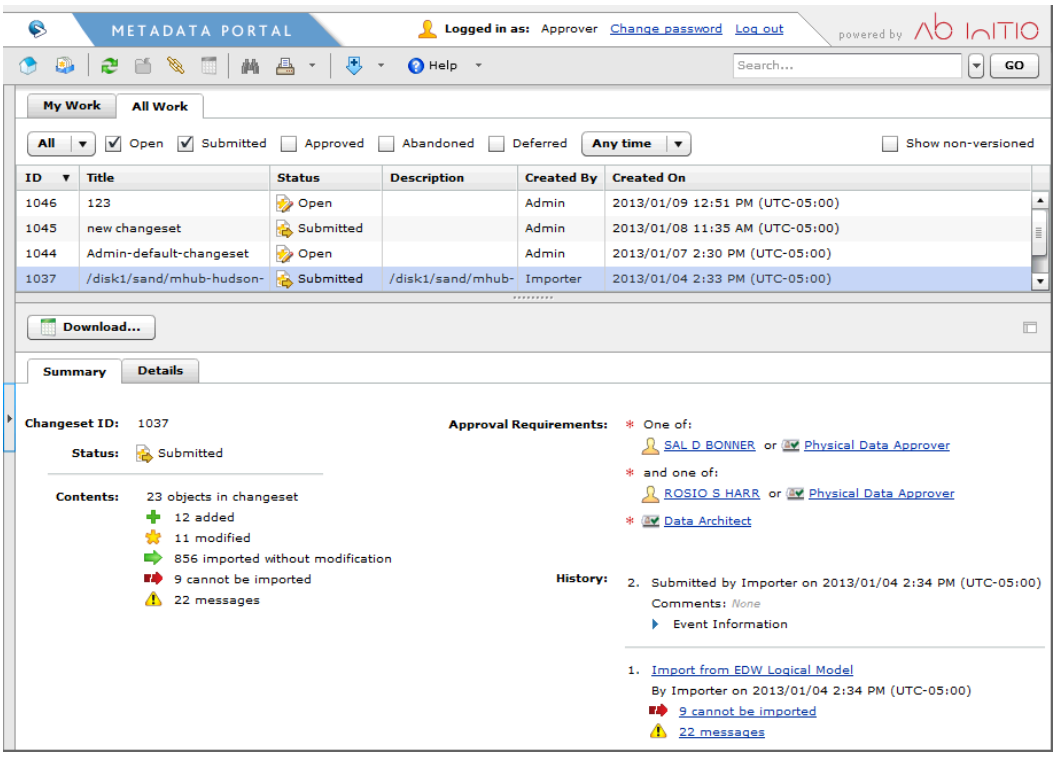
METADATA GOVERNANCE

The EME provides sophisticated governance processes that can be customized to meet the needs of large enterprises.

For technical metadata (applications and business rules), the EME supports a complete source code management system, with checkin/checkout, locking, versioning, branching, and differencing.

For business and operational metadata, the EME comes with a built-in metadata governance workflow, including work queues, approvals, and audit trails. The EME can also interface with external approval workflow tools. The EME's proposal/approval workflow mechanism is based on changesets. Users create changesets to propose metadata additions, updates, and/or deletions, and then submit them for approval.

Below is a screen shot of the changeset submission process:



When a user submits a changeset for approval, the EME sends an email message to the appropriate metadata stewards. These stewards can inspect the proposed changes and approve or reject them. If approved, the changeset is applied and becomes visible to the general user population.

The EME also supports integration of changesets via its web services API, as well as with external workflow approval/BPM systems, such as Oracle's AquaLogic. In this case the external workflow system is responsible for communicating items in work queues, documenting communications, managing escalations, and resolving final status.

All approved changesets result in new versions of metadata in the EME. The EME maintains a complete history of all previous versions and their details.

A FINAL WORD

Enterprise metadata management was long a goal of large companies, but an unattainable, impractical one. Passive "repositories" (in many cases simply glorified data dictionaries) held only a fraction of the relevant metadata and soon became stale, out-of-date "islands" of metadata. The organizations that most needed a comprehensive approach to managing metadata – complex, global companies with inherent problems of scalability, with diverse metadata sources, with security issues that cross lines of business, and with huge amounts of information to display and navigate – were the least likely to succeed.

But Ab Initio's Enterprise Meta>Environment has finally made enterprise metadata management possible, even in the largest of companies. Some examples:

- A major global bank is finally able to meet the requests of its regulator that its accounting numbers be verifiable. A full-scale data quality program across the enterprise, including quality measurements at various points of the data lineage, is being rolled out using the EME.
- A major financial institution is saving tens of millions of dollars on the replacement of a key software system because the EME has enabled a complete understanding of how the legacy code worked, and empowered business and IT to collaborate in describing how the replacement system should function. Many person-years of planned effort were simply eliminated.
- Several multinational enterprises with incredibly complex IT environments – operations in as many as 100 countries, thousands of disparate systems, and hundreds of thousands of files, database tables, and messages – are using the EME to inventory every piece of data and to define its meaning and value; they're using the EME as an asset management system. These companies realize that data items are assets to be tracked, just like cars, buildings, and office equipment.

The Ab Initio EME didn't happen overnight, and it didn't come from an ivory tower: it's the result of years of serious engagement with companies like these.

CONTINUOUS>FLOWS: Real-time Processing

Real-time business data processing is truly challenging, and few software products actually tackle the requirements. Ab Initio does.

Ab Initio addresses a wide spectrum of real-time applications, ranging from "mini-batch", to high-volume "asynchronous messaging", to service-oriented applications (SOA), to low-latency "request/response" applications, all with a single technology — the Co>Operating System's Continuous>Flows facility.

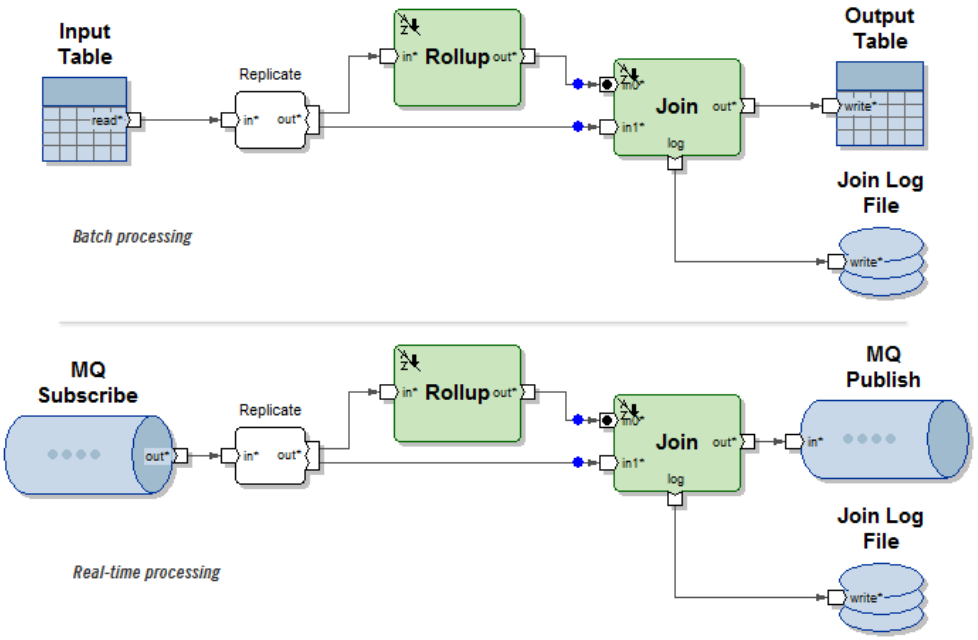
The Co>Operating System is a "dataflow engine." This engine flows streams of records, or transactions, through a sequence of "components." These components each do a variety of computing on input records, applying business rules, for example, to produce output records. Complex processing logic is decomposed into easily understood steps, with each step carried out by a different component. Records flow through the necessary set of components until the processing is complete.

This dataflow model is perfectly suited for both batch and real-time processing. While most of a batch and corresponding real-time application may be similar if not identical, the end-point components determine whether the application is batch or real-time. A batch application connects to flat files and static database table components. A real-time application connects to messaging queues, web services, RPCs, CICS servers, and/or special purpose components (usually via TCP/sockets).

With Ab Initio, the fact that batch and real-time applications have so much in common, and that they both use a single technology — the Co>Operating System — results in significantly lower complexity and higher performance. Lower complexity translates to higher productivity, and higher performance lowers costs.

REUSABILITY OF BUSINESS LOGIC ACROSS BATCH AND REAL-TIME APPLICATIONS

In most cases the business-logic components between the input and output components stay the same, meaning that the same business logic can be reused across batch and real-time applications. This has big productivity implications. With non-Ab Initio approaches, the technologies and methodologies for batch and real-time processing are usually very different, so that the same business logic is reimplemented multiple times for the range of batch to real-time uses. With Ab Initio, you develop the business logic just once and then plug it into Ab Initio's batch and real-time infrastructures:



ACHIEVING PERFORMANCE FOR DIFFERENT REAL-TIME EXECUTION MODELS

Application architects are often challenged by the need to meet seemingly conflicting performance requirements:

- Batch applications need to process data as efficiently as possible. A batch job may take a long time to run because there are so many transactions to process, and none of the results are available until the job has completed. But while it is running, a batch job is expected to process a very high number of records per second.
- "Mini-batch" applications are collections of batch jobs that individually process small volumes of data. However, there may be thousands or even tens of thousands of such small jobs that run each day. By limiting the amount of data processed by a job, the response time for each job is minimized. This approach also allows the same applications to process very large data volumes efficiently in a traditional batch setting. (Managing tens of thousands of jobs a day presents its own set of complexities, which are addressed by Ab Initio's Conduct>It.)

- Asynchronous messaging applications connect to message queues and also need to process transactions as efficiently as possible. However, the downstream systems usually expect their response messages within a few seconds to tens of seconds. Indeed, if an asynchronous application can respond within a second or two, it can support interactive use.
- "Request/response" or synchronous messaging applications are expected to process a transaction as soon as it shows up and to respond as quickly as possible, usually with a latency of less than a second. If multiple such applications work together to process a transaction, individual applications may need to turn around responses in tenths to hundredths of a second. Ab Initio directly addresses this "sweet spot" of reliably performing meaningful units of work in the tens of milliseconds range (in contrast to the extremes that some narrow, specialized systems go to).

The Ab Initio Co>Operating System's Continuous>Flows capability is a single technology that is effectively used by customers in all these modes. This is because the Co>Operating System was architected, from the beginning, to meet all the requirements of these different approaches.

There are two primary differences in Ab Initio between batch (including mini-batch) and real-time applications:

- **Termination:** Batch applications terminate once they have finished processing all the data in their inputs. After termination, no system resources remain associated with a batch application. Once started, real-time applications stay up indefinitely to receive and process transactions that arrive on their inputs. If there is a lull in the flow of new transactions, a real-time application waits until new transactions appear.
- **Checkpointing and recovery:** Batch applications take checkpoints at predetermined locations in an application, and all data that passes through one of these locations is saved to disk (or the checkpoint has not been successfully taken). Recovery is just simply restarting an application at the last successful checkpoint. Real-time applications can take checkpoints between transactions, as often as every transaction or infrequently based on other criteria (such as elapsed time or number of transactions). A restarted application automatically picks up at the last transaction that was successfully checkpointed.

