PhUSE 2009

DH03

ETL and the Clinical Programmer – They don't have to be Enemies!

Chris Decker, Life Sciences Director, d-Wise Technologies
Chris Olinger, President, d-Wise Technologies

ABSTRACT

ETL, better known as Extract/Transform/Load, usually conjures up images of graphical point and click interfaces, pre-defined rigid processes, and those little lines you draw between variables. These are all images that are very foreign to a long time clinical programmer and can immediately lead to resistance to try something new that doesn't let them "just write code". In general, an ETL process and associated tools use very rigorous processes and modularized code, both concepts that do not lend themselves to the flexibility needed within the clinical programming environment.

If implemented correctly, ETL can provides a framework and re-usable code base that means less time spent transforming data and more time spent focused on the 'real' science of analyzing data. It can also provide significant advantages when implementing and managing data standards such as CDISC. This paper will discuss the challenges of understanding and rolling out an ETL process and provide a case study of implementing an ETL solution.

INTRODUCTION

ETL, better known as Extract/Transform/Load, usually conjures up images of graphical point and click interfaces, pre-defined rigid processes, and those little lines you draw between variables. These are all images that are very foreign to a long time clinical programmer and sends them running for the hills and shouting at their ETL enemies "Just let me write code". In general, ETL products on the market use a very rigorous process that separates inputs from the transformation code and the target tables. In some industries this very rigid process works well because the source structure, and more importantly, the target data are very standard and robust. However, within clinical data, both the underlying medical science and the analytical science are always changing, so defining rigid targets and repeatable processes can be difficult. In addition, traditional generic ETL tools are built generically and don't usually understand the needs of clinical research. Therefore, it becomes an awkward process of fitting a round peg into a square hole.

Another challenge revolves around the individual programmer's need to get the work done. They just want to write code and this new ETL 'tool' only makes their work more tedious and slows down their production. What they don't realize is that by supporting a 'write some code' approach to transforming data, they can create a process that is fractured, unrepeatable, and does not support the management and reuse of metadata which is critical to developing and maintaining standards.

ETL is not necessarily a software product but a concept which separates the extraction of the data from external sources (E), transforms the data for operational needs (T), and loads the data into a final deliverable data set (L). If implemented correctly, ETL can provides a framework and re-usable code base that means less time spent transforming data and more time spent focused on the 'real' science of analyzing the data. It can also provide significant advantages when implementing and managing data standards such as CDISC.

This paper will discuss the following topics:

- Overview of ETL and the advantages that can be gained from using an ETL process
- Challenges of understanding and implementing a true ETL process in the clinical world
- A case study of implementing an ETL solution including the best practices and challenges
- Extending generic ETL solutions to support the clinical trial process

WHAT IS ETL?

AN OVERVIEW OF ETL

ETL is defined as the process of extracting (E) data from an external source, transforming (T) the data for operational needs, and loading (L) the data into a target or analysis table (Figure 1).

Figure 1

The first part of an ETL process involves extracting the data from source systems. In most industries these source systems might be very different in terms of the organization, underlying technology, and metadata. In the past the clinical programmer usually gets a dump of SAS® data sets and is on his merry way. However, over the past decade the breadth of source systems and data formats has expanded as data is extracted from different laboratories, EDC vendors, and CROs makes the access to data much more complex. In addition, the constant mergers between companies can also create the same challenges.

The transformation step of ETL is the stage where a series of rules or functions are applied to the extracted data to derive the data for creating an end target. Within clinical programming the ETL 'language' is predominantly SAS. Some data sources will require very little or even no manipulation of data while in other cases, a range of transformations types might be needed. Some of these transformations might be very complex and might not be supported by standard out of the box tools. Within clinical programming this is the meat of the work that is done. At the most fundamental level, clinical programmers write free text SAS code to create these transformations.

Finally, the last step in the ETL process is to load the data into some end target, usually a data warehouse in a traditional ETL process. Depending on the requirements of the organization, this process varies widely. In a traditional warehouse, as the load steps interacts with a database, constraints are defined in the database schema which supports validation of the data (e.g. uniqueness, formatting, mandatory fields). This contributes to the overall data quality performance of the ETL process. In clinical programming users sometimes build this ad-hoc checking during the process but in general there is not always a very rigorous, and more importantly, automated process for validating the target data.

