PhUSE 2009

Paper AD10

SAS ® Clinical DI – Will we need new MetaProgrammers ?

Simon Wilcock, ICON Clinical Research, Marlow, UK
Sireesha Potula, ICON Clinical Research, Marlow, UK

ABSTRACT

SAS ® Clinical Data Integration is a SAS tool which, through its interface to the SAS ® Metadata Server, allows users to quickly build data transformations and processes based on industry (and company defined) standards via a GUI front end.

The power of the tool is that it allows transformation jobs to be built, and easily visualized, based on metadata definitions bringing the flexibility to change data sources, targets and mappings with minimal programmer effort.

But is the tool counter-intuitive to the battle hardened SAS programmer?
Is building a Clinical DI transformation job more labour intensive then using standard Base SAS ® code?

Through some specific examples of regular programming tasks faced within the industry, this paper will look at how issues of data transformation may need to be approached differently and where some different skill sets may be required in order to reap the benefits of the approach.

INTRODUCTION

The purpose of this paper is to look at some of the day to day coding tasks that form the staple part of a SAS programmer's life within the Pharmaceutical Industry – and compare how the same activities are achieved within SAS Clinical DI.
Specifically the focus will be on data transformations, which is essentially the core purpose of the tool, and where data needs to be mapped from multiple source tables (typically raw extracts from a company's clinical database) to a standardised target format (either CDISC SDTM or company defined tabulation standards)

There is much talk in the industry around creating a metadata driven end to end process, from study definition to final reporting, and Clinical DI is positioning itself to be one of the key products within this stream.
This paper will focus on what skills are needed in order to use the tool and whether this will drive the need for a different type of SAS programmer in the future.

Definitions:

METADATA

Metadata is, quite simply, data which describes data. For example Metadata for a table would describe things like its column names, lengths, types, labels, formats along with information about location and creation date.

SAS ® DATA INTEGRATION STUDIO

SAS DI Studio is an application which allows the user to integrate many different forms of data based on information derived from the Metadata. It gives the user the ability to define both source and target tables, based on this metadata, and to use visual mapping tools to generate the transformations necessary – code is built dynamically in the background.

SAS CLINICAL DATA INTEGRATION

SAS Clinical Data Integration is, effectively, the Clinical extension to DI Studio. Amongst other enhancements, it provides a common metadata model for representing data standards that are based on CDISC – and provides the ability to switch certain attributes on and off dependent on specific individual company needs.

CDISC

CDISC is the Clinical Data Interchange Standards Consortium. Increasingly CDISC is becoming the preferred format for study data tabulation submission.

METADATA – WHAT'S THE BIG DEAL?

There is a lot of talk in the industry around using Metadata, but why is it considered so important?

The notion of Metadata has been around for quite some time and has been available in SAS for many years, accessible via the SASHELP library, for those programmers looking to make use of it.
The SAS Metadata Server, accessed by SAS DI Studio, makes this information much more readily available for processing and, as a result, becomes a powerful tool to drive reusability of programs.

Re-use of programs is a key component to productivity in the Clinical Trial reporting cycle. To this end, most companies have developed libraries of reusable macros in an attempt to standardize the processing of data from a wide variety of trial designs and protocols. There have been varying degrees of success in this approach – often dependent on the success of standardisation in other areas (for example CRF design, data collection forms, or reporting requirements)

To a large extent Metadata can be seen as providing the next step in this quest to automate the whole process.
In theory, because Metadata knows how the incoming data is structured, and how the output data has been requested, then it can set all of the necessary macro variables on behalf of the programmer.

This removes the need for study programmers to have to correctly set a long list of macro inputs, in order to achieve their desired results. This has potential for huge time savings and quality improvements especially in areas where macros, themselves, have become large and unwieldy.

Metadata can also be as equally effective without the use of large and cumbersome macros – allowing programmers to quickly assemble jobs using simple mapping techniques and then reusing these jobs by simply resetting the metadata for new studies

METADATA AND DATA TRANSFORMATIONS

Using metadata has particular resonance in the data transformation field – specifically in terms of taking source data, output from a CT-DBMS, and performing the necessary steps to create CDISC compliant SDTM format for regulatory submission.