INTERFACING WITH A WIDE RANGE OF REAL-TIME SYSTEMS

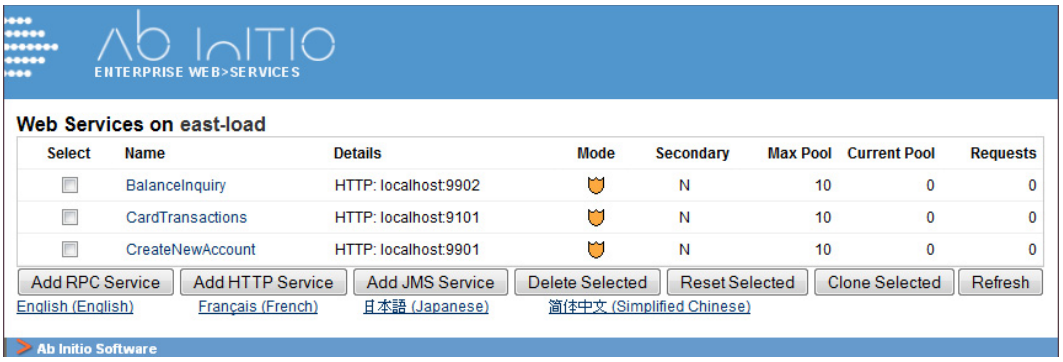
Ab Initio's Continuous>Flows provides interfaces to a wide range of real-time systems:

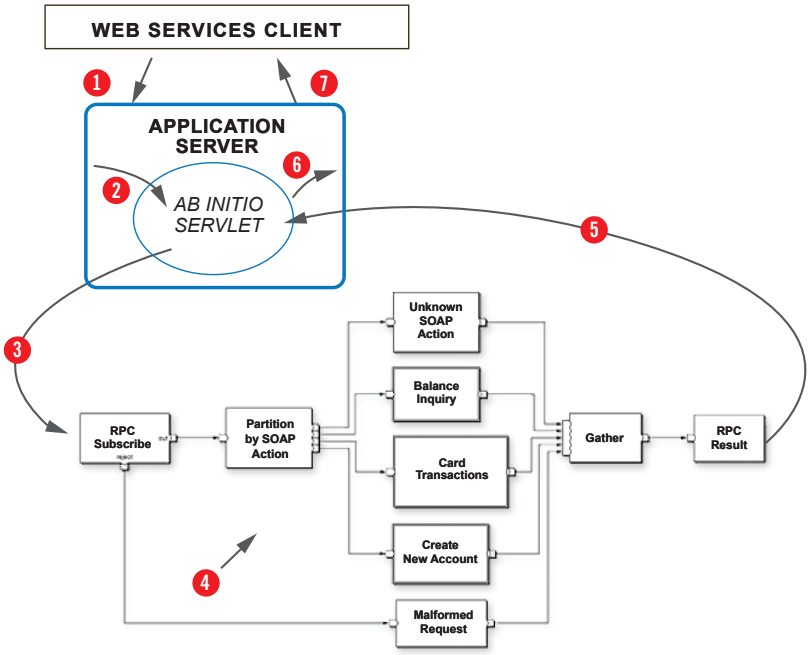
- 3rd party queuing systems: IBM MQ, JMS, TIBCO Rendezvous, and Microsoft MQ. Ab Initio provides components for directly connecting to all of these queuing systems

- Web services: WebSphere, WebLogic, IIS, and Apache/Tomcat
- Ab Initio queuing and RPC for low-latency and high-volume point-to-point connections between Ab Initio applications
- Legacy / in-house messaging software

NATIVE SUPPORT FOR WEB SERVICES AND SOA

Ab Initio applications can easily implement web services in a service-oriented architecture (SOA). This is accomplished through an Ab Initio-provided servlet that is installed in a standard application server (WebSphere, WebLogic, IIS, or Apache/Tomcat) of the customer's choosing. The servlet provides a registration mechanism for associating particular web service calls with specific Ab Initio applications. When a web service call comes in, the servlet communicates with the Ab Initio application via TCP (the Ab Initio application runs in its own set of processes and is outside the application server) and returns to the original requestor the results returned by the Ab Initio application.





The Co>Operating System also provides full support for parsing messages defined via WSDL.

INTERFACING WITH SPECIAL PURPOSE AND LEGACY MESSAGING SYSTEMS

Commercial queuing products and web services are relatively new to the industry, and their performance rates are modest. Customers with large messaging volumes, or whose needs pre-date the existence of commercial queuing products, have built their own in-house high-performance messaging solutions.

Continuous>Flows supports robust interfacing to these legacy solutions through special processing components ("Universal Subscribe" and "Universal Publish"). These components call custom C++ subroutines that interface with the legacy messaging system. The components also handle rollback and recovery in the event of failures.

Ab Initio, in concert with special-purpose messaging systems, can achieve extremely high performance rates – sustained rates of over 500,000 messages per second in mission-critical applications have been measured.

Furthermore, the Universal Subscribe and Universal Publish components are used in just the same way as Continuous>Flows components for 3rd party queuing products. This provides users with the option of switching from their in-house queuing solution to a 3rd party queuing product with minimal changes to their Ab Initio applications.

ROBUSTNESS IN THE EVENT OF FAILURES

The Co>Operating System checkpointing facility provides robust handling of application failure. A checkpoint allows an application to commit changes to multiple databases and input and output systems (queues). In the event of an application failure, the Co>Operating System does a "rollback" of the environment back to the last successful checkpoint. Once the underlying problem has been cleared up (database refuses to load, out of disk space, network failure, ...), the application can be restarted, and it will automatically pick back up after the last successfully processed transaction.

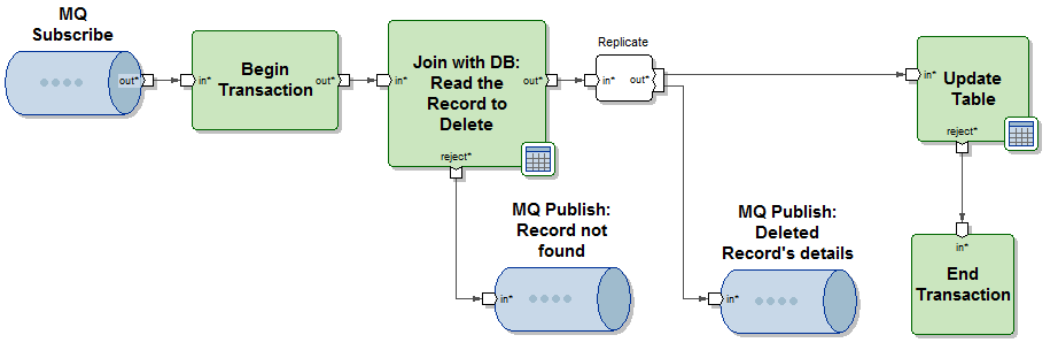
Most checkpointing schemes are notoriously expensive from a computational perspective, and developers' efforts to try to minimize that cost often result in complex, unstable applications. The Co>Operating System was architected both to have extremely high performance and to be robust. The Co>Operating System provides a number of checkpointing alternatives that trade off transaction latency against recovery time. In all cases, the Co>Operating System guarantees that all transactions will ultimately be written once and only once to all target devices (databases, files, and queues).

The Co>Operating System provides two basic mechanisms for checkpointing. The best known is the XA standard protocol for 2-phase-commit. The Co>Operating System includes a transaction manager that will coordinate commits across disparate databases and queuing products, and can even batch transactions into a single commit to increase throughput.

However, XA has its limitations: it has high computational overhead; it is complex to administer; and it is not supported by all desired input and output devices. As a consequence, most Ab Initio

users depend on the Co>Operating System's native checkpointing mechanism, which is very much like 2-phase-commit. This mechanism works with all input and output devices (databases, files, and queues) to which Ab Initio connects, works across heterogeneous and distributed servers, and is very performant and extremely robust. Ab Initio has even built into its connector components ways of compensating for limitations of certain 3rd party queuing products – for example, in certain products a queue manager crash can result in transactions being delivered more that once.

With the Co>Operating System's native checkpointing system, developers can control the frequency of checkpoints in a number of ways, such as the number of transactions and elapsed time, or as a result of an event such as a special token in the message stream. Furthermore, they can control the degree of transactional consistency at checkpoint time across multiple output devices. Default settings yield very high performance; transactions are never lost; and the correctness of the application is never sacrificed.



OPERATIONAL ROBUSTNESS

The Co>Operating System seriously addresses operational requirements for mission-critical, real-time applications. Some example mechanisms include:

- Running multiple instances of a real-time application simultaneously for load-balancing and failover
- Bringing down pieces of an application system so that updated modules can be initiated without suspending a 7×24 nonstop system
- Pooling connections to databases to limit resource usage
- "Folding" multiple components into a single process to lower CPU overhead and memory footprint
- Using "micrographs" — dynamically loadable graph logic — to dramatically reduce the use of operating system resources

CONCLUSION

The Continuous>Flows approach to real-time processing brings together the productivity increases of graphical dataflow implementation, a truly general model for connecting to data streams, and robust mechanisms for reliably checkpointing work and coordinating transactions.

BUSINESS RULES ENVIRONMENT:
Putting Business Users in the Driver's Seat

Ab Initio serves a wide range of users. At one end of the spectrum are professional application developers. These folks understand how to design and build sophisticated systems that process large amounts of data in complex environments. For these users, Ab Initio provides the Graphical Development Environment (GDE) for graphically building complex processes and business logic. The resulting applications run on the extremely robust and scalable Co>Operating System.

At the other end of the Ab Initio spectrum are business users. These people are not skilled at or even interested in the mechanics of processing terabytes of data or thousands of messages per second. However, they do know what their systems are supposed to accomplish and often best know the details of the data and the rules that drive the business.

Over the years, a division — sometimes an adversarial one — has evolved between IT teams and the business users they serve. The business users know what they want done and what it means to the business, and the IT professionals know how to do it. The IT shop asks for specs and the business users do their best to provide them. But there are inevitable ambiguities and mistakes – mistakes in the specs and mistakes in the translation of the specs into code. The process – from spec to code to test to release — never works as cleanly or as quickly as everyone hopes. At each stage, one human has to figure out what another human meant. That takes time and, worse, introduces big opportunities for errors. And each error (of course) takes more time to find, revise in the spec, and change in the code. Productivity suffers. Correctness is never completely assured. Sometimes the cycle never converges — and projects simply fail.

What if business users had a way to control the specific parts of the systems that implemented their business logic? What if their business rules could be expressed in a way that they understood, and that they could even write, and that could be automatically converted to code that ran inside a larger system? Sounds pretty good? That's the whole idea behind the Business Rules Environment.

The Ab Initio Business Rules Environment (BRE) allows business analysts to specify business rules in a form that is very familiar and comfortable to them: grid-like spreadsheets. In the BRE, the rules are specified in business terms, not technical terms, and with expressions that are clear to anyone who has worked with Microsoft Excel. As a consequence, not only can rules be specified quickly and accurately, they are also easily understood by other business people.

Furthermore, the BRE puts the business users in the driver's seat when it comes to verifying the business rules. The business users are not only able to put the rules directly into the system, they are also able to immediately see the results of applying those rules to test data. If they don't like what they see, they can change the rules on the spot. The savings in time are enormous.

THE AB INITIO BUSINESS RULES ENVIRONMENT IS NOT A STANDALONE TOOL

Traditional "rules engine" products are standalone tools. They require work to interface to other products and the rest of the computing infrastructure. They have performance and scalability limitations. They can process only simple data structures. Their perspective is limited to the rules they process, so they cannot trace lineage across entire systems.

The Ab Initio BRE solves all these problems and limitations, and many more, because it is tightly coupled with Ab Initio's Co>Operating System and metadata technologies. It was architected and implemented that way — this is not an afterthought or the result of a typical software company's marketing strategy.

- The same BRE rules can be run without reimplementation in batch, continuous, web services, and real-time applications.
- BRE rules run extremely efficiently and can be scaled to run across many CPUs and many servers. There is no limit to the scalability of systems built with Ab Initio software.
- BRE rules can process complex legacy hierarchical data. Whatever may exist in a legacy environment (EBCDIC, packed decimal, international formats, XML, COBOL copybooks, ...) can be handled natively by the BRE in conjunction with the Co>Operating System.
- The same BRE rules run identically on all platforms supported by the Co>Operating System (Unix, Linux, Windows, z/OS, and z/Linux).
- The BRE benefits from all of the Co>Operating System's robustness features, such as error handling, checkpointing, and so on.

WHAT ARE "BUSINESS RULES"?

The BRE supports three different styles of business rules: decision rules, validation rules, and mapping rules. While they are fundamentally similar, business users are comfortable thinking of rules as belonging in one of these categories.

As required by the nature of the business, rules can be simple and short or extremely long and complex. The BRE handles both extremes. Currently, the largest BRE rulesets in production have more than 50,000 rules.

