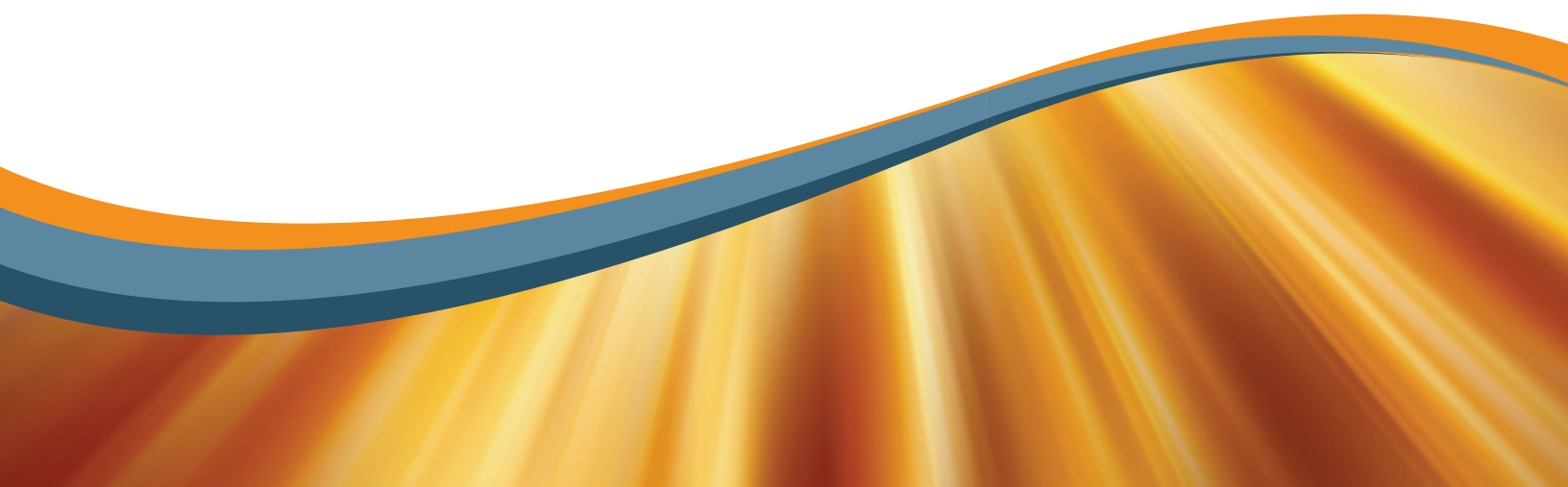




WHITEPAPER

# Implementing Metadata-Driven ETL Using Semantic Types

*January 2011 v1.0*



Semantic Types enable the rapid evolution of ETL applications to support nearly any format of physical data and thereby avoid the application maintenance nightmare of uncontrolled proliferation of procedural customizations.

## The Need for Semantic Types to Capture and Reuse Enterprise Data

As data integration requirements multiply, traditional extract, transform and load (ETL) infrastructure continues to fall farther behind in addressing diverse enterprise requirements. Many data integration software vendors have reacted to expanding ETL requirements by retrofitting their traditional ETL platforms with “bolt-on” and often poorly integrated products in an attempt to keep pace. Unfortunately, such reactive approaches have failed to address the essence of the problem—the *inability of their core product architectures to capture and reuse the true meaning of enterprise user data*. This whitepaper summarizes some of the most significant challenges that data integration projects confront today, illustrating how many of them stem from either:

- A failure to understand the scope of the data.
- The inability of traditional ETL products to capture that knowledge in a way that the scope can be spanned automatically.

It enumerates the ETL/data integration infrastructure requirements for an automated solution to integrate data between heterogeneous information systems in a way that preserves the semantics of that data.

This whitepaper also introduces a new technology developed by expressor software that provides a vital link needed to semantically unify multiple implementations of a common business data element. Semantic Types™ make it possible to map any number of physical representations of a data element across various ETL source and targets to a common semantic definition. By using declarative instead of procedural constructs, Semantic Types enable the rapid evolution of ETL applications to support nearly any format of physical data and thereby avoid the application maintenance nightmare of uncontrolled proliferation of procedural customizations.