Each study is unique and, despite efforts to standardise the source data, macros written to be reusable in this context tend to continuously evolve into large, multipurpose -and often unmaintainable - code monsters.

This can become further complicated by evolving data standards, for the target, as a result of CDISC version upgrades. In a CRO the complexity increases, exponentially, as each sponsor adopts a different version or "flavour" of SDTM.

Clinical DI offers a method to control these transformations and, through the use of metadata driven processes, the opportunity to significantly reduce the task of manual coding. In effect, Clinical DI takes on the primary role of code generator and leaves the human programmer to manage the metadata environment and perform tactical configurations where necessary.

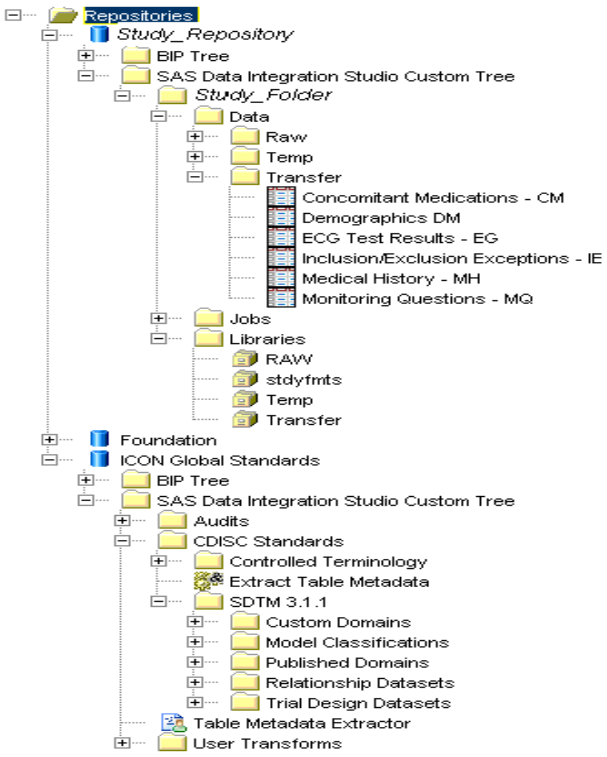
However, building transformations this way requires the programmer to approach familiar tasks from a different angle – and concentrate far more rigorously on the structure rather than the content.

WORKING IN THE METADATA ENVIRONMENT

DIRECTORIES

The principle concept to keep in mind, when within the SAS Clinical DI environment, is that the tool works with the metadata and not the physical data. As such the directory structures, which the programmer is presented with, are folders which contain the metadata and not the data itself.

Fig.1 Metadata structure in Clinical DI



In Fig.1, the directory “Data->Raw” does not contain the raw data as such – but the metadata that describes the structure of the raw dataset, itself.

The physical raw data can be viewed from within DI Studio but, actually, the tool will open the dataset from its physical location (the link to which is stored further down in the ‘libraries-> Raw’ directory)

Notice here that the SAS DI Custom Tree contains all the required metadata for CDISC standards. (Only 3.1.1 is displayed in this example)
This folder can be used to determine the target data structure which is subsequently seen in the Transfer folder.

The key to developing jobs within SAS DI Studio is to start with a complete understanding of the structure of the source tables to be used (held in Raw) and the target tables to be generated (placed in Transfer)

SAS DI Studio forces the programmer to focus more on the structure and less on drilling down into the detail of the dataset content. This is an important mind shift for developing code which is independent of the data itself.