Here is an example of a very simple decision-style business rule specified in the BRE. This is the tax computation for U.S. income taxes (form 1040):

	Triggers		Outputs
	Filling status	Taxable income line 43	Tax (line 44)
1	Single	<= 100000	Tax from Table, Single (Taxable income line 43)
2		> 100000 and <= 171550	Taxable income line 43* 0.28-6280.00
3		> 171550 and <= 372950	Taxable income line 43* 0.33-14857.50
4		> 372950	Taxable income line 43* 0.35-22316.50
5	Married filing jointly or Qualifying widow(er)	<= 100000	Tax from Table, Jointly (Taxable income line 43)
6		> 100000 and <= 137050	Taxable income line 43* 0.25-7625.00
7		> 137050 and <=208850	Taxable income line 43* 0.28-11736.50
8		>208850 and <= 372950	Taxable income line 43* 0.33-22179.00
9		> 372950	Taxable income line 43* 0.35-29638.00
10	Married filing separately	<= 100000	Tax from Table Separately (Taxable income line 43)
11		> 100000 and <= 104425	Taxable income line 43* 0.28-5868.25
12		> 104425 and <=186475	Taxable income line 43* 0.33-11089.50
13		>186475	Taxable income line43* 0.35-14819.00
14	Head of household	<=100000	Tax from Table, Household (Taxable income line 43)
15		> 100000 and <= 117450	Taxable income line 43* 0.25-5147.50
16		>= 117450 and <=190200	Taxable income line 43* 0.28-8671.00
17		>= 1900200 and <= 372950	Taxable income line 43* 0.33-18181.00
18		>= 372950	Taxable income line 43* 0.35-2564.00
*			

This rule has two inputs, Filing status and Taxable income line 43, and computes a single output named Tax (line 44). In the BRE, there is an implicit AND operator between columns and an implicit ELSE operator between rows. So the first few rows of this rule read as follows:

- IF the Filing status is Single AND the Taxable income line 43 is less than or equal to 100000, THEN the Tax (line 44) is equal to looking up the Taxable income line 43 in the Tax from Table and returning the amount for Single filers.
- ELSE IF the Filing status is Single AND the Taxable income line 43 is greater than 100000 and less than or equal to 171550, THEN the Tax (line 44) is equal to the Taxable income line 43 times 0.28 minus 6280.00.
- ELSE IF the Filing status is Single AND the Taxable income line 43 is greater than 171550 ...
- ... and so on ...

Next is an example of a validation rule. It validates multiple inputs in the same rule, and sets multiple outputs in each case:

	Triggers		Outputs		
	One Way Flag	If true	Error Code	Disposition	Replacement Value
1	any	missing (Starting Airport) or missing [Airport Lookup[Starting Airport])	Missing Required Field	Reject Record	–
2	any	missing (Destination Airport] or missing [Airport Lookup(Destination Airport])	Missing Required Field	Reject Record	–
3	any	Starting Airport=Destination Airport	Destination is Same City	Reject Record	–
4		missing (Departure Date)	Missing Required Field	Reject Value	today()
5	round trip	missing (Return Date)	Missing Required Field	Reject Record	_
6	round trip	Return Date < Departure Date	Invalid Return Date	Reject Value	Departure Date
7	round trip	missing (Return Date)	Invalid Return Date	Reject Value	NULL
8	any	missing (Number of Passengers) or Number of Passengers<1	Missing Required Field	Reject Value	1

And here is an example of a source-to-target mapping rule. For this style of rule, the BRE displays on the left a list of the potential input values (the "Inputs" column) such as fields, variables, constants and calculated values. In the middle is a list of the target fields ("Output Name" column). To the right of each target field is a column ("Expression/Rule") where the user can drag in values from the "Inputs" column or create an expression for calculating that output. (We'll see what the "Computed Value" column is all about later.)

Inputs		Output Name	Expression / Rule	Computed Value
custid (100D047)	1	Customer_EDW_Metadata.record	'I'	I
first (KEVIN)	2	Customer.CustomerID	custid	100D047
middle (TRAVIS)	3	Customer.PrimaryPhone	string_relace(phonenum,"=","")	5857768002
last (CIESZVINSKI)	4	Customer.OutstandingBalance	0	0
street (9265 OAK RD	5	Customer.CustfirstName	first+' '+middle	KEVIN•TRAVIS
city (WHITEHALL)	6	Customer.EmailAddress	email	kenvin_cieszvinski@hotmail.com
state (NY)	7	Customer.CustomerSince	null	NULL
zipcode (12887)	8	Customer.PreferredFlag	null	NULL
statename (NewYork	9	Customer.CustLastName	null	NULL
▼ sex (M)	10	Customer.HomeOwnerFlag	null	NULL
• Male	11	CustPerson.EDW_Metadate.reco-	null	NULL
• Female	12	CustPerson.CustomerID	null	NULL

BRE RULES CAN INCLUDE ARBITRARILY COMPLEX LOGIC

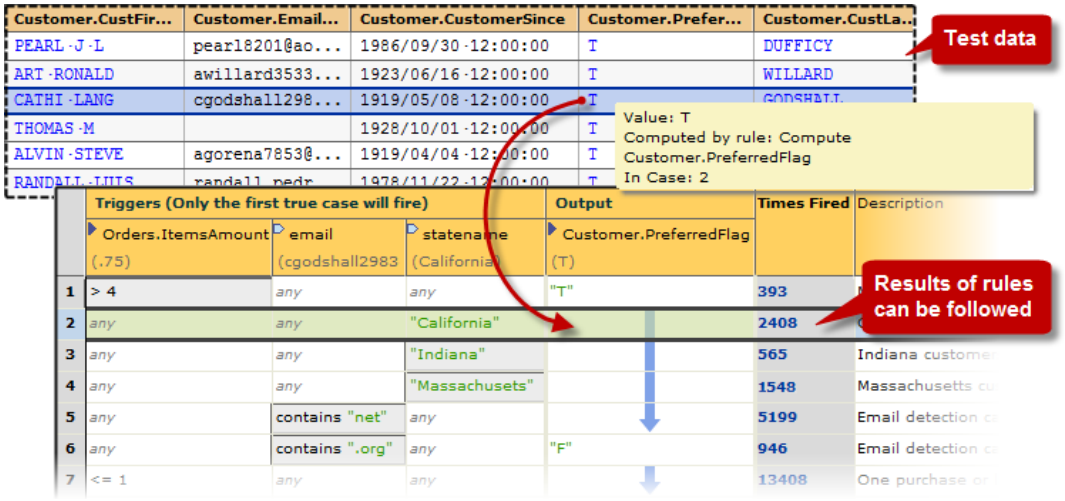
The logic inside BRE rules can be simple, as in the examples above. However, in real life, business users often have very complex rules. With non-Ab Initio technologies, those rules cannot be implemented in the business rules product. Instead, they require hand-coding in programming languages such as C++ or Java. This can have a huge negative impact on usability and productivity. It reintroduces the whole spec-interpret-code-test-fix cycle that dooms projects to failure.

Not so with the Ab Initio BRE. The BRE inherits the complete data processing capability of the Co>Operating System. This means that all Co>Operating System functions — and there are hundreds — are available to the BRE, as well as complex logic structures.

As a result, there is no limit to the size or complexity of the expressions within rules, or the size or complexity of the rules, or the number of rules within a ruleset.

BUILT-IN TESTING ALLOWS BUSINESS USERS TO VERIFY RULES — BY THEMSELVES

A key advantage to using the Ab Initio Business Rules Environment to develop business rules is the BRE's built-in testing capability: rules developed within the BRE can be executed against sample data without ever leaving the editing environment. During testing, the BRE reports the outputs computed for each test record, as well as the states of every intermediate variable and calculation. A BRE user can inspect output records, and then click on any computed value to see exactly how and why that value was computed:



For each test record, the BRE shows the case(s) that triggered (in yellow), and also darkens any cells that resulted in a failed comparison. The BRE also calculates the number of times each particular rule is triggered ("Times Fired" in the screen shot). This information can be used to find missing test cases or invalidly constructed rules (if a case never triggers).

Testing can be run at any time during rule development, allowing for an incremental approach to the creation of a large set of rules. There is no need to wait until the rules are all entered before starting to evaluate behavior.

The output of a test run can be saved as a benchmark against which later test runs are automatically compared. When rules are changed, either during development or when modifications are being designed, it is possible to see all the differences between the current behavior of the modified rules and the saved benchmarks results. This feature is invaluable when evaluating the impact of a proposed change. Without the Ab Initio BRE, business users are often "flying blind" when changing their rules.

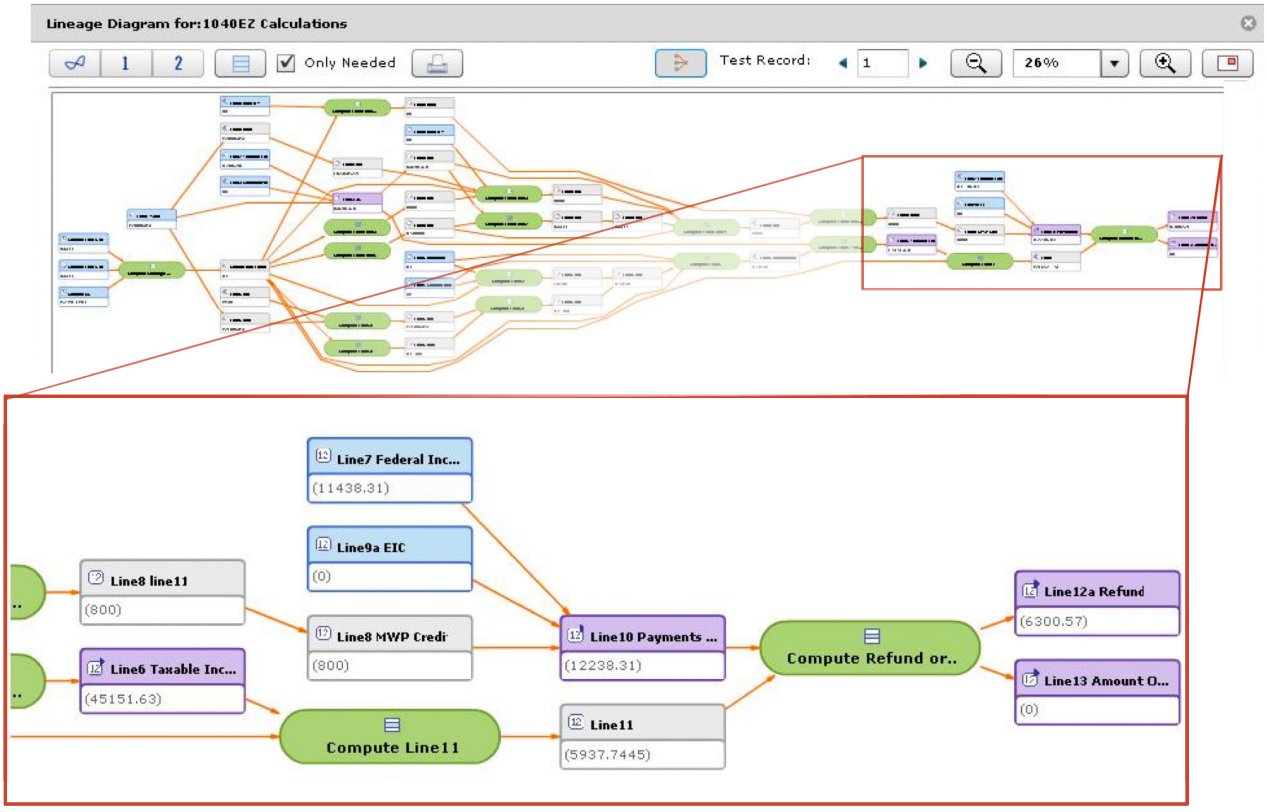
Furthermore, rule execution details are also available from rulesets running in production. Users can configure the amount of tracking information that is saved, ranging from just information about which rule cases (rows) are triggered for every input record to all input, lookup, intermediate, and final values. This allows analysts to study the behavior of the rules over time, often leading to improvements in results by adjusting the business logic. These details can also be critical to answering questions about how specific decisions were made in the past.