## Simplifying Data Mappings with Semantic Types

The use of Semantic Types provides a powerful new approach to data mapping and transformation that greatly reduces your time-to-value. Instead of defining point-to-point, source-to-target data mappings and individual rules, you can define a Semantic Type representative of business data, create business rules you want to apply and then implement canonical mappings to map multiple sources and targets to that same Semantic Type.

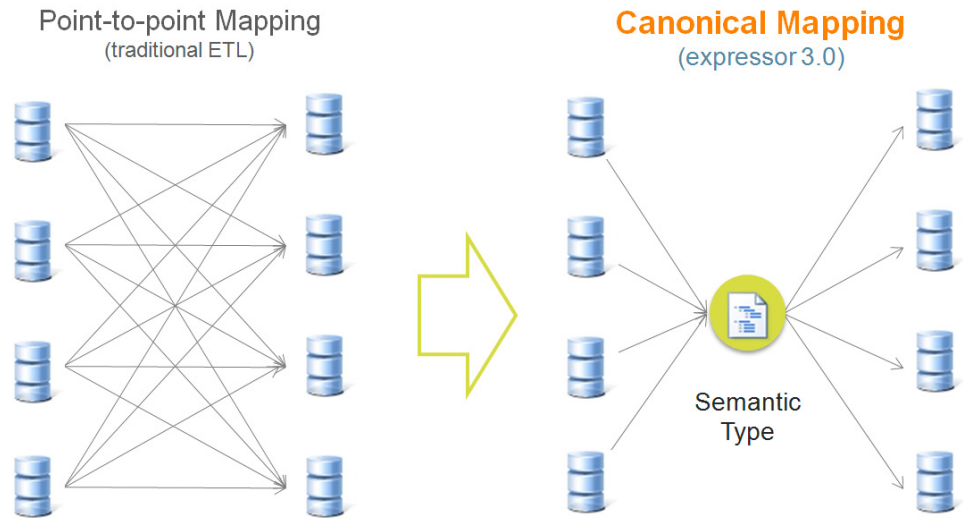


Figure 1: The expressor data integration platform avoids labor-intensive, point-to-point data mapping and allows you to leverage Semantic Types to simplify mappings across the enterprise.

Instead of defining point-to-point, source-to-target data mappings and individual rules, you can define a Semantic Type representative of business data, create business rules you want to apply and then implement canonical mappings to map multiple sources and targets to that same Semantic Type.

Since Semantic Types are captured as reusable artifacts, you can use them within your ETL data flow applications and build up semantic models of everything that is happening in your enterprise. With the expressor data integration platform, developers can simplify data mapping through unique name and data type rationalization, and can:

- Automate data type conversions and eliminate conversion errors.
- Build new Semantic Types from existing ones and reuse Semantic Types in existing and new applications.
- Create multiple, reusable business rules and apply them repeatedly.
- Rationalize source and target field names to improve user communication and data governance.
- Easily implement data quality rules and constraints.
- Check the validity of a data flow application before deploying.

## A Sample ETL Application

To illustrate some of the challenges in ETL projects, consider the data integration objectives of a hypothetical energy company—“Unerco”—that has decided to create a data warehouse for regular exploration cost assessments. To populate its data warehouse, its IT department had to develop a data integration

Since Semantic Types are captured as reusable artifacts, you can use them within your ETL data flow applications and build up semantic models of everything that is happening in your enterprise.

application that loads the data warehouse with fresh exploration expense data from hundreds of separate drilling sites across the globe on a nightly basis.

Unerco relies on subcontractors to perform the various drilling operations, and each subcontractor uploads its expense data via files. While subcontractors are not required to format their data, they are required to adhere to the following rules:

- They can submit at most one expense file per contract per day.
- Each expense file contains itemized expense records for exactly one contract.
- The expense files must be in the form of delimited text files with separate fields for the amount of the expense, the type of expense (material, labor, capital equipment), the date on which the expense was charged and the bore depth range at which the expense applies.
- They are required to include a unique contract identifier as part of the name of each expense file.
- They must use consistent data file formats for the expense files submitted for any single contract.