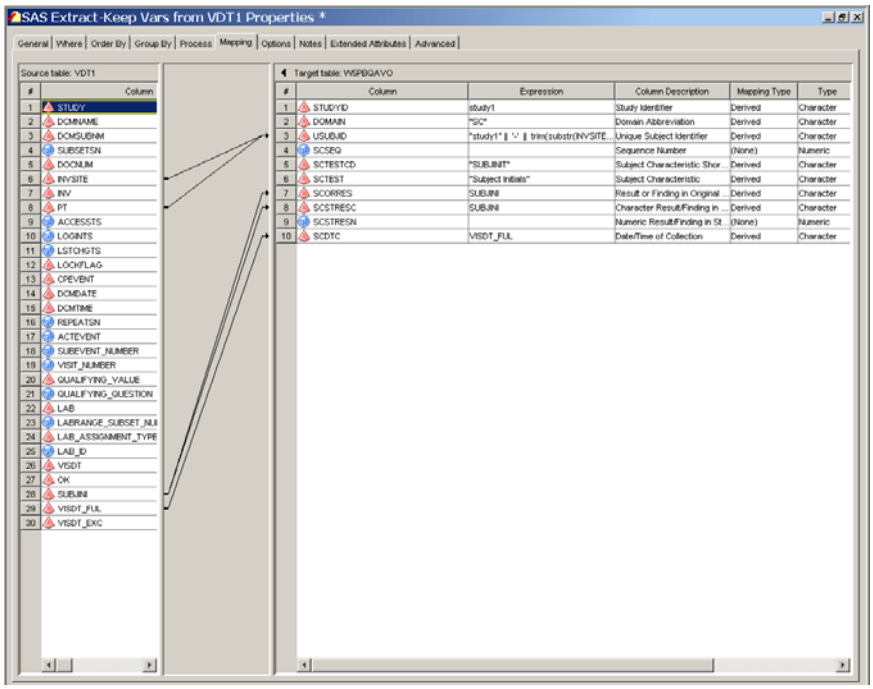
A SIMPLE EXAMPLE

The use of metadata, to achieve data transformation, can be demonstrated in the following example:
Data has been received from the database in a dataset called VDT1 – all the required data is there but the naming conventions, and a number of the formats, are non CDISC compliant. Furthermore a few variables need to be extracted and appropriate labels assigned.

This is not a difficult task in base SAS but it does involve a lot of routine coding (for example writing ATTRIB statements to ensure each variable is labeled correctly) and references to a data specification. It also requires close attention to detail to ensure the precise SDTM requirements are met.

In Clinical DI this is all preloaded. It’s a question of deciding which variable needs to be mapped across to where. In fact the AutoMap feature does most of this for the programmer.