TRANSPARENCY — BUSINESS LOGIC EXPOSED, AT LAST

Large, complex business rules systems can consist of many rules that have been implemented on top of other rules. Understanding how these rules interrelate and how data flows through them is critical for business users. Unfortunately, typical business users can rarely see how their systems work.

With the BRE, business users have complete visibility into their systems, because the BRE displays diagrams of how data flows through interconnected rules, irrespective of how large and complex those rules are. Furthermore, graphical lineage can be displayed for applications that consist of many rulesets that may be distributed across an application, or even across multiple applications.

The following figure shows a simple lineage example for the ruleset shown above that calculates income taxes based on the US 1040 EZ form. Green ovals represent rules (calculations) and rounded rectangles represent variables. In testing mode, sample test values are shown below each variable name, along with all intermediate calculations. Below the full lineage diagram is a zoomed section showing how the calculation of the deductible (line 5) impacts the final refund or amount owed.



The lineage ability of the BRE allows business users to truly understand how large sets of complex rules work.

NO "RETE ALGORITHM"

What's RETE? Don't worry — with the BRE, you don't have to know. With traditional rules products, however, rules are executed in an order determined by something called the "RETE algorithm." This means that a "business" person has to understand a fairly complex computer-science / artificial intelligence concept (see http://en.wikipedia.org/wiki/Rete_algorithm if you really want to know).

In practice, the RETE approach makes it very difficult to understand the consequences of a change to a rule, since it may impact other rules that are very far away in the specifications. Performance is also very difficult to tune, since small changes may have big, unintended consequences.

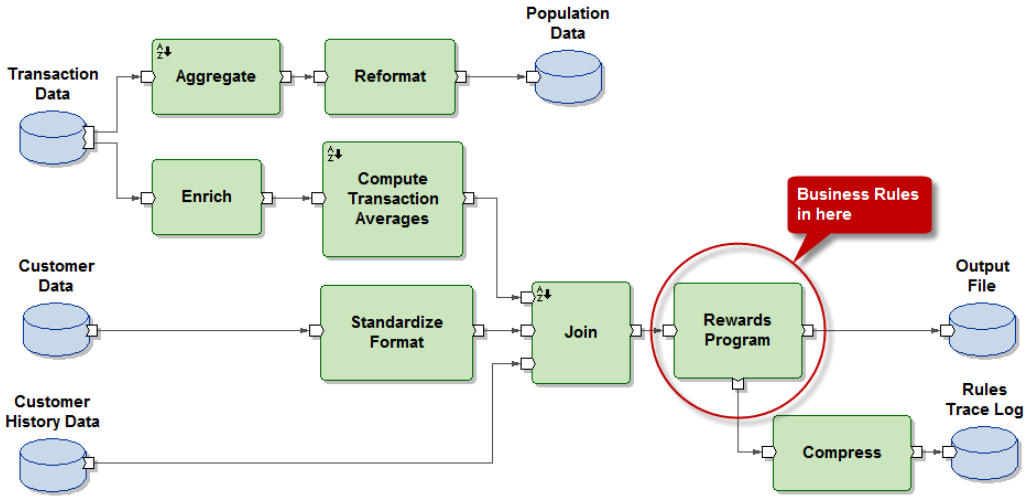
In the BRE, rules run in the order in which you specified them. Performance is not only substantially higher, it's predictable! You don't have to be a computer scientist to figure out how rules work.

HOW DOES THE BRE WORK WITH THE CO>OPERATING SYSTEM?

It's straightforward: the BRE takes the rules created by the user and puts them into a component in a graph run by the Co>Operating System.

Because business rules run in the Co>Operating System, they inherit all the advantages and strengths of the Co>Operating System. Rules can interface to any data source or target, can process any kind of data, can handle any volume of data, can run in batch and/or real-time, can run distributed across multiple servers running different operating systems, all in a completely robust way. The Co>Operating System includes everything for building and running robust mission-critical applications, and the BRE inherits all of this.

Below is an example of a Co>Operating System graph (application) in which a key component contains rules specified with the BRE:



COMPLETE RULES MANAGEMENT AND GOVERNANCE

All rules are stored in the Ab Initio Enterprise Meta>Environment (EME), which has complete source-code management capabilities, including versioning. As a consequence, BRE users have the choice of two deployment methodologies:

- Treat rules like other application elements. This means they must go through all the standard code promotion stages: development, testing, and finally promotion to the production environment. While this robust methodology is very familiar to IT development staff, the number of steps can take a lot of time. Nevertheless, the BRE's integrated testing streamlines and shortens the overall process.
- Treat rules more like reference-code data in an operational database. Business users are often able to make changes (adds, updates, and deletes) to reference codes without running those changes through the complete application-code promotion process, thereby allowing those users to respond quickly to business needs.

Ab Initio does not promote one approach over the other; we just make it possible for our users to choose the approach that best fits their needs.

Finally, because of the EME's strong support for versioning, users can easily configure their environment to enable "champion/challenger" deployment of rules. The idea is to run two versions of the same rules side by side to compare and contrast their outputs. Users then establish processes to review these differences and improve their rules until they get the desired results.

UNITING THE ENTERPRISE WITH UNIFIED TECHNOLOGY

With integrated, interactive testing and automatic generation of production-ready, executable modules, the BRE allows business users to participate directly in the development of applications in a way that complements the expertise of IT staff. By presenting business logic in a way that business users can express and understand, the BRE removes huge amounts of uncertainty and risk surrounding the development and deployment of new applications.

BRE rules and the applications driven by them are executed by the Co>Operating System, which means they can be arbitrarily complex, yet run extremely efficiently, and with complete scalability. And the same rules can be run in batch applications and web service applications without recoding. Every benefit of the Co>Operating System, in fact, is realized through the BRE. Why? Because the BRE was built with the same, unified technology as the Co>Operating System. The BRE and Co>Operating system work together, seamlessly, to unite the enterprise.

DATA QUALITY: An End-to-End Solution

Data quality matters. Data quality problems can have a significant impact on a company's bottom line. Bad data can result in redundant work and missed opportunities. Data quality problems can accumulate, increasing in scope and impact, as data moves through the enterprise. In the worst cases, this can cause executives to reach incorrect conclusions and make bad business decisions. Pretty serious stuff. Yet most companies have no formal data quality programs that can measure and mitigate data quality problems. Most companies are not even aware that they have a data quality problem.

The solution is to institute an enterprise data quality (DQ) program. By its very nature, an enterprise DQ program is beyond the capabilities of any single canned solution. DQ requires a holistic approach – with touchpoints throughout the business and implemented across a range of technologies. DQ should be an integral part of the data processing pipeline and should not be limited to just offline, retrospective analysis. DQ is not just about customer name and address cleansing. It's about the consistency and representation of all enterprise information.

If the technologies used for DQ are to be part of the processing pipeline, they have to be production-level robust. They have to deal with complex legacy data, real-time transactions, and high, sustained processing volumes. Approaches that do not meet all these requirements end up being relegated to offline deployments and rarely meet expectations. This is what typically happens with special-purpose niche DQ tools that specialize in certain types of data and that can be used only in limited circumstances.

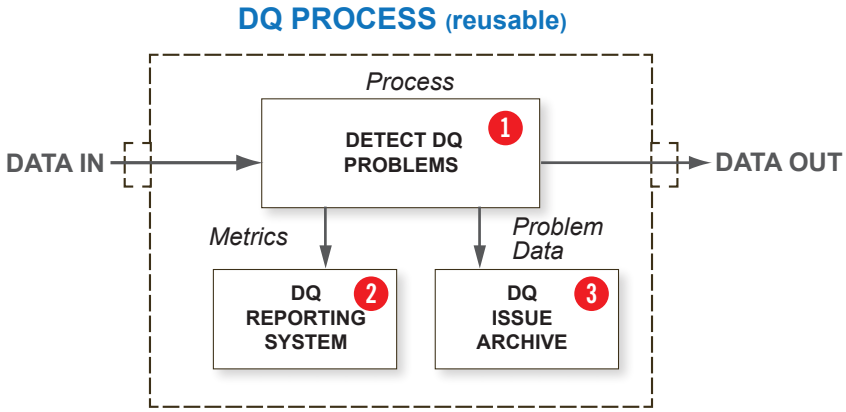
Ab Initio's approach to data quality is different — it is end-to-end. Because the Ab Initio Co>Operating System is a complete application development and execution environment, Ab Initio's approach to data quality works anywhere the Co>Operating System can be deployed, which is in practically any operational or analytics environment. The Co>Operating System natively processes complex legacy data, runs distributed across heterogeneous sets of servers, is very high performance and completely scalable, and can implement highly complex logic.

Ab Initio's end-to-end approach to data quality is based on design patterns using Ab Initio's seamlessly coupled technologies — they are all architected together — including the Co>Operating System, the Enterprise Meta>Environment (EME), the Business Rules Environment (BRE), and the Data Profiler. Using Ab Initio, a company can implement a complete data quality program that includes detection, remediation, reporting, and alerting.

ARCHITECTURAL OVERVIEW

When it comes to DQ, one size does not fit all, especially for large organizations with many legacy systems. Ab Initio, therefore, provides a series of powerful building blocks that allow users to put together custom data quality solutions that meet their specific needs, whatever those needs might be. For users who are just starting to put a data quality program in place, Ab Initio supplies a reference implementation that can serve as the foundation of a complete program. For users who have different needs, or who already have pieces of a data quality program in place, Ab Initio's DQ building blocks can be plugged together with existing infrastructure as desired.

A typical data quality implementation starts with constructing a powerful, reusable DQ processing component with the Co>Operating System, as shown below:

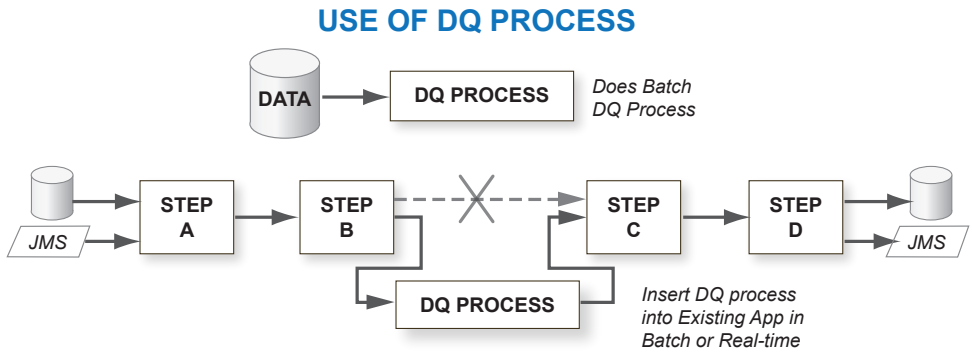


The Co>Operating System enables components to contain whole applications. This particular reusable DQ process component is an application in its own right, and includes the following:

- A subsystem that detects and possibly corrects data quality problems. The Co>Operating System serves as the foundation for implementing defect detection. The BRE can be used to specify validation rules in an analyst-friendly interface, and the Data Profiler can be integrated into the process for trend analysis and detailed problem detection.
- A data quality reporting system. The EME includes built-in data quality reporting that integrates with the rest of an enterprise's metadata, data quality metrics and error counts, and data profile results. Users can extend the EME schema to store additional data quality information and to augment the base EME capabilities with their own reporting infrastructure.