Figure 2 depicts the schema for the staging database that Unerco’s IT department plans to use for staging nightly loads into the data warehouse. The **Contract** and **Contractor** tables contain reference data that rarely changes; however, the **Expense** table is refreshed daily with the daily expense data from the subcontractors.

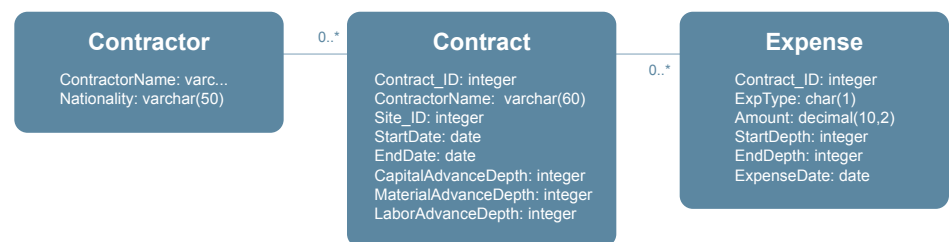


Figure 2: RDBMS schema for the exploration expense staging database

The ETL team is chartered with developing a data flow application that can load the **Expense** table with the data from the subcontractors. The team’s major pain point is the requirement that the ETL application supports each specific contract’s data format coupled with the fact that each contract’s data format uses a different combination of field layout order, field and record delimiters,

Deeply nested code complicates the process of identifying (and reporting on) the impact of proposed changes because data lineage must be tracked through that procedural (e.g. if/then/else) logic.

date masks, units of length (e.g. *meters* versus *feet*) and currency (*Euros* versus *US Dollars*).

In a nutshell, standardizing the inbound data is the ETL team's biggest challenge.

The application the team develops to solve this problem is shown in **Figure 3**. Note that the 2nd step in the flow, labeled **Standardize Units**, is used to programmatically standardize the units and measures provided in the subcontractors' data files. Also note that the **Check Bore Depth** step programmatically checks to ensure that each expense item is for a valid bore depth range. Ultimately, data records that are invalid are separated from valid data.

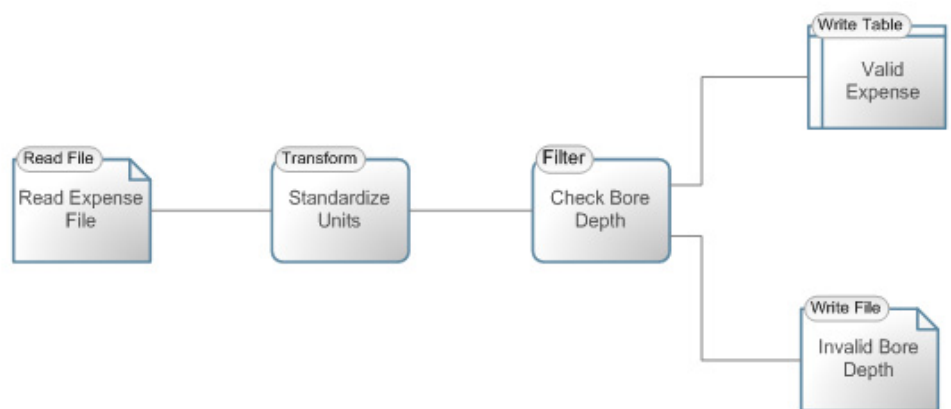


Figure 3: Initial ETL data flow for expense item loads

## Requirements for Semantic Automation in ETL

Although there is a wide range of possible solutions to this challenge, all traditional data integration approaches tend to suffer from one or more of the following costly problems:

- The data standardization rules are not declarative but instead buried in complex code blocks, effectively placing business people at least one step away from specifying actionable business rules in terms that make sense to them.
- Deeply nested code complicates the process of identifying (and reporting on) the impact of proposed changes because data lineage must be tracked through that procedural (e.g. if/then/else) logic.

The fundamental problem with traditional ETL products is that they force developers into procedural solutions for accommodating the wide variation in source and target data formats.