Fig.2 A simple mapping

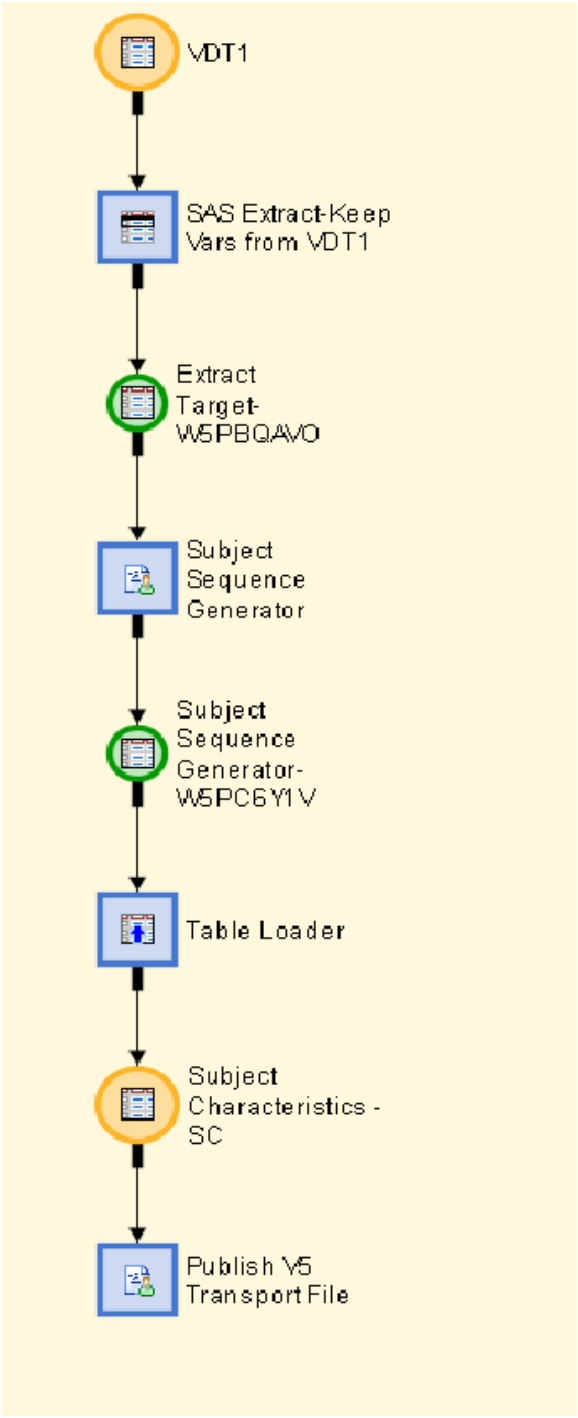


This makes the mapping of data to specific standards very quick and easy. The job is easily ported to other studies and any differences in input data or output specification handled quickly via the mapping interface.

SDTM comes preloaded into the system and as new versions are released these can be added – however, the principle applies to any data standards and companies can load and use their own metadata as appropriate.

This is particularly useful in a CRO environment where multiple different sponsor standards may be in operation. As a CRO does a transformation for a specific client it is simply a case of going to the directory which contains that clients standards and using those as Target.

Fig.3 An example job flow in Clinical DI



Behind the mappings there is also a process flow. The example displayed in Fig.3 is simply a case of extracting the data, performing the mapping, adding a sequence number (this is an ICON specific task) and then loading the data into the final output dataset, the CDISC domain, SC.

These processes are built using a simple drag and drop interface – transformations and metadata are simply picked up from the library and positioned in the required part or the process flow.

Of course, this is a very basic example, and programmers are often involved in much more sophisticated data transformations than this.

However, for the simpler CDISC domains we should not underestimate the benefit this approach brings – data which has been pulled directly from a proprietary database has been mapped quickly and efficiently to a predefined submission format with minimal intervention.
There is very little scope for 'programmer error' in terms of the final format because we are mapping *into* the required structure not building it via our own data steps.

There is also an additional benefit of traceability. The process flow shows where data has come from and where it is going.

Each step in this process is referred to as a "Node".
As jobs become more complex these process structures and nodes need to be built in a controlled manner, from the bottom up. It helps to design the jobs in the smallest possible units, testing nodes as they are added and ensuring any bugs are isolated early.

Programmers who are used to laying down code quickly, and then debugging later, may find this structure quite limiting at first – but getting the right building blocks is essential. This comes back to the notion, referenced above, of knowing what the beginning and end structures need to be.

A MORE COMPLEX TRANSFORMATION

Clinical DI Studio comes, packaged, with a number of purpose built transformations to cover the process steps required to map structures from source to target. These are the building blocks that the programmer needs to use in order to construct their jobs. Inevitably this will be seen as a restriction to those who are used to the freedom of "Open SAS".

However, with experience, it can be seen that most programmatic approaches to performing a specific transformation are still available within the tool. Dependent on the route taken, it can require the programmer to break down the problem into smaller components with the result that the more efficient methods soon become apparent.

A prime example, of this, is the simple transpose. This is a, commonly recurring, coding need in most day to day pharmaceutical activity – especially in the field of CDISC transformation.
Data coming from a database management system is usually in a 'horizontal format', as displayed in Fig.4, but the required SDTM format is 'vertical' and therefore a data transpose is needed (sometimes referred to as "fat to skinny").

Fig.4 An example of Physical Examination data received from the database in horizontal format.

VIEWTABLE: Work.Pe1

	SUBJECTID	SITEID	VISITID	VISITINDEX	FORMINDEX	PEORRES1	PEORRES2	PEORRES3	PEORRES4	PEORRES5	PEORRES6
1	001	101	1	1	1	ABNORMAL	NORMAL	NORMAL	NORMAL	ABNORMAL	NORMAL

The horizontal format makes sense from a database perspective, as this is more than likely how the data has been entered into the data capture system, with each test result stored in a variable under the test name:
The SDTM submission format, however, is vertical and has one variable for all the tests and one variable for the associated results as displayed in Fig.5

Fig.5 An example of SDTM vertical format

	SUBJECTID	SITEID	VISITID	VISITINDEX	FORMINDEX	DOV	_name_	col1
1	001	101	1	1	1		PEORRES1	ABNORMAL
2	001	101	1	1	1		PEORRES2	NORMAL
3	001	101	1	1	1		PEORRES3	NORMAL
4	001	101	1	1	1		PEORRES4	NORMAL
5	001	101	1	1	1		PEORRES5	ABNORMAL
6	001	101	1	1	1		PEORRES6	NORMAL
7	001	101	1	1	1		PEORRES7	COUGH
8	001	101	1	1	1		PEORRES8	NORMAL
9	001	101	1	1	1		PEORRES9	NORMAL

And so the dataset needs to be transposed. Every programmer in the pharmaceutical industry will have seen this kind of requirement, cropping up in a number of different scenarios, so it should be familiar to everyone.