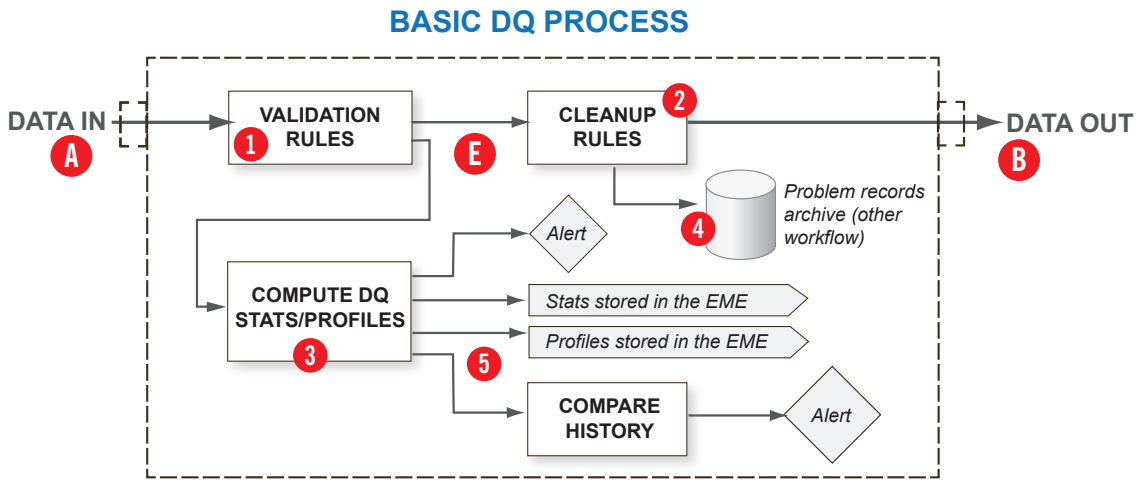
- An issue-reporting database. Records that have data quality issues are logged in a database or file so they can be examined as part of a complete data quality workflow. Ab Initio provides the technology to store, retrieve, and view those records, although users are free to select any data storage technology that meets their needs.

This DQ processing component is typically run as part of existing applications. If an application has been built with Ab Initio, the DQ component can easily be plugged into it. For applications not built with Ab Initio, the DQ processing component has to be explicitly invoked. The DQ component can also be implemented as an independent job that sources data directly. Below are examples of both deployment cases, standalone and integrated into an existing application:



DATA QUALITY PROCESSING WORKFLOW

The diagram below illustrates a sample complete data quality detection workflow. It is important to remember that each DQ deployment is tailored to the user's specific needs.



As indicated earlier, the input to this DQ Process (A) can be any type of data from any source. It can be a flat file, a database table, a message queue, or a transaction in a web service. It can also be the output of some other process implemented with Ab Initio or another technology. Because the DQ Process runs on top of the Co>Operating System, the data can be anything the Co>Operating System can handle: complex legacy data, hierarchical transactions, international data, and so on.

The output of the DQ Process (B) can also be any type of data going to any target.

The first step is to apply Validation Rules (1) to the data. Validation rules can be run against individual fields, whole records, or whole datasets. Since each record may have one or more issues, the validation rules may produce a set of DQ issues on a per-record basis (E). The severity of these issues and what to do about them is decided further downstream.

Next, cleanup rules are applied to the data (2), and the output is the result of the DQ Process (B). Users may use built-in Ab Initio cleansing rules or build their own with the Co>Operating System. While validation and cleansing rules are easily entered with the BRE, there is no limit to the sophistication of these rules since they can use the full power of the Co>Operating System's data processing.

Records that cannot be cleansed are output to a Problems Archive (4). These problem records then typically go through a human workflow to resolve their issues.

The list of issues for each record (E) may also be analyzed (3) to generate reports and alerts (5). Because this process is built using standard Ab Initio "graphs" with the Co>Operating System, practically any type of reporting and processing can be done. Ab Initio's standard DQ approach includes:

- Calculating data quality metrics, such as completeness, accuracy, consistency, and stability
- Determining frequency distributions for individual fields
- Generating aggregate counts of error codes and values
- Comparing current values for all the above with historical values
- Signaling significant deviations in any of the current measurements from the past

All the information generated above is stored in the Ab Initio EME for monitoring and future reference. All DQ information can be integrated with all other metadata, including reference data that is also stored in the EME.

While all the computation associated with these steps may consume significant CPU resources, the Co>Operating System's ability to distribute workload across multiple CPUs, potentially on multiple servers, allows full data quality processing to always be part of the processing pipeline.

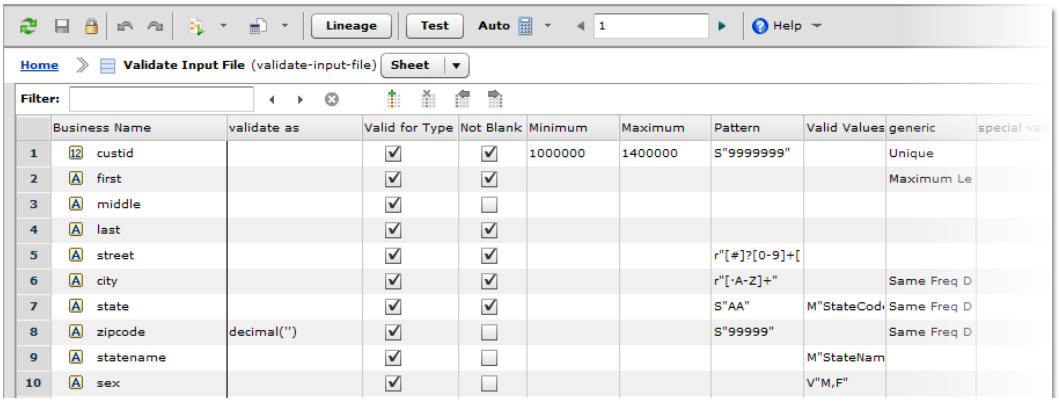
As demonstrated above, Ab Initio's approach to data quality measurement includes a rich set of options that can be customized and configured to meet a user's needs. The processing of the data, calculation of results, and all the steps in between are implemented using the Ab Initio Co>Operating System. This means that data quality detection can be run on almost any platform (Unix, Windows, Linux, mainframe z/OS), and on any type of data, with very high performance. In situations where large volumes of data are being processed, the entire data quality detection process can be run in parallel to minimize latency.

The next several sections present examples of the analyst-friendly user interfaces for creating validation rules and reporting on data quality results.

VALIDATION RULES

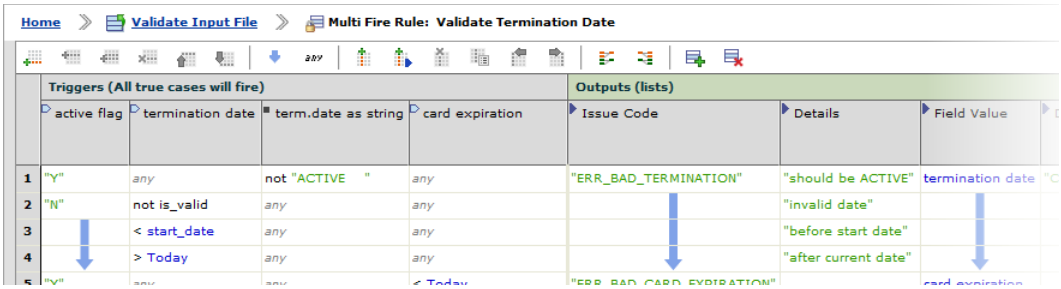
Most data quality issues are detected by applying validation rules to the source dataset. With the Ab Initio data quality design pattern, record-at-a-time validation rules can be defined using the Ab Initio Business Rules Environment (BRE). The BRE is designed to allow less technical users, subject matter experts, and business analysts to create and test validation rules using a spreadsheet-like interface.

Using the BRE, there are two ways to define validation rules. In most cases, users define rules by filling out a simple spreadsheet (validation grid) with field names down the left side and validation tests across the top:



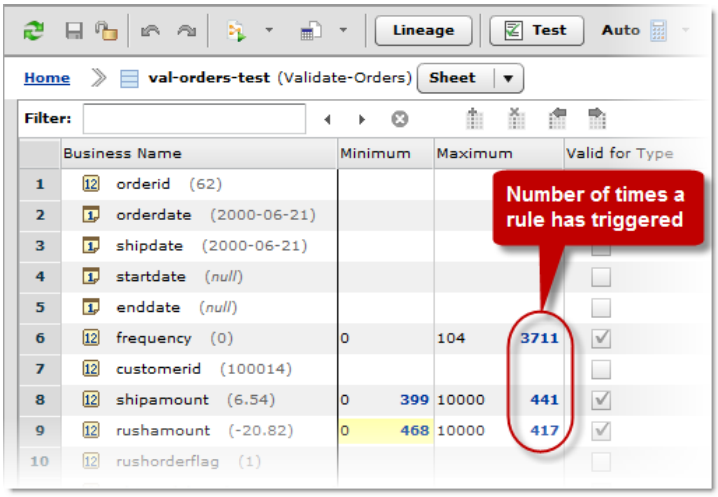
This interface makes it very easy to specify which validation tests should be applied to each field or column in a dataset. The BRE includes a number of built-in validation tests (nulls, blanks, value ranges, data formats, domain membership, etc.). But it is also possible for the development staff to define custom validation tests that can be applied to individual fields. Custom validation tests are written by developers using the Ab Initio Data Manipulation Language (DML), and then made available in the BRE.

For more complex validation rules, the BRE allows for the definition of "tabular rules." These complex validation rules may process multiple input fields within a record to determine whether there are data quality issues. Each rule can produce an error and disposition code, which together drive the amelioration process.



The BRE enables subject matter experts to design, enter, and test validation rules, all from the same user interface. The BRE's testing capability allows users to interactively see which rules trigger for various inputs. This makes it easy to ensure that the rules are behaving as expected.

The screen shot below shows validation rules during testing. The BRE displays trigger counts for every validation test, as well as the details for each test record.



Validation rules are saved in the EME, which provides for version control, access control, and configuration management. For applications that are built entirely with Ab Initio, including the DQ process, the application and the DQ rules are versioned, tagged, and promoted into production together. This ensures a robust DQ process.

While the BRE makes it easy for less technical users to define validation rules, it is not the only way to define such rules. The full power of the Co>Operating System's transformation technology is available for implementing the most complex rules. Because the BRE and transformation rules both run on top of the Co>Operating System, it's possible to create a very comprehensive data quality measurement strategy.

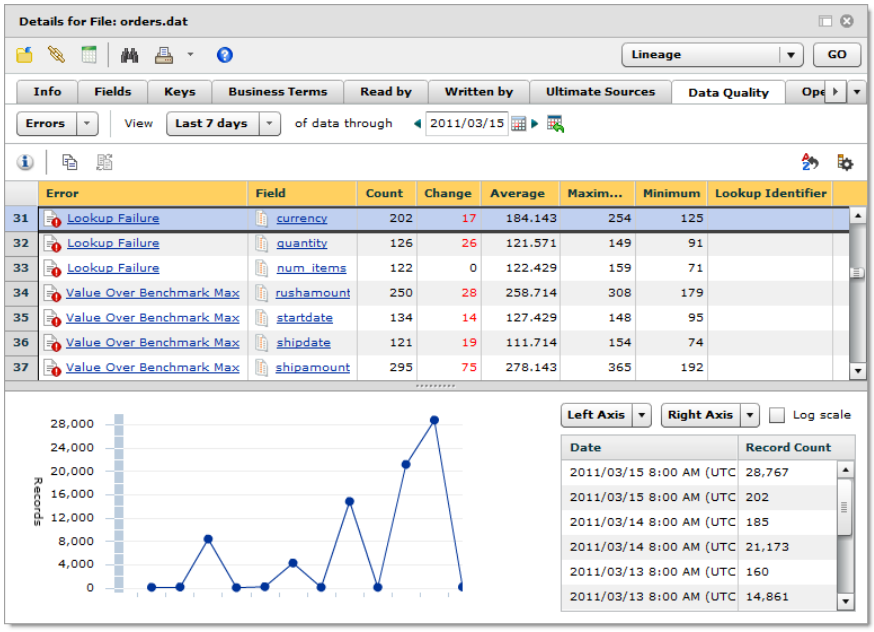
REPORTING

Detection is the first part of a complete data quality implementation. The second major component of a data quality program is reporting.