METADATA

Metadata is data about data and is a concept this industry now understands much better than it did a decade ago. In the ETL world it is usually comprised of the information about the data – the column names, the types, the formats, the references to other columns for key relationships, and so on. Good ETL tools let you manage this metadata in a consistent fashion through a centralized metadata repository. Use of the repository allows for consistent sharing of metadata across all functions of the ETL tool. For instance, if a column is declared as a character column of length 32 then this information is used for table stub generation, handling extracts and transfers to the table, and for any transformation code that may be generated to move the column to another table.

The consistent use of metadata also allows for the auto-generation of code. In fact, the majority of ETL tools were created because of the vision that there should be better ways to build data processes than by writing error prone code. You should be able to visually represent a flow of a job by connecting various transforms in a series of steps. These transforms are then responsible for generating whatever underlying script/code is necessary to transform the data.

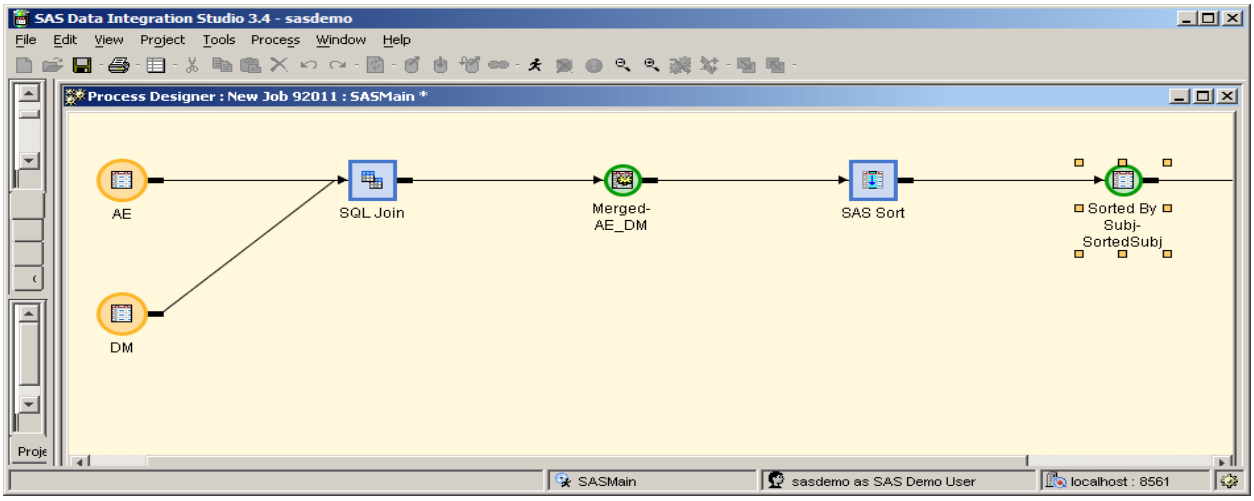
However, the auto-generation of code is the type of functionality that can make using an ETL tool so frustrating. If you are a programmer (especially a SAS programmer) then you are used to having complete control over your environment. In an ETL tool world, you are manipulating metadata and are not in complete control of the code that is being generated. This code is wholly managed by the tool itself and not the programmer. As well, access to data is very tightly controlled. In a centrally managed environment, data managers and other IT staff usually are the people that have the power to grant you access to data. For a SAS programmer, not being able to specify your own LIBNAMEs is not familiar.

The question you have to ask is if an ETL tool worth the added complexity and control? We would argue that the answer is yes – with some caveats. In the SAS world, the answer is not as cut and dry. In the general IT world of PL/SQL and Oracle then the answer is a resounding yes.

ADVANTAGES OF ETL

As we stated earlier, an ETL tool has some distinct advantages over code only solutions. For the clinical programming environment probably none are more important than documentation and change control. A typical transformation job in any of the various ETL tools involves a visual display representing the flow of the ETL job. This display is MUCH easier to fathom as compared to a set of code (especially SAS macros). For instance, the following screenshot (Figure 2) shows a job from SAS® Data Integration Studio (DI Studio). What is it doing? Just by looking at the job you can tell that it is merging AE and DM using a SQL join, and sorting the data on Subject ID. Again the user gets a decent idea of what it is doing just by following the flow.