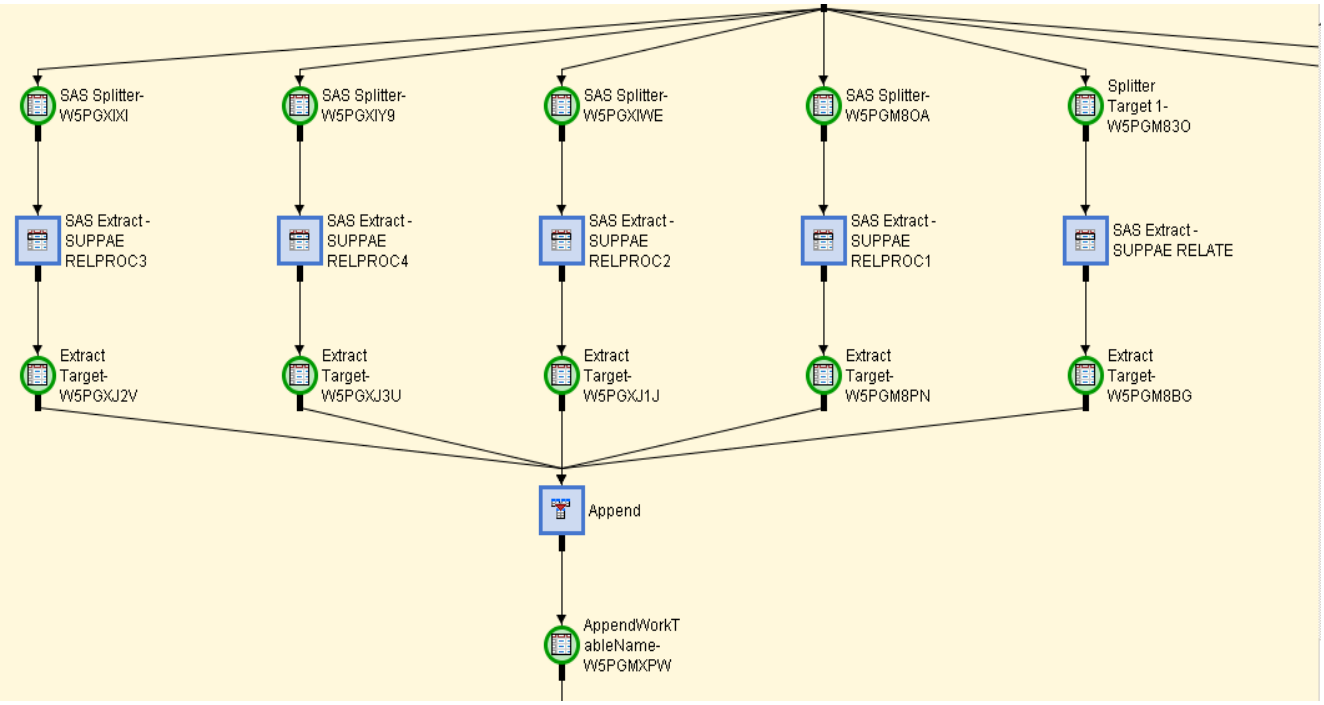
The problem can be approached in a number of different ways within Base SAS:

- Using data steps and output statements
- Using PROC SQL
- Using Proc Transpose

Likewise within Clinical DI, there are a number of different ways to achieve the same result.

The first, which is synonymous with the Data Step solution, is to break the data out into separate components using the SAS Splitter process and then to append back together.