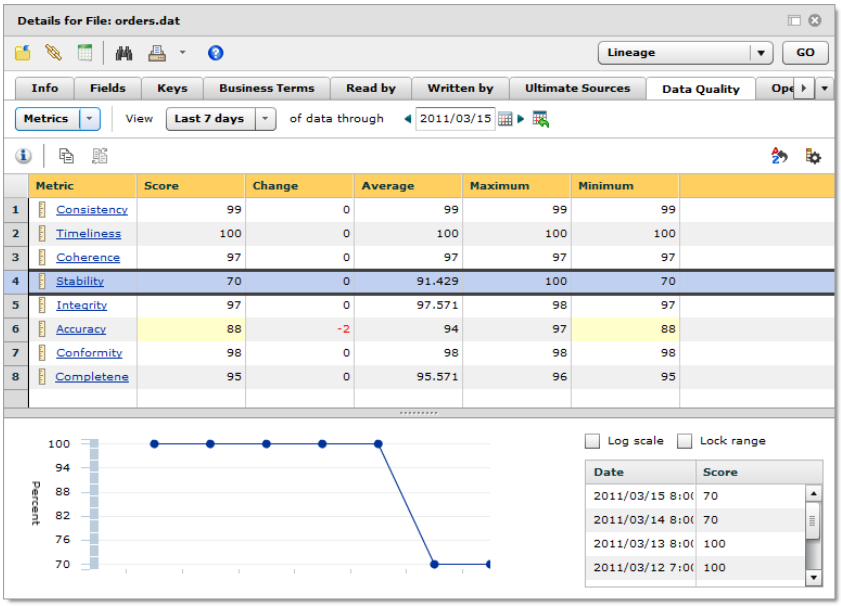
Data quality reporting is driven by the Enterprise Meta>Environment (EME). Ab Initio's EME is an enterprise-class and enterprise-scale metadata system architected to manage the metadata needs of business analysts, developers, operations staff, and others. It handles many types of metadata from different technologies in three categories — business, technical, and operational — and this metadata includes data quality statistics.

Ab Initio stores data quality statistics in the EME for reporting purposes. One type of DQ information stored in the EME is the aggregated counts of error codes (issues) of individual fields and datasets. The counts are linked to the dataset being measured and to the fields with issues. Issues are aggregated and reported by error code, which is in a global set of reference codes, stored in the EME (the EME supports reference code management).

The screen shot below shows the EME's ability to display field-level issues along with historical trending graphs. Counts that exceed configurable thresholds are highlighted in yellow or red.

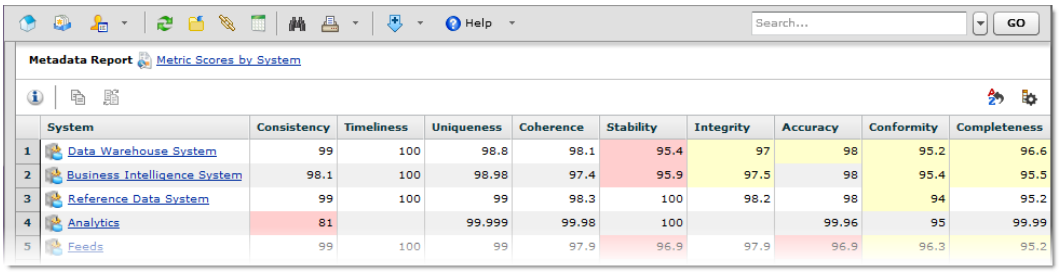


As shown below, Ab Initio is able to calculate data quality metrics for datasets and fields (columns), and these too are stored in the EME. There is a corresponding tabular report of these metrics, which includes trending graphs and yellow/red thresholds.



When data quality measurements are captured throughout a large environment, it is possible to aggregate the information according to the user's organizational structure. This makes it possible for managers to view data quality metrics for entire systems, applications, and/or subject areas in one report. From this report, problem areas can then be investigated by drilling down into the details.

The screen shot below shows a number of higher-level subject areas and their aggregated data quality metrics:



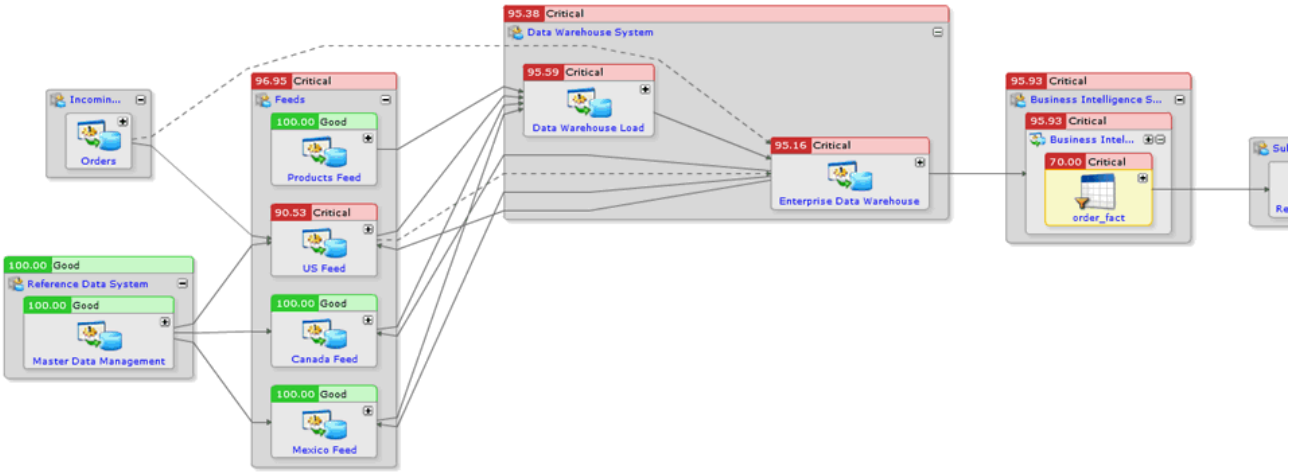
	System	Consistency	Timeliness	Uniqueness	Coherence	Stability	Integrity	Accuracy	Conformity	Completeness
1	Data Warehouse System	99	100	98.8	98.1	95.4	97	98	95.2	96.6
2	Business Intelligence System	98.1	100	98.98	97.4	95.9	97.5	98	95.4	95.5
3	Reference Data System	99	100	99	98.3	100	98.2	98	94	95.2
4	Analytics	81		99.999	99.98	100		99.96	95	99.99
5	Feeds	99	100	99	97.9	96.9	97.9	96.9	96.3	95.2

REPORTING: LINEAGE

Many users roll out a data quality program by implementing data quality detection across a number of datasets in a single system. For example, it is not unusual to see data quality measured for all the tables in an enterprise data warehouse, but nowhere else. Although measuring data quality in one system is better than not measuring data quality at all, a more useful data quality program includes data quality checks at multiple stages across the entire enterprise processing pipeline. For example, data quality might be measured at the enterprise data warehouse, but also at the system of record, at intermediate processing points, and downstream in the various data marts or extract systems. Each of these systems can capture quality metrics whether or not they were built with Ab Initio.

The EME multiplies the value of a data quality program when data quality measurements are made at multiple points in an enterprise. This is because the EME can combine data lineage with data quality metrics to help pinpoint the systems in which — and precisely where — data quality problems are being introduced.

Consider the following screen shot:



This screen shot shows an expanded lineage diagram in the EME. Each large gray box represents a different system. The smaller green, red, and gray boxes represent datasets and applications.

Data quality metrics may flag individual elements. Green is good. Red indicates a data quality problem. With these diagrams, it is easy to follow the path of data quality problems, from where they start to where they go. For the first time, management can actually see how data and problems are flowing through their environment.

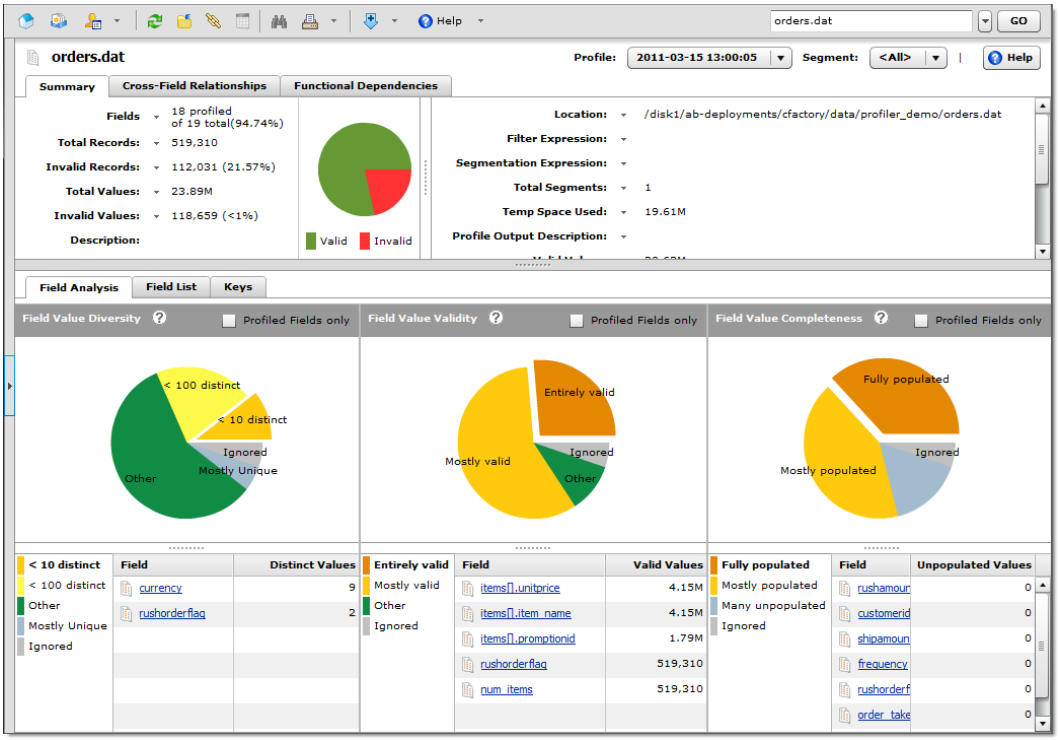
Finally, DQ reporting is not limited to the built-in EME screens. The EME's information is stored in a commercial relational database, and Ab Initio provides documentation on the schema. Users are free to use business intelligence reporting tools of their choice to develop custom views of their enterprise data quality.

REPORTING: DATA PROFILER

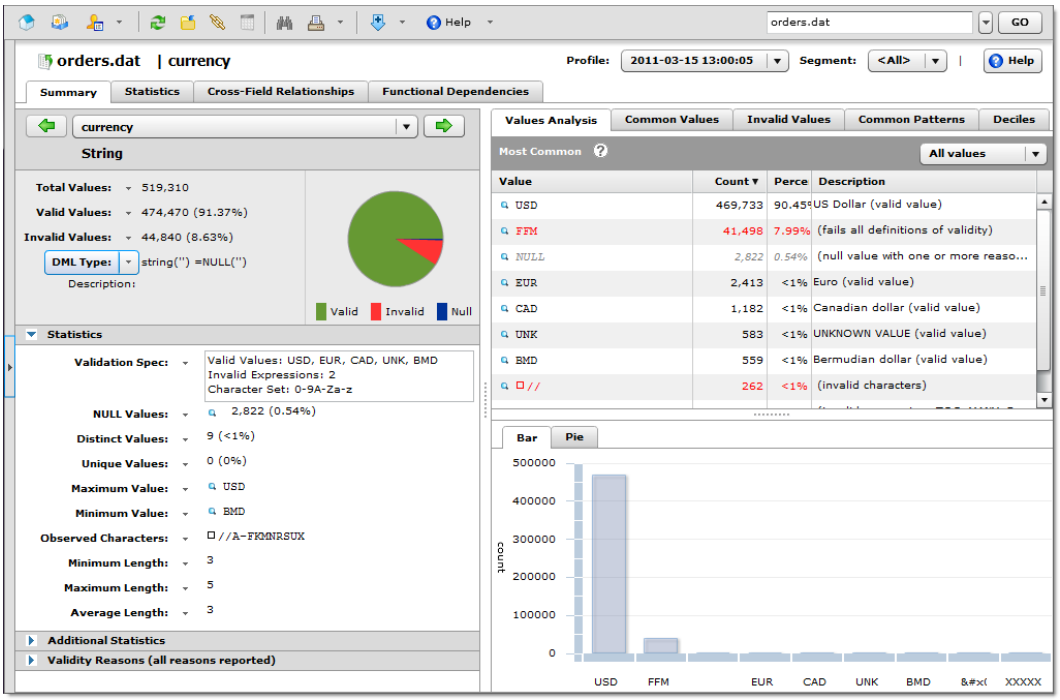
The Ab Initio Data Profiler results can also be used as part of a DQ workflow. As with all other DQ measurements, these results are stored in the EME and can be viewed through the EME web portal.