- The data standardization rules are expressed in terms of physical (“technical”) field names and not in terms of context-independent concepts to which business domain experts can easily relate.
- Because business rules are expressed in terms of physical field names, many conventional ETL products require data transformation functions to be created for each new source data format (i.e. procedural logic proliferation).
- Standardizing the source data to fit the target system’s context requires the creation of either code procedures for each source or one very complex generalized procedure that would typically perform poorly and further alienate the business domain experts due to its excessive complexity.
- When change requests arrive, the maintenance challenge associated with implementing the change can be significant.

## A New Approach for Addressing the Fundamental Requirements

The fundamental problem with traditional ETL products is that they force developers into procedural solutions for accommodating the wide variation in source and target data formats. Because they lack the features to express the semantics of the data, these products cannot possibly provide an automated means to retain those semantics across unavoidable, context-to-context transformations. This is a deficiency rooted at the very architectural core of traditional data integration software products and, for that reason, cannot be corrected without significant product re-architecture.

As the leader in semantic data integration, expressor software tackles the complexity and cost of enterprise IT projects with a metadata-driven solution that delivers breakthrough development productivity and data processing performance at a significant price/performance advantage. expressor provides both free and affordable enterprise-class data integration software, and has developed an innovative, patent-pending ETL solution that overcomes the limitations of legacy architectures and is comprised of:

**expressor Studio** provides game-changing ease-of-use with a drag-and-drop and wizard-driven interface that enables developers to easily connect to standard data sources and targets, map data to common business names and types, and design and run complex data flow applications in minutes.

...the foundational requirement of a semantic ETL product: the capability to model business concepts independent of any specific environments' physical representations of those concepts

**expressor Repository** is an enterprise-class semantic metadata repository that collects, stores and manages project management information, reusable data descriptions, application file versioning and performance metrics. It enables collaborative team development and centrally maintains the details of a data integration application, including user roles and assignments.

**expressor Data Processing Engine** is a high-performance parallel data processing system that runs a deployed data integration application. It enables breakthrough scalability by supporting batch and low-latency data processing. With expressor 3.0, the expressor Data Processing Engine now provides high-performance connectivity for all major RDBMSs and data warehouse appliances as well as generic ODBC support.

The remainder of this whitepaper defines the requirements for a semantically aware ETL product and illustrates how its features are used to assemble applications that are powerful, intrinsically capable of handling data variety and that avoid the classic pitfalls inherent in solutions built using traditional ETL products.

The requirements defined in the following sections describe capabilities that are in varying stages of development by expressor software; some are existing capabilities and available in expressor 3.0 and others are in the development stages and targeted for delivery during 2011. For more detailed information on product capabilities visit our website, and for more information on market requirements—including more granular information on the example highlighted in this whitepaper, contact us.

## Requirement #1: Build a Semantic Type Model

Implementers of IT systems are often forced to make assumptions about the meaning of the data they store and communicate. For example, in the Unerco case, some of the subcontractors decided that depth values should be reported in *meters* while others assumed they should be reported in *feet*.

In order for the information stored in many heterogeneous systems to be interchangeable, there must be a mechanism for describing the meaning of that data so those assumptions can be properly accounted for in data integration processing. This is the foundational requirement of a semantic ETL product: *the capability to model business concepts independent of any specific environments' physical representations of those concepts*.

Semantic Types are highly reusable design artifacts that enable expressing business transformation rules in terms that are independent of the multiple, heterogeneous external type models that define ETL data sources and targets.

To illustrate, let's take the concept of distance. To account for the fact that some contractors provide depth values in feet while others provide them in meters, the semantic model must somehow capture these different representations and relate them to the same semantic concept of distance. expressor Studio will fulfill this primary requirement by enabling users to model business concepts using Semantic Types.

Semantic Types are highly reusable design artifacts that enable expressing business transformation rules in terms that are independent of the multiple, heterogeneous external type models that define ETL data sources and targets. That independence translates into two significant benefits for the ETL development lifecycle:

1. Greater agility in application evolution over the time.
2. Full traceability of data lineage from targets back to their original data sources with tight correlation to the semantic artifacts that govern data interpretation, quality and transformations.

To illustrate, **Figure 4** depicts a set of related Semantic Types that model the concept of contractor expenses in the Unerco example.