Fig.6 Using the SAS splitter



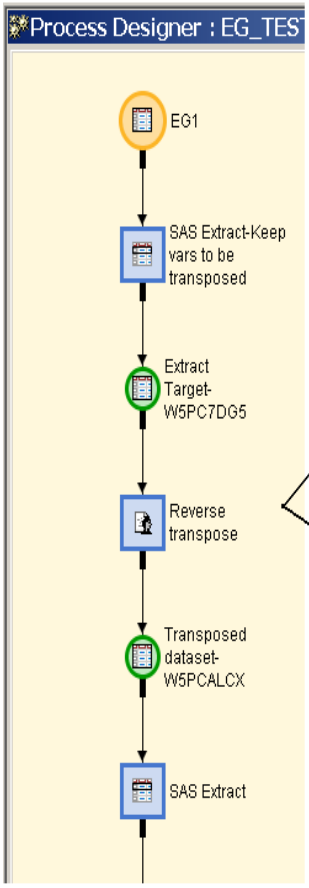
This feels like a somewhat 'clunky' solution but, as with the use of data steps and OUTPUT statements, it does have the advantage of being relatively easy to follow and to reproduce.
From a reusability perspective this may be the optimum way to create the job.

However, in terms of speed of development creating mappings using this technique can be quite slow and laborious. Therefore it is only really recommended for transfers which are going to be "build once, reuse often".

It should be noted that in Clinical DI, if a new mapping is added within a job, all subsequent nodes need to be remapped to ensure the overall integrity of the process. Therefore any changes, up stream to the splitting performed here in Fig.6, will necessitate each component to be recreated – thus making this solution more labour intensive in a changing environment.

Clinical DI also contains an SQL transformation which would seem to offer a second way of performing the transpose. Unfortunately, at the time of writing this paper, it was not sophisticated enough to perform the UNION processing required for the task. Although of course this may change in future releases.
However, there *is* the option to include user written code, into the process, and this was an approach that was, initially, investigated for performing the transpose task (demonstrated in Fig.7 below using ECG rather than PE) :

Fig.7 Embedding SQL in a Clinical DI job




```
/*---- Start of User Written Code ----*/
Proc SQL;
create table &_output
as
select invsite, pt, cpevent, ecgtim_ful, ECGCHQ,'HR' as EGTESTCD, 'Heart rate' as EGTEST, HR as EGORRES from &syslast
union all
select invsite, pt, cpevent, ecgtim_ful,ECGCHQ,'RR' as EGTESTCD, 'RR Interval' as EGTEST, RR as EGORRES from &syslast
union all
select invsite, pt, cpevent, ecgtim_ful, ECGCHQ,'PR' as EGTESTCD, 'PR Interval' as EGTEST, PR as EGORRES from &syslast
union all
select invsite, pt, cpevent, ecgtim_ful,ECGCHQ,'QRS' as EGTESTCD, 'QRS Interval' as EGTEST, QRS as EGORRES from &syslast
union all
select invsite, pt, cpevent, ecgtim_ful,ECGCHQ,'QT' as EGTESTCD, 'QT Interval' as EGTEST, QT as EGORRES from &syslast
union all
select invsite, pt, cpevent, ecgtim_ful,ECGCHQ,'QTc' as EGTESTCD, 'QTc Interval' as EGTEST, QTC as EGORRES from &syslast
union all
select invsite, pt, cpevent, ecgtim_ful, ECGCHQ,'INTP' as EGTESTCD, 'ECG Interpretation' as EGTEST, ECGINT as EGORRES from
&syslast
order by pt;
quit;
/*---- End of User Written Code ----*/
```

Fig.7 provides an example of how to embed some user written SQL code into a Clinical DI job.
The problem with this approach becomes one of reusability.
Our aim was to take this ECG code and make it more generic - bundling it up so that metadata could be passed through and thus making it reusable for the PE example, given in Fig.4, and other transpose needs.