Figure 2



As well as documentation, change control plays a large part in the clinical development process due to rigorous regulatory requirements. Jobs that are based on metadata can use the standard metadata repository controls for allowing and denying access to metadata elements. What this means is that access controls can be applied across all jobs in the metadata server. This may actually be a bad thing if you desire more control.

Other areas where ETL tools provide advantages include the following:

- **Centrally managed table metadata** – That is, one version of the truth. Typically, ETL tools require the pre-loading of tabular metadata from the various source systems. This metadata is then used by all clients writing ETL jobs.

- **Centrally managed data access** – whether through SAS (LIBNAMES) or databases (Oracle, SQL), all linkages to data are managed in a central location and can be globally maintained across the tool set.

- **Impact analysis** – How do I know I am not going to break anything if I change the format or structure of an entity? Impact analysis gives us the ability to see who is referencing a given set of tables or columns. In the code only world the only way to do this is to search the code base for references to the tables. Centralized metadata enables the easy searching of references and usage.

- **Building complex jobs** – ETL provides the ability to build complex jobs with little experience (which some might see as a detriment). It is entirely possible for people not familiar with the underlying code base to build an ETL process because they understand the data and the relationships contained therein.

- **Monitoring** – Good ETL tools provide a robust platform for monitoring jobs and for error detection and recovery.

- **Multi-threaded run ability** – There are often times where segments of a job should be able to be run in parallel. A good ETL tool will provide this ability out of the box.

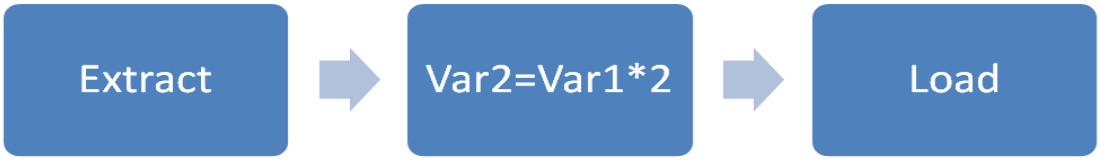
- **Robust set of pre-configured and customizable transformations** – Most tools provide standard controls for a suite of ETL oriented tasks (good ones provide for a lot of non-standard items as well). For instance, the ability to add messages to a system bus, HTTP web service access, and regular expression parsing of data are all aspects that should be provided in a robust and flexible ETL tool.

- **Controls for the life-cycle of data** – Includes controls for data extraction, cleansing, transformation, metadata management, and master data management. That is, ETL is only one aspect of the entire lifecycle of the data and ETL tools should also allow you to take advantage of these other areas.

CLINICAL PROGRAMMING AND ETL: THE DISCONNECT

In general, ETL products use a rigorous process that separates inputs from the transformation code and the target tables. In some industries this very rigid process works well because the source structure, and more importantly, the target data are very standard and robust. For example, within the financial industry a dollar is a dollar and therefore applying repeatable processes to this industry makes sense and is fairly straightforward. When ETL vendors demonstrate their ETL technology they always show you the following transformation mapping.

Figure 3



However, within clinical data, both the underlying medical science and the analytical science are always changing, so defining rigid targets and repeatable processes can be challenging. The famous saying for a clinical programmer is ‘This study is unique’. In addition, the derivations that sometimes occur in the transformation process can be quite complex so in reality the picture looks more like this and the argument is always that you can’t automate the big black box.

Figure 4

CURRENT PROCESS

Over the last three decades, the process for building clinical data sets has to been to write and run SAS code. SAS has been ingrained in the clinical programming environment for a very long time. During that time people have built entire systems and frameworks around SAS and in some cases those systems have been quite robust and efficient. However, relying on a single technology or programming language to build a robust framework is usually not a good idea. In addition, using a very high level language like SAS to build a 'system' can be fraught with issues. You sometimes can stretch the capabilities of a high level language like SAS by asking it to do too much.