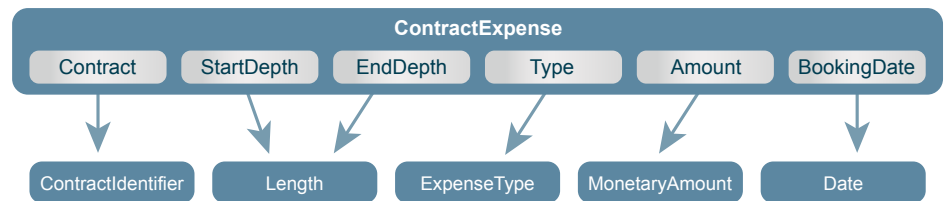


Figure 4: A library of expressor Semantic Types such as ContractExpense and ContractIdentifier created after importing RDBMS metadata for the target Expense table

With the expressor data integration platform, users create libraries of reusable Semantic Types in expressor Studio.

## Requirement #2: Create a Semantic Model Repository

The semantic ETL product must provide a repository for users to easily store, retrieve and report on the artifacts of the semantic model. With the expressor data integration platform, users create libraries of reusable Semantic Types in expressor Studio. These libraries can also be stored in a central repository (expressor Repository) so that they can be shared among multiple ETL projects. By organizing semantic models in searchable repositories, users can easily produce reports of their applications' artifacts to meet data governance and other application administration needs.

Types make it possible for users to define the structure of their data in terms that are important for business processes.

The facility for creating and maintaining Semantic Types and their representations provides its greatest utility when combined with the capability to bind semantic artifacts to actual data from source and target data environments.

One of the corollary requirements of storing semantic models is the need to bootstrap semantic models by leveraging existing metadata sources that lie beyond the boundaries of the semantic ETL product itself (e.g. RDBMS catalogs, XML models, etc.). expressor not only fulfills this requirement but also works in the reverse direction to generate external metadata structures using the Semantic Types contained in its repository, further accelerating the ETL development process.

### Requirement #3: Define an Integration Model

Semantic Types make it possible for users to define the structure of their data in terms that are important for business processes. To be useful for ETL applications, the semantic ETL product must provide a means to map Semantic Types to physical data sources and targets so that the data can be treated as occurrences of semantic concepts. To address this requirement, expressor Studio provides an integration model to align physical data structures with their appropriate Semantic Types.

The central role of the integration model is to define the different ways in which occurrences of Semantic Types can be physically represented and, if there is more than one representation, how to convert between them with no (or acceptably minimal) loss of semantic meaning. Semantic Types that are not composite – meaning those that do not contain any attributes – must be related to at least one representation type in the integration model in order to be associated with source or target data.

In expressor Studio, such Semantic Types are referred to as atomic Semantic Types, since their values are not compositions of other semantic values but instead atomic values conforming to a representation type. Each representation type concretely defines the domain of values that can be used to represent valid occurrences of their corresponding semantic atomic type.

### Requirement #4: Develop and Maintain a Schema Mapping Model

The facility for creating and maintaining Semantic Types and their representations provides its greatest utility when combined with the capability to bind semantic artifacts to actual data from source and target data environments. The binding process involves creating a set of mappings that bind artifacts in the semantic model to the structural definitions of records and fields to be read or

An attribute mapping is a reusable design artifact that associates a specific field in a schema with an attribute in a composite Semantic Type.

written by the ETL application. In expressor Studio, these structural definitions are referred to as schemas. ETL developers need the ability to easily create schemas of external source and target formats and map them to Semantic Types.

## Schemas

A semantic data integration product must support several different kinds of schemas. Table schemas describe the structure of database tables, XML schemas describe the structure of XML files (or streams), delimited schemas describe the structure of delimited text files, Copybook schemas describe the structure of COBOL working storage files, etc. Usually the most efficient way for users to create a schema is to import their structures from existing meta-data sources (e.g. an RDBMS catalog) but it should also be possible to create schemas manually using the ETL product's GUI. It is also very helpful to have a feature that can automatically generate particular kinds of schema from existing Semantic Types. For example, expressor Studio has a GUI wizard for generating a delimited schema from a composite semantic type.