Many organizations consider data profiling to be an activity reserved for data discovery at the beginning of a project. But periodic automated data profiling can add significant value to a complete data quality program. While data quality metrics can capture the overall health and characteristics of the data, data profiler statistics allow for drilling down into a more detailed analysis of the contents of various datasets.

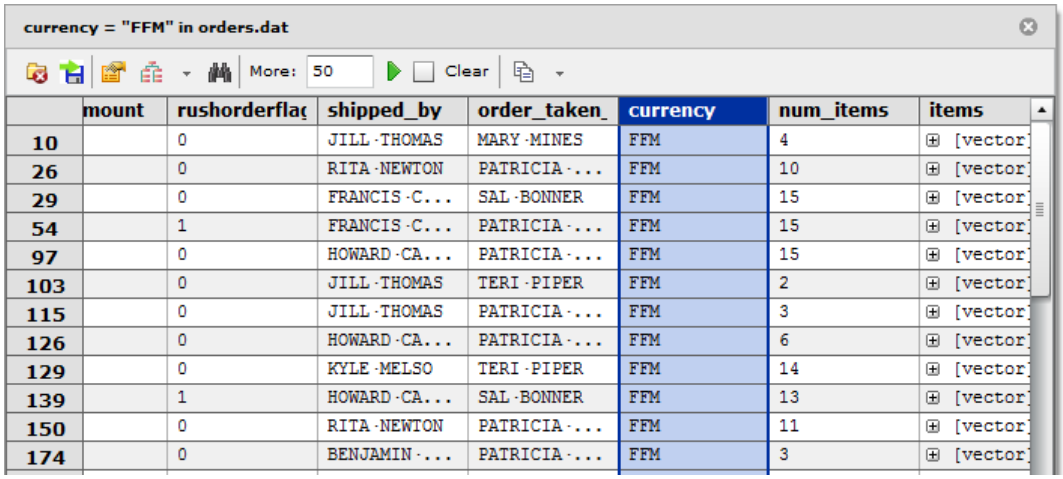
Below is a screen shot of the top-level report of a Data Profiler run on a particular dataset. Diversity (distinct values), validity, and completeness are just some of the information discovered by the Data Profiler. This information can be used to select which fields require further inspection.



Below is a screen shot of a particular field chosen by the user for additional analysis.



From this screen, it is possible to drill down to a display of the actual records that contain specific values in the selected field.



	mount	rushorderflag	shipped_by	order_taken_	currency	num_items	items
10		0	JILL·THOMAS	MARY·MINES	FFM	4	[vector]
26		0	RITA·NEWTON	PATRICIA·...	FFM	10	[vector]
29		0	FRANCIS·C...	SAL·BONNER	FFM	15	[vector]
54		1	FRANCIS·C...	PATRICIA·...	FFM	15	[vector]
97		0	HOWARD·CA...	PATRICIA·...	FFM	15	[vector]
103		0	JILL·THOMAS	TERI·PIPER	FFM	2	[vector]
115		0	JILL·THOMAS	PATRICIA·...	FFM	3	[vector]
126		0	HOWARD·CA...	PATRICIA·...	FFM	6	[vector]
129		0	KYLE·MELSO	TERI·PIPER	FFM	14	[vector]
139		1	HOWARD·CA...	SAL·BONNER	FFM	13	[vector]
150		0	RITA·NEWTON	PATRICIA·...	FFM	11	[vector]
174		0	BENJAMIN·...	PATRICIA·...	FFM	3	[vector]

CONCLUSION

While data quality is a problem that every company faces, there is no single approach to detecting, reporting, and studying data quality problems that fits every organization's need.

Ab Initio's end-to-end data quality design patterns can be used with little or no customization. For users with specific data quality needs, such as additional types of detection, reporting, or issue management, Ab Initio provides a general-purpose, flexible approach based on powerful pre-existing building blocks.

Ab Initio's approach to data quality is based on the Co>Operating System. The Co>Operating System provides a high-performance, multi-platform computing environment that performs data quality detection, amelioration, data profiling, and statistics aggregation for any type of data. The Co>Operating System provides unlimited scalability, and so can perform all of these tasks on very large data volumes.

Ab Initio's Business Rules Environment allows validation rules to be developed and tested by analysts and/or subject matter experts using an easy-to-use graphical interface. The result is significantly improved productivity and agility around creating and maintaining data quality rules.

And, Ab Initio's Enterprise Meta>Environment provides an unprecedented level of integration of data quality statistics with other metadata, including data lineage, data dictionaries, domain code sets, operational statistics, data stewardship, and other technical, operational, and business metadata.

The unmatched combination of these capabilities within a single integrated technology puts Ab Initio's data quality capabilities in a class of its own.

CONDUCT>IT: Managing a Complex Environment

As the number and complexity of IT applications within a business grow, the importance of operational management grows as well. And the expectations of the business for timely and reliable results grow even more. But, as any operations manager will tell you, getting those timely and reliable results is easier said than done when there are thousands and thousands of moving parts across multiple interdependent applications that may span several servers and geographic locations.

To be successful, the operational team needs to:

- Understand, articulate, and enforce all the key dependencies within an application and across applications. For example, the team needs to be able to say that B can only run when A has completed, and that C should run if A fails.
- Manage all the actions that can trigger a part of the process. A trigger could be a specific time on a particular day, the arrival of one or more files, the availability of one or more resources — or, perhaps, a combination of all of these.
- Proactively monitor all the dependencies so that alerts can be automatically raised and sent to the appropriate people. Alerts should be triggered if a dependency or event has not been satisfied within stated times, allowing business service level agreements (SLAs) to be tracked and reported against.
- Monitor the low-level processing characteristics of key parts of an application, such as the number of records being rejected, the latency of messages being processed, or the amount of CPU time being consumed by the processing logic. Again, alerts should be raised when thresholds are exceeded.
- Develop and test the end-to-end operational process in a dedicated test environment before formally promoting the process to the production environment.
- Record and analyze detailed operational statistics, such as actual start/end times for each part of an application over time, so that processing trends can be identified to support capacity-planning activities.

AB INITIO'S CONDUCT>IT PROVIDES ALL THESE CAPABILITIES, AND MORE

Conduct>It is a process automation facility that provides the monitoring and execution environment for deploying complex applications in complex environments. It facilitates the definition of arbitrary, hierarchical job steps for large, multistage applications, as well as the dependencies, sequencing, and scheduling of these job steps. These applications can be composed of Ab Initio graphs and job definitions, as well as custom executables and third-party products, all of which are managed by Conduct>It.

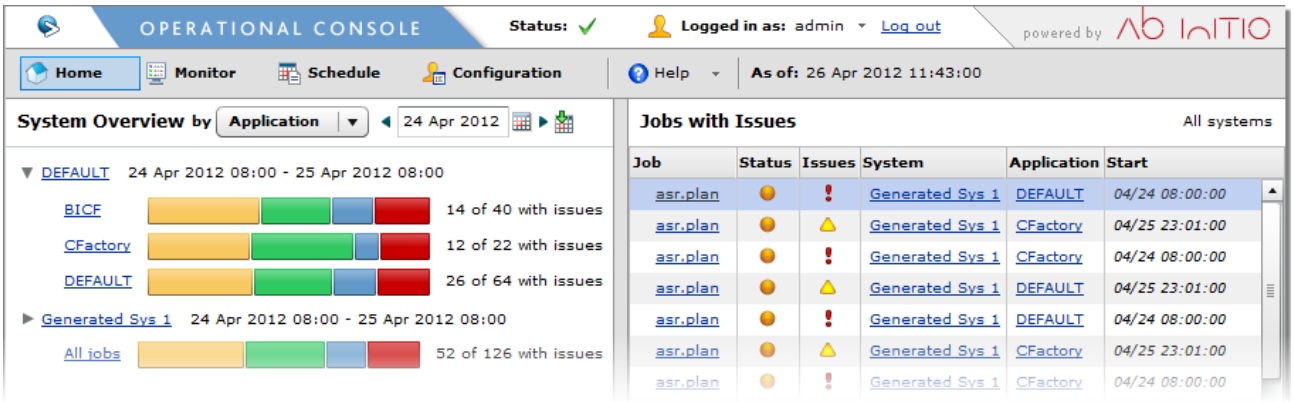
Conduct>It has two main elements. The first is a process automation server, called the Operational Console, that provides monitoring and job control in complex processing environments. Second, where sophisticated process management logic is required, Conduct>It offers the ability to graphically develop and execute complex control flow logic.

Let's look at the Operational Console first.

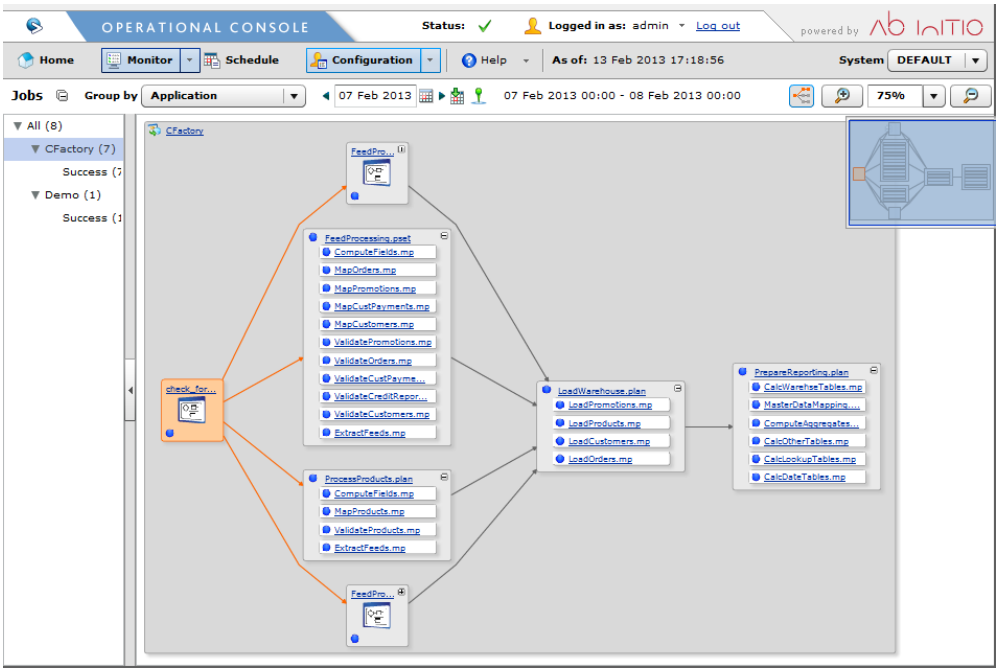
OPERATIONAL CONSOLE

The Operational Console provides a wide range of capabilities that are essential to daily operations. These include job scheduling, monitoring, alerting — and performing job-level actions such as starting, stopping, and rerunning jobs. Across all applications, the Operational Console collects, integrates, and manages the associated operational metadata, helping the operational team and business analysts in planning and maintaining efficient operations.