An ETL process as defined above has been used many times by scores of clinical programmers – even if they do not want to put a formal name to it. For instance, data is extracted from Oracle Clinical into SAS data sets. Programmers then write SAS code to clean, analyze, and then finally translate source data into alternative forms (e.g. SDTM domains). In fact, a normal SAS programmer will say that they have been practicing ETL for many years! Technically, the SAS program below could be called an ETL process.

```
/** Extract **/
Data VS;
set in.VS;
run;

/** Transform **/
Proc transpose data=VS out=vitals;
  By USUBJID VISIT;
    ID VSTESTCD:
    VAR VSSTRESN;
 Run;

/** Load **/
Data out.VITALS;
  Set vitals;
Run;
```

However, a true ETL process will bring more to the table than just the code that is used to move data from point A to point B. The majority of ETL tools available today allow you to use metadata to do things from document standard processes to allow for impact analysis on tables and other artifacts. These functions bring value added to the table for any clinical programmer.

DESIGNING THE CODE

In general, clinical programmers will try to write code from the top down. This contradicts the traditional methodology of ETL and robust programming in which you design and modularize your code as much as possible. In building large scale systems architects usually spend a majority of their time designing the components before they ever write a line in a code and the same methodology should be used within clinical programming.

In addition, programmers will sometimes try to write as much code within a single task as possible. Breaking down the tasks within your program into smaller encapsulated pieces makes the code more reusable and easier to review. While writing smaller more straightforward pieces might make the program lengthier and seem very tedious it provides a more scalable solution and hopefully provides a significant amount of reusable code.

EFFICIENCY

Another challenge revolves around the individual programmer's need to get the work done. They just want to write SAS code and this new 'tool' only makes their work more tedious and slows down their production. What they don't realize is that by supporting a 'write some SAS code' approach to transforming data, they create a process that is fractured, not repeatable, and does not support the management and reuse of metadata which is critical to developing and maintaining standards.

It's time to let the secret out. Implementing and using an ETL process, especially within ETL technology, might not make the individual programmer more efficient. Making a single programmer go through a number of extra steps to deliver the final product is cumbersome and frustrating to that individual and can create challenges during the adoption process. It is essential that you help the individual user understand the value of an ETL process and/or technology and ease their pain as much as possible.

The overarching question is whether existing ETL tools provide the flexibility needed within clinical programming while maintaining the structure and rigors of an ETL process.

CASE STUDY: IMPLEMENTING AN ETL SOLUTION

OVERVIEW

This section will describe a case study of implementing SAS Data Integration Studio (DI Studio), a generic ETL solution, within a clinical programming environment. While this is specific to a SAS technology the best practices and challenges can be applied across both within an ETL process and the associated technologies.

This customer had multiple sites they had acquired over a period of a few years. All sites performed their clinical programming and data transformations using different processes and tools. The customer made the decision to implement SAS Data Integration Studio including the clinical plug-ins.

BEST PRACTICES

As described earlier, since the out of the box recommendations for using an ETL solution don't necessarily map very well to a clinical programming process we had to define best practices for learning and using the solution.

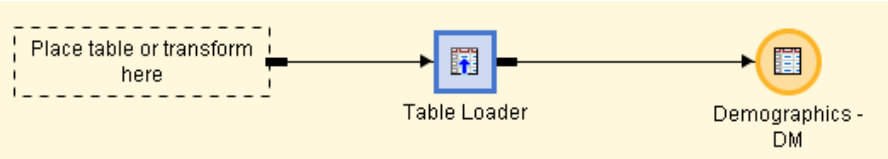
Gap Analysis

The first step was to define what we called a gap analysis. This was sort of a translation book between the terminology used in the ETL solution and the customer's process. For example, in clinical programming macros are used for standard code that might be reused. Within DI Studio these are called transforms. Within DI Studio all joins of data sets use PROC SQL whereas clinical programmers mostly use data step and merge functionality. Defining this gap analysis and providing a 'translation book' helped facilitate the learning curve.

Ignore Training

This might seem counterintuitive but in most cases when you attend a training involving an ETL solution the vendor provides your company the out of the box training which isn't remotely like your own process. A great example of this within the ETL world is that most solutions tell you to start with your target and build backwards. For example, the second step within the creation of a DI Studio job is to define the final table. You then end up with a process that looks like the following:

Figure 5



As we all know, within clinical programming we might have a good idea of our final table but the process to get there doesn't start at the end and it's not how code is really developed. Within DI Studio and other ETL solutions, ignore what they tell you in the training and think about how your business process can be applied to the technology. In this example, it is just as easy to start at the front and not include the final table until you are sure it's correct. Add a step, test your code, add another step, test your code, and so on. While the training and documentation won't describe this process to you, it's possible and maps much easier to the iterative development environment of clinical programming.