## Attribute Mappings

An attribute mapping is a reusable design artifact that associates a specific field in a schema with an attribute in a composite Semantic Type. Depending on the combination of the primitive data type and the field type, certain properties may be required in the mapping to make the read and/or write operations possible (e.g. reading and writing datetime values expressed as strings in delimited text files requires a date mask such as *YYYY-MM-DD*). Attribute mappings also contain the knowledge of how to obtain the values for the meta-properties needed to fully characterize the interpretation of external data fields (e.g. whether a specific schema's expense field represents cost in *US Dollars* or *Euros*). It is common for such context values to be absent from the actual source data records themselves, but instead require some kind of derivation on the basis of knowledge of the context in which the data set was created (e.g. looking up the currency for a contract with the help of a database query predicated on the contract id).

## Deriving Schemas and Mappings from Available Metadata

Up to this point, our description of the semantic ETL infrastructure has proceeded in a top-down fashion, starting from the semantic model and proceeding down through the details of defining the integration and schema mapping models. While this approach was helpful for introducing the concepts of semantic ETL, it is not necessarily reflective of the preferred application development workflow in all cases. In fact, it is almost always the case that an ETL project has existing metadata available from which to derive artifacts in the semantic, integration and schema mapping models automatically from pre-existing

The richness of their meta-models gives semantic ETL products the advantage of supporting many more runtime scenarios with fewer design artifacts because the semantic knowledge contained in those artifacts imparts a higher degree of model reusability.

metadata. For this reason, semantic (even conventional) ETL products should include features that automate this derivation process.

There is a great deal of variation in the amount of detail provided from various kinds of metadata. For those sources that have considerable detail such as the metadata of database tables stored in RDBMS catalogs, expressor Studio makes it possible to derive the following semantic ETL model artifacts automatically:

- A composite Semantic Type that captures the concept conveyed by the table's relation.
- A semantic attribute for each of the columns of the relation.
- Either new or existing atomic Semantic Types for each of the attributes in the composite type.
- An initial set of representation artifacts (i.e. string, integer, decimal, etc. value data types and their constraints) in the integration model for each newly created atomic semantic type.
- An internalized model of the RDBMS table's structure expressed as a table schema (e.g. including the name of the table and the names of the columns needed by the ETL engine at runtime to formulate SQL statements and choose appropriate programmatic data type bindings).
- All of the attribute mappings for binding the columns of the table schema to the representations of the semantic attributes.

The main benefit of automatic derivation of model artifacts from existing metadata sources is to get an application running very quickly. Although several kinds of artifacts are created from a single import operation, the RDBMS table import wizard in expressor Studio automates their construction so that users can produce working flows efficiently yet still retain the option of refining the artifacts afterward (e.g. add additional attributes, change the default choice of representation, etc.).

## Requirement #5: Compile, Deploy, Execute

Any data integration product would be incomplete without the facilities for analyzing data flows, generating the necessary runtime packages, deploying those packages and efficiently executing those packages. The richness of their meta-models gives semantic ETL products the advantage of supporting many more runtime scenarios with fewer design artifacts because the semantic knowledge contained in those artifacts imparts a higher degree of model reusability.

expressor Studio's richer semantic ETL model makes application development extremely easy for less technical users since many of the problems that required manually-coded procedural logic using a traditional product are eliminated outright through the use of automatically generated logic.

This translates into fewer deployments, fewer development cycles and more meaningful feedback from product error and warning messages since more of the behavior of the product is configured declaratively and according to meta-data properties that have business significance (e.g. *Warning: Source data from field 'BeijingExpress\_CSV.Amount' has 'Currency' in 'Yuan' and target field 'StagingExpenses.Amount' requires 'Currency' in 'Baht' but there is no 'Currency' function capable of converting from 'Yuan' to 'Baht'*).

Overall, expressor Studio's richer semantic ETL model makes application development extremely easy for less technical users since many of the problems that required manually-coded procedural logic using a traditional product are eliminated outright through the use of automatically generated logic.