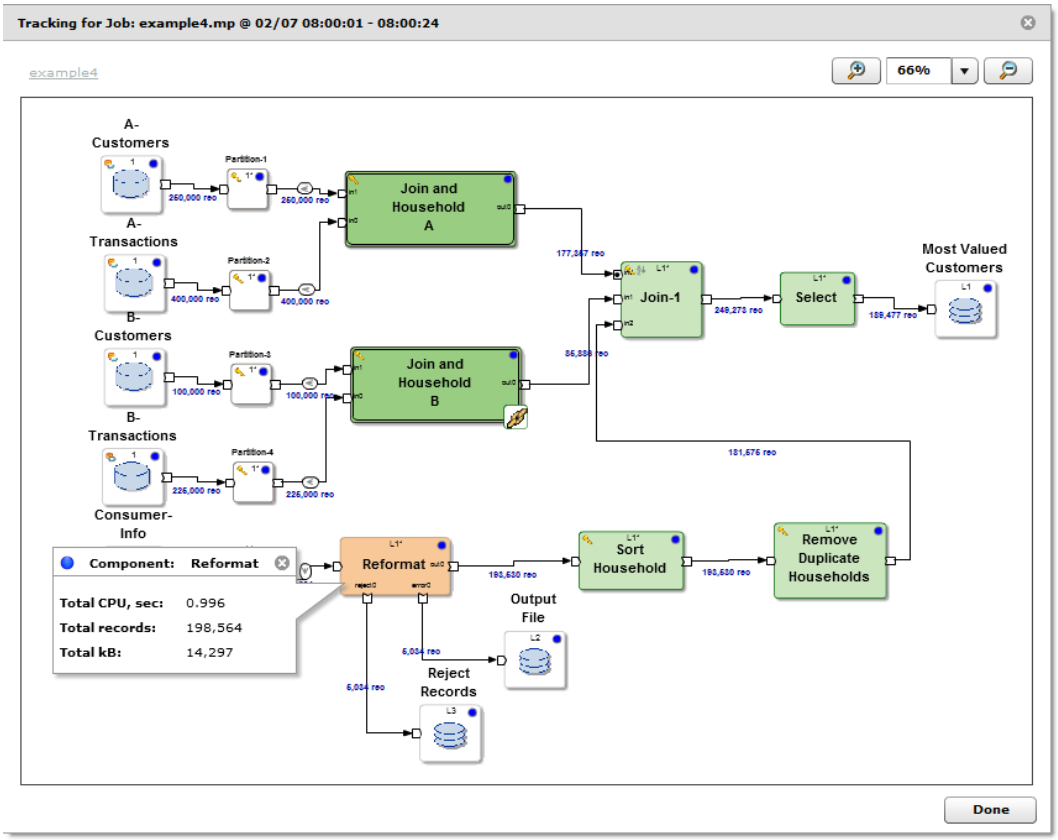
It all starts with the Home page on the Operational Console's browser-based interface, as shown below. The Home page provides a summary of all of today's jobs by application, system, or host server, showing whether they are running (green), completed (blue), failed (red), or waiting (yellow). It also lists all the issues or warnings that have been raised and are still outstanding.



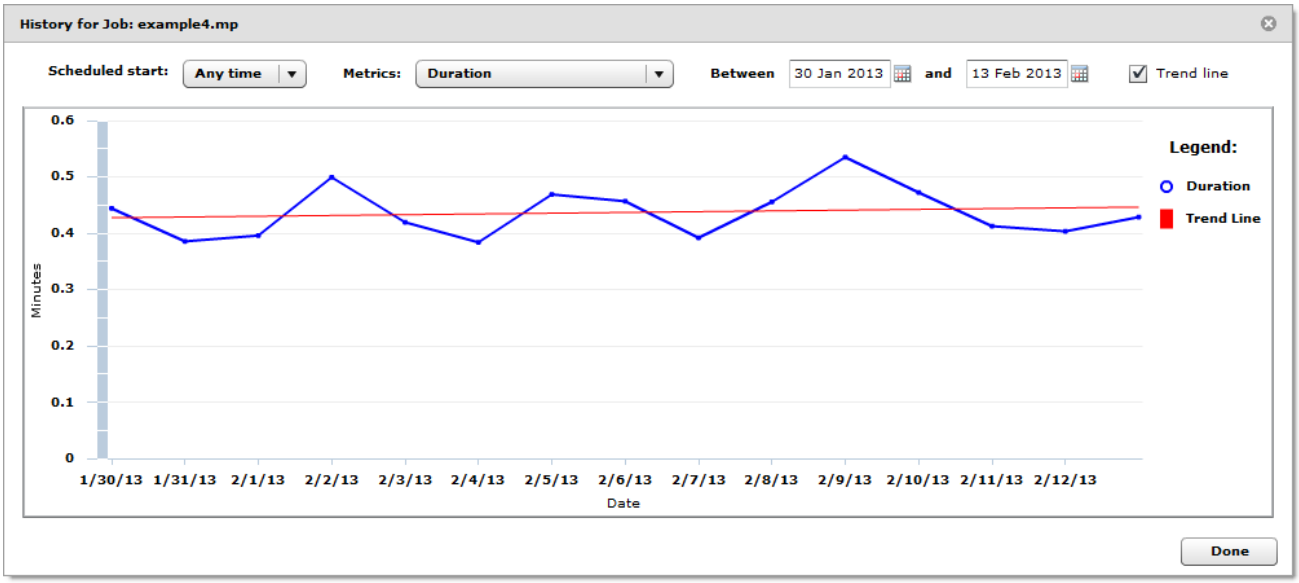
From the Home page you may drill down to see different types of information for any job in the environment — the reason for the failure, what a job is waiting on, when a job completed, or when it is expected to complete, etc. For example, you may wish to see all the jobs related to a specific application to understand how it is progressing:



This monitoring screen shot shows the dependencies between the different tasks within the selected application and the progress of each task. At any stage you can drill down further to see the tracking details of one of the tasks:



As shown above, low-level tracking information on CPU seconds consumed, as well as record and data volumes processed, is available for every component within a specific run of an Ab Initio job. Alternatively, you may wish to understand how one of your jobs has performed over time, together with a trend line for planning purposes:

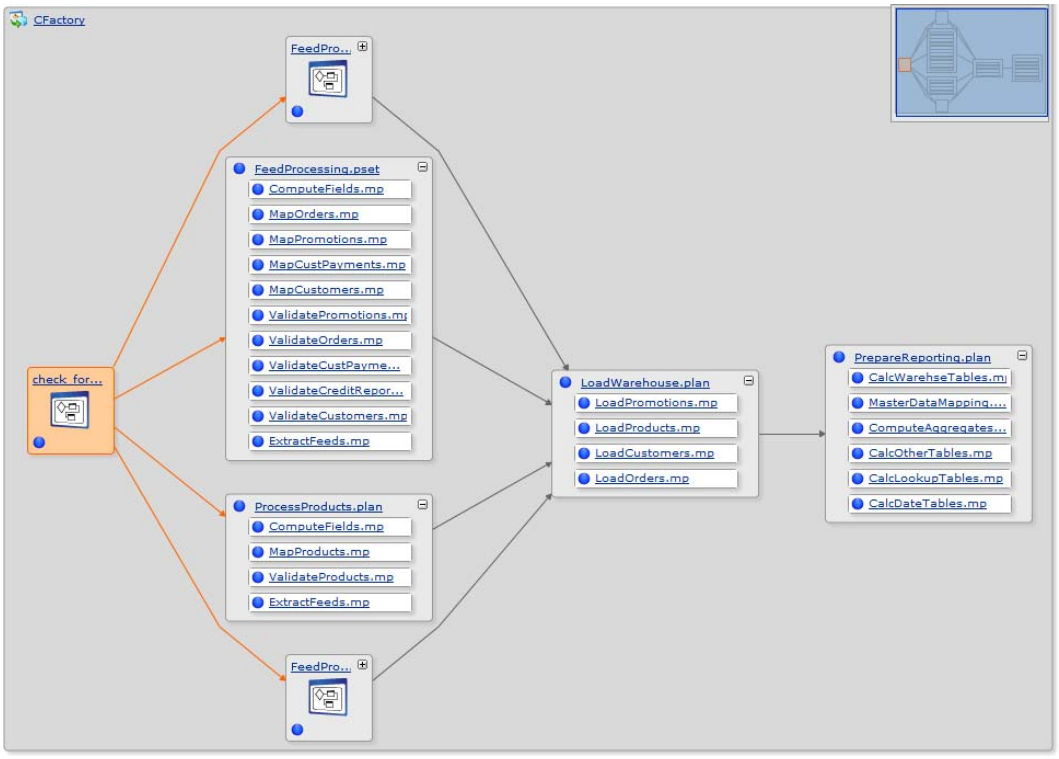


The Operational Console collects a wide range of statistics for each task, from the ability to meet the stated SLAs through to the amount of user and system CPU time being consumed.

However, it doesn't stop there. Using the full power of the Ab Initio Data Manipulation Language (DML), the operations team can also define their own operational probes — called "custom metrics" — for alerting and tracking purposes. You can add probes without changing or disturbing an application, and they have no impact on the application's execution. These metrics can be computed using any combination of tracking information for any flow or component within a graph. Hence, it is easy to add a custom metric that reports and alerts on the number of records processed by a specific component, or the amount of CPU time it is consuming, or the latency of the real-time messages being processed.

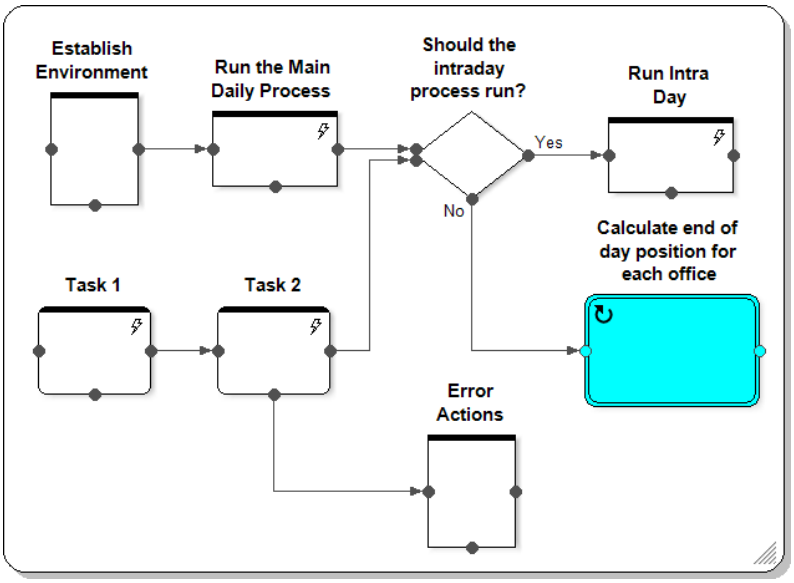
All of the Operational Console's monitoring capabilities are available for Ab Initio jobs, whether they have been initiated by the Operational Console or a by a third-party scheduler.

For customers who don't have access to a corporate scheduler, the Operational Console also provides full day/time-, event-, and file-based scheduling capabilities, allowing sophisticated applications to be fully scheduled without the need to write and maintain traditional scripts. The following screen shot shows the same application shown earlier, but with all the time- and event-based dependencies shown against the expanded tasks:



Because dependencies between tasks can become extremely complex in large applications, Conduct>It also provides a fully graphical environment to help developers define advanced job control flow.

A control flow is a way of expressing detailed logic about the sequence of execution. It uses a collection of connected tasks, called a plan, to describe what should be run — the connection between these tasks specifies an execution dependency (run this before that, for example):



The above plan shows that Task 2 can execute only after Task 1 has completed. It also shows that a condition is subsequently evaluated ("Should the intraday process run?"), and if found to be "No", then the end-of-day position is calculated for each office by iterating over the highlighted "subplan." A subplan, as the name suggests, is itself a collection of tasks and dependencies.

Custom tasks can also be triggered on the failure of other tasks — this is illustrated in the above plan by the "Error Actions" task, which is run if Task 2 fails for any reason. In a similar manner, each task can have "methods" associated with it that are executed when certain events occur, such as "At Start", "At Success", "At Shutdown", "At Failure", or "At Trigger", allowing reporting and logging capabilities to be easily added to the end-to-end process.

With plans, Conduct>It provides a development-time framework for deconstructing a complex application into manageable units of work that constitute a single, recoverable system. These plans are then available to be scheduled, monitored, and managed using the Operational Console. The result is a sophisticated end-to-end operational environment.

Get the Right Answer

Ab Initio software is transforming the way large institutions manage their data. Contact us to learn more.

CORPORATE HEADQUARTERS

AB INITIO

201 Spring Street

Lexington, MA USA 02421

+1 781-301-2000 (voice) +1 781-301-2001 (fax)

Ab Initio has a growing number of offices. Please check our web site at www.abinitio.com to find your regional office, or send inquiries to solutions@abinitio.com.