In the figure below, the first step is to extract the data (Figure 6). Add this section, test the code, and make sure it works. Then add the second node (Figure 7) in which you do a sort. After each step you can test your code to ensure you are getting the correct results.

Figure 6

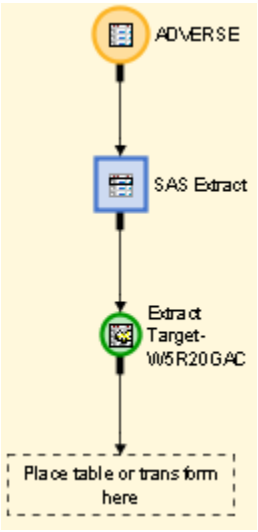
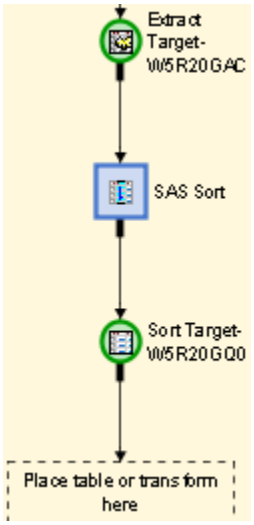


Figure 7



Plan! Plan! Plan!

The next two best practices are common sense and good idea in any general coding environment. The first is to design your code up front. Most of the time programmers will just jump in and start coding and not take the time to design the process. While it's possible, in general, ETL solutions don't work well with this mindset. Spend time breaking your process down into a set of encapsulated code bits. With this customer we developed a code design document which allowed them to think about the code prior to building it which included a section for each part of the process.

Don't Overload

As mentioned earlier in this paper, designing the code is critical to ensure robust and reusable code. In Figures 6 and 7 above you could do an extract and sort in the same step as well as a dozen more functions (e.g. rename, keep, converting character to numeric) but overloading a process creates a single piece of code which can't be used anywhere else. It might seem tedious and even 'simple' for veteran programmers to develop code this way but the advantages of easy to read and reusable code greatly outweighs the speed of getting the code done.