## Pulling It All Together

The preceding sections elaborated the requirements of a semantic ETL product by illustrating the kinds of artifacts that would be created in the Unerco ETL scenario. The following list summarizes the steps that the user would follow to build the artifacts needed for a semantic ETL application:

1. Derive the ContractorExpense Semantic Type, table schema and attribute mappings by importing the RDBMS metadata for the Expense table in the staging database.
2. Add new atomic Semantic Types for ScaleFactor, Currency and LengthUnits and define their representation constraints to the permissible set of scale factors, currencies and length units.
3. Augment the representation types that were generated for the Depth and Cost types so that they have LengthUnits, Currency and ScaleFactor meta-properties to parameterize the interpretation of their value parts. Write the necessary conversion functions to convert from all scale factors to scale factor 1, all expected source length units to *feet* and all expected currencies to *US Dollars*.
4. Edit the attributes mappings for the Expense table schema to set the meta-properties for length units, currency and scale factor to *feet*, *US Dollars* and *1*, respectively.
5. Generate delimited schemas for each of the different contractors by importing a sample expense file from each and bind their fields (or context variables) to either the base values or meta-property values of the representations to which they correspond.
6. Configure the input and output operators in the flow to properly initialize the source and target contexts.

In the semantic solution, there are no normalization functions to write or maintain since the semantic ETL analyzer/compiler can derive these rules dynamically when necessary.

Three of the steps above would have counterparts if the application were built using a traditional ETL product, namely steps 1, 5 and 6. Steps 2-4, which in most cases are optional, provide the additional semantic knowledge needed by the semantic ETL product to automatically support any combination of currency, length units or scale factor.

Of course, the most important difference between the conventional ETL solution and the semantic solution is that the semantic solution's flow does not require the **Standardize Units** transform operator shown in **Figure 3**.

This is because the semantic ETL product's dataflow analyzer and package compiler modules are capable of providing the read-file operator with the logic needed to dynamically compute the correct unit normalization logic according to the schema, mapping and contract for each contract expense file.

Although different delimited schemas will need to be generated (via import operations) for the different contractor reporting formats (and perhaps an occasional new representation type in the process), the procedural rules to map between the different units or format masks has been completely eliminated since all such variations are specified in the schema mappings declaratively. In the semantic solution, there are no normalization functions to write or maintain since the semantic ETL analyzer/compiler can derive these rules dynamically when necessary. **Figure 5** shows the flow used for the semantic ETL application.

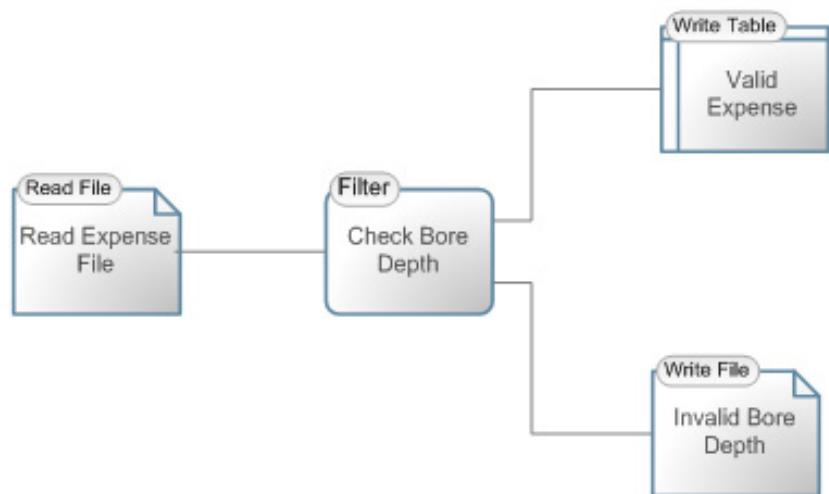


Figure 5: A simpler ETL flow made possible with solutions from expressor software

## Design and Run Complex Data Flow Applications in Minutes—for Free!

Building and maintaining ETL applications that are flexible enough to accommodate a wide variety of input and output data formats can be challenging, and ETL developers typically solve this problem programmatically by writing one-off procedures (i.e. imperative code) to convert each different data format into a standard or proscribed format. As the application evolves over time, the procedural logic for each variation needs to be explicitly refactored. These changes are difficult to manage, track and maintain and require a high-degree of programming expertise to implement.