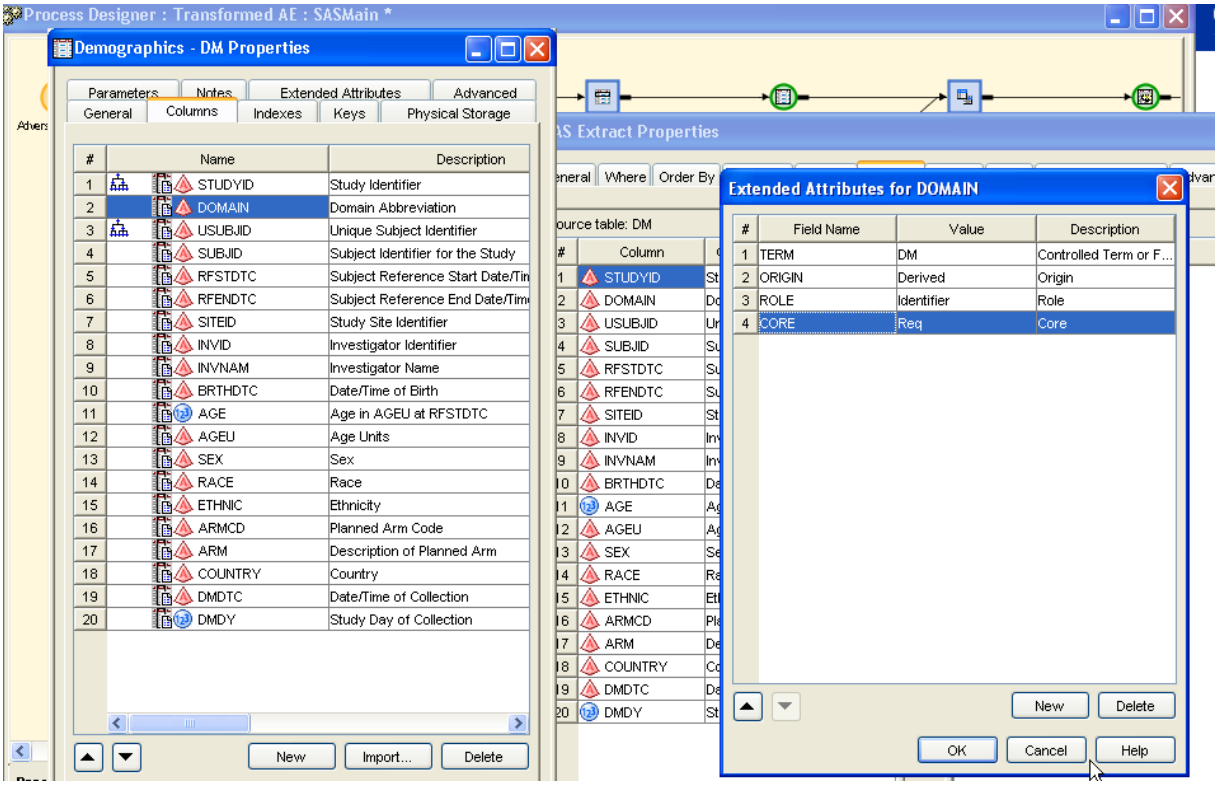
CHALLENGES

Despite identifying best practices and working with the user community to transition to this new environment there were still a number of challenges in the adoption of the solution.

Too much UI!

An easy to use point and click interface, one of the strengths of an ETL solution, call also be its biggest downfall with traditional programmers. As mentioned already, clinical programmers just want to write code. This is the case within DI Studio as well as other ETL solutions. Within DI Studio for example, to modify an extended attribute for a column, which might be needed for a SDTM column, the user has to first open the properties of the data set, navigate to the columns tab, and then open the extended attributes. This is one of many examples within the DI Studio interface where a user can easily get frustrated by the amount of pointing, clicking, tabs, and menus they have to access to change a single event. Figure 8 shows an interface just to change an extended SDTM attribute on a column. To mitigate this issue we extended the capabilities of the tool as described in the next section.

Figure 8



Scalability

Another issue identified during this project revolved around scalability. Using an ETL solution for clinical programming requires a different paradigm which these solutions are not necessarily designed around. In this implementation DI Studio is based on the concept of a metadata repository. The metadata repository is a collection of all the metadata for a collection of jobs, tables, columns, transformations, and other various entities. In the clinical world, you can think of a repository as a study. DI Studio has the ability to point at different repositories at login time, as well as what is known as a “User Repository.” A user repository is a repository that inherits from a study repository and can be tied to an individual user. User repositories allow for change management and the segregation of a user’s work from other individuals. For change management to work, you must create a new project repository for each user/custom combination that requires access.

Given the scenario described above, the implementation can create an explosion of repositories that must be created if a company decides to develop custom repositories to model each clinical study. If the company decides to create a repository for each study and then have multiple users creating Project Repository, these repositories can grow into the hundreds which is the original intent of the solution.

Change is Hard

Probably the biggest challenge we faced in implementing the solution was the same problem every company has with implementing new technology in the business process. In general individuals don't like change and managing that change is critical to the adoption of new technology. We attempted to support this challenge by implementing the best practices described above and hand holding the customer through the process. However, one component that is sometimes out of your control is the ability of the customer's management to support and enforce the change.

EXTENDING THE SOLUTION

Most ETL solutions provide you with the opportunity to extend their capabilities through a variety of methods. Some have an open API (Application Programming Interface) or a web service that allows you to build specific functionality that meets your business needs. In this project we extended DI Studio with a number of additional components. Below is a sample of two issues we addressed.

CDISC Editor

Problem

DI Studio provides support for managing SDTM metadata at the table and column levels. This metadata enables users to automate a number of tasks that might otherwise be tedious to implement by hand. While the out-of-the-box features provide the foundation for capturing this metadata, editing this metadata is unwieldy. For a table with numerous columns, the interface forces a user through a tedious workflow of clicking on each column individually to manage metadata. Ideally, users would have a single interface to manage all SDTM related metadata for a table, including table level metadata and column level metadata. The business need was to provide a single interface to manage table and column level metadata, greatly streamlining the user experience by reducing the number of clicks required to view and edit this information.

Solution

d-Wise developed the CDISC Editor plug-in as an extension to DI Studio to meet this need. The editor provides a user with a single interface to manage SDTM metadata at the table and column level and is only applied to tables that have been identified as SDTM domains. This plug-in opens the CDISC Editor at the domain level and displays a dialog with two tabs, one to manage the table metadata (Figure 9) and one to manage the column metadata (Figure 10).

Figure 9

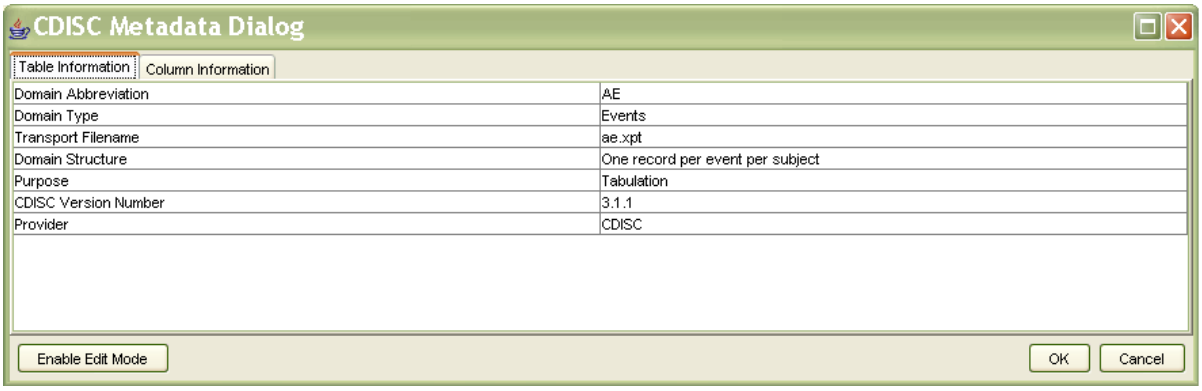
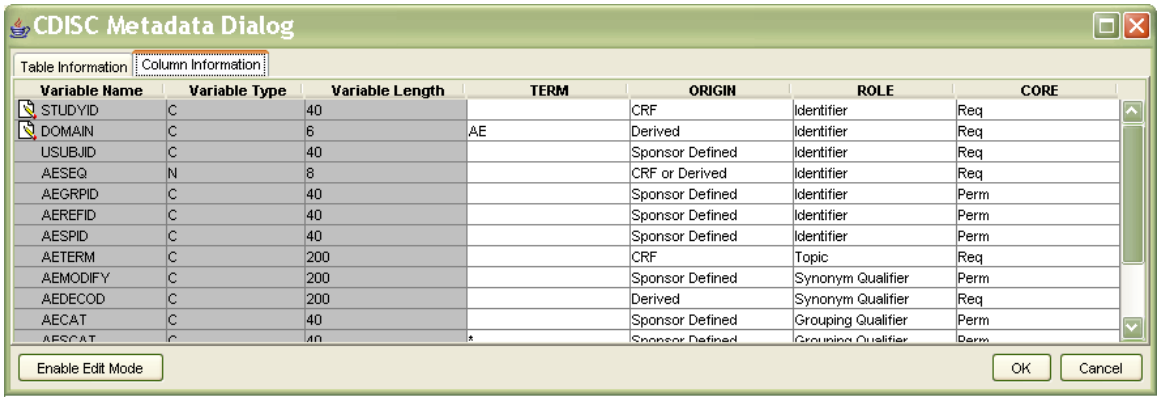


Figure 10



Assume a user is managing the SDTM AE domain and wants to view all columns that originate in the CRF rather than those that are derived. Without the CDISC Editor plug-in, the user would need to click on each column individually to view and edit the metadata associated with the column. Using the CDISC editor, all of this information is viewable and editable in a single interface.