The Semantic Types incorporated into expressor Studio enable a fresh approach to solving this problem. expressor Studio provides ETL developers with a data processing engine that automatically interprets data according to its business semantics. The business semantics of the data are stored as semantic model design artifacts in a shared repository and not in the form of complex procedural logic (i.e. code).

expressor Studio provides ETL developers with a data processing engine that automatically interprets data according to its business semantics.

Therefore, when changes are made to the physicality of the data flowing through an application, expressor Studio users are not forced into refactoring complex procedural code but instead need only define new representation types and mappings to accommodate the changes in the external schemas. These changes are fundamentally additive rather than destructive in nature and occur at a level above the data flow model and its embedded business rule logic, making the changes ubiquitous in the application space.

Not only is the model easily structured by business users, it also allows for the easy reporting of data mappings, business rules usage and data flow artifacts for the purposes of data governance and lineage reporting. In summary, the Semantic Types capability in expressor Studio allows users to easily create and maintain reusable and reportable data structures, business rules and ETL applications. Ultimately, expressor Studio's Semantic Types capability lowers the overall cost of implementing ETL applications by lowering application complexity.

For more detailed information on the requirements and on the specific example discussed in this whitepaper, contact us. You can also find out for yourself how easy it is for your company to benefit from semantic data integration. expressor Studio provides game-changing ease-of-use with a drag-and-drop and wizard driven interface that enables developers to easily connect to standard data sources and targets, map data to common business names and types, and design and run complex data flow applications in minutes.

expressor Studio is a high-reward, low-risk option that allows you to immediately get started and benefit from semantic data integration. With expressor Studio, expressor software delivers enterprise-class ETL onto the desktop and allows you to benefit from:

- Enterprise-class ETL on your desktop—for free!
- A familiar Microsoft Office-like look-and-feel.
- The ability to create reusable data mappings--point-to-point data mappings will be history.
- Fast data processing via an embedded engine.
- Access to a vibrant community, and online support from expressor software and your peers.

You can create reusable business rules, collaborate with team members and design, test and run integration applications right from within your Windows desktop. And most importantly, you can download expressor Studio right now for free—with no strings attached.



### About the Author

Bill Kehoe is a Founding Engineer at expressor and has been a key developer since the original 1.0 version of the product. He is now a Lead Architect providing technical leadership on all aspects of product engineering.

Previously, Bill was an Architect at Blue Agave Software where he was the Lead Developer for the data sub-system of a supply chain management product. Bill also held a Senior Architect role at Versata and was a Senior Developer and Program Manager at Sybase, where he architected and led the development for SQL Debug, a client server application for debugging Sybase Transact-SQL stored procedures. Bill graduated Magna Cum Laude from Tufts University in Civil Engineering and is a member of the Tau Beta Pi Engineering honor society. He has also done postgraduate work at Harvard University.

## About expressor software

expressor software knows data integration—and we know today's most capable data integration tools are too complex and too expensive. Our vision is to provide enterprise-class data integration software that is cost-effective, fast and easy to use.

expressor's game-changing usability enables you to use less technical, lower-cost development resources. Our unique, active metadata foundation simplifies data mapping and transformation to reduce your time-to-value. And our dramatically lower cost makes it easy to justify replacing your brittle and expensive hand-coded implementations or underperforming in-house ETL tools. expressor 3.0 is a comprehensive design, development and deployment platform available in three editions tailored to support the full range of data integration applications, from tactical data migrations to the largest enterprise data warehouses and strategic, predictive analytics.

To learn more about what makes expressor the leader in affordable, enterprise-class data integration, visit [www.expressor-software.com](http://www.expressor-software.com) or download the free expressor Studio at [www.expressorStudio.com](http://www.expressorStudio.com).

expressor software corporation  
1 New England Executive Park  
Burlington, MA 01803 USA  
[www.expressor-software.com](http://www.expressor-software.com)

©2011 expressor software corporation. expressor, Semantic Types and redefining data integration are trademarks of expressor software corporation. All other trademarks or trade names are properties of their respective owners. All rights reserved.