SUPPQUAL SPLITTER

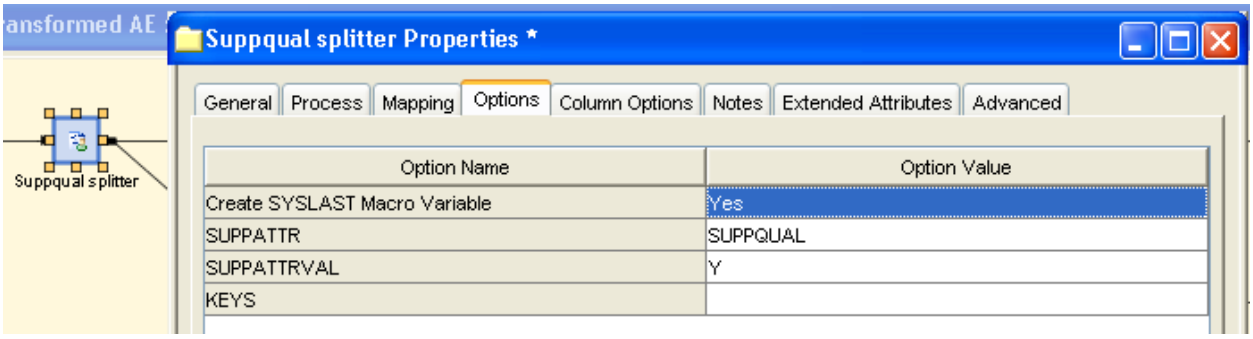
Problem

During the implementation of the DI Studio solution, the customer wanted to implement an SDTM+ standard. This includes the standard SDTM domains and variables as well as additional variables required for their operational needs. During this process they identified the need to define extended column metadata to allow the SDTM+ domains to be split into standard SDTM domains and SUPPQUAL domains

Solution

The Transformation Generator within Data Integration Studio, in combination with the metadata server API, was used to create a custom transform. The first step was to add an extended attribute to each of the SDTM variables. This attribute would identify whether the column should be included in the supplemental qualifier for the specific domain. The next step was to create a custom transform that used the API to read the metadata for the domain, pull out the variables flagged with this attribute, and transpose the data to match the requirements of the SUPPQUAL domain.

The SUPPQUAL Splitter transform shown in Figure 11 only needs the name of the attribute, the value of the attribute, and the keys for the SDTM+ domain. The node is then executed and creates both the pure SDTM domain and the associated SUPPQUAL.



This custom transformation used a combination of SAS code, calls to the metadata API and the standard structure of a SUPPQUAL domain to generate the necessary code to read the metadata, split the tables, and load the appropriate

SDTM standard domains. This solution provided the customer an automated process for maintaining their SDTM+ domains for operational needs but also allowed them the ability to generate both the pure submission ready SDTM domains and the associated SUPPQUAL domains.

POST MORTEM

At the end of the project we were able to implement a solution that met the needs of the business, but it involved a significant amount of work to close the technical gaps the deal with the challenges of change. The business is still working through the challenges of adopting the technology and the pains of the individual user but a well documented adoption process is mitigating this change.

Overall, the ETL solution could not meet the necessary requirements for the business out of the box and some significant customizations and plug-ins had to be developed to close those gaps. The decision to implement a full fledge ETL solution is to balance the cost with the ability for the solution to meet your requirements. At the end of the day the cost associated with customizing and closing those gaps in the process could outweigh the efficiencies you might gain.

In our opinion, if you can identify an ETL solution that is flexible, can meet most of your requirements, and is extensible, the long term benefit greatly outweighs any home grown system. The challenge is navigating the implementation process, closing the gaps in your business process, and helping users through the change.

SUMMARY

When people here the term ETL they immediately conjure up images of GUIs and tools that auto-generate their code. People have to get away from this concept and think of ETL as a process before they think about technology. At a minimum, your organization should strive to implement an ETL methodology to manage metadata, optimize your code and make components as reusable as possible. Separating the metadata from the source systems from the code will facilitate building a robust and scalable solution.

As part of implementing an ETL process, your organization might want to evaluate the use of an ETL solution to support this process. While ETL solutions are usually challenging to use out of the box developing best practices and extending the solution to meet your specific business needs is possible with many of the ETL solutions on the market. The ultimate question you will face is whether the cost and challenge of implementing this type of solution will provide your company the streamlined process and necessary efficiencies you need.

The concept of a traditional ETL solution is a difficult change for most traditional clinical programmers. At the end of the day the successful implementation of an ETL solution might not make an individual programmer's job easier or more efficient. However, in the long run, the ability to manage the metadata across an organization and provide an easy to understand set of reusable processes can only make a company more efficient.

CONTACT INFORMATION

Chris Decker, Life Sciences Director
d-Wise Technologies
Work Phone: (888) 563-0931 - Ext 105
E-mail: cdecker@d-wise.com
Web: www.d-wise.com