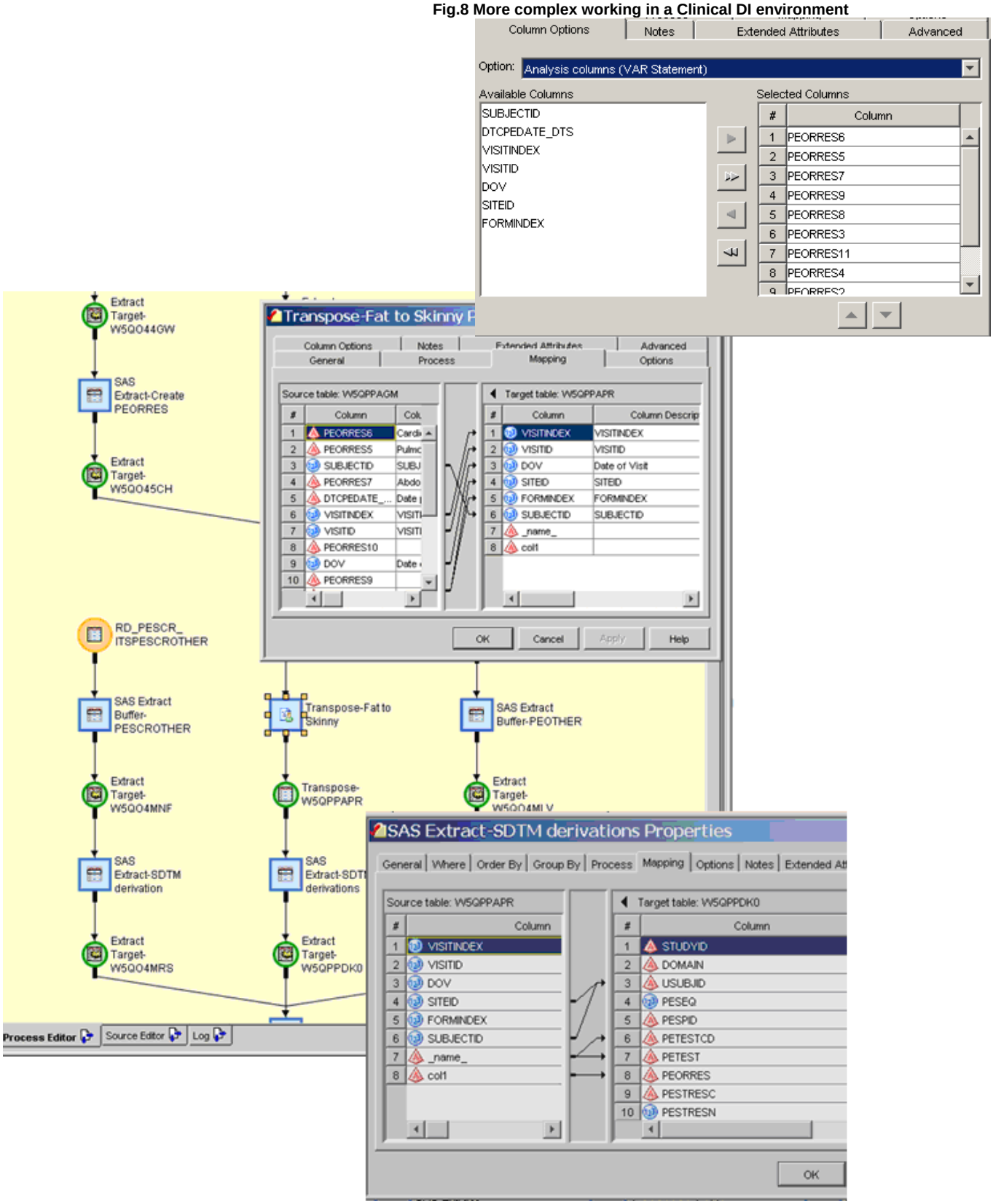
However, we made a decision, within ICON, to steer clear of developing ‘user specified’ transformations where ever possible as this inevitably results in issues of maintenance and ‘scope creep’.

The preferred approach is always to use the pre-provided transformations and, where we find these lacking, to have the conversations with the SAS Institute around making available in subsequent releases. (This has not proved a major problem to date but as we continue scale up there may be more transformations that we require)

The final approach to performing the transpose is to use the “Transpose” transformation provided within the tool. This is probably the most efficient way of getting the job done although the first few times it is used it can be very counter intuitive during the set up. The initial feeling of the programmers, using this at ICON, was that the transpose transformation had been designed with “vertical to horizontal” in mind and it took some time to understand how to use it effectively in the reverse scenario of “horizontal to vertical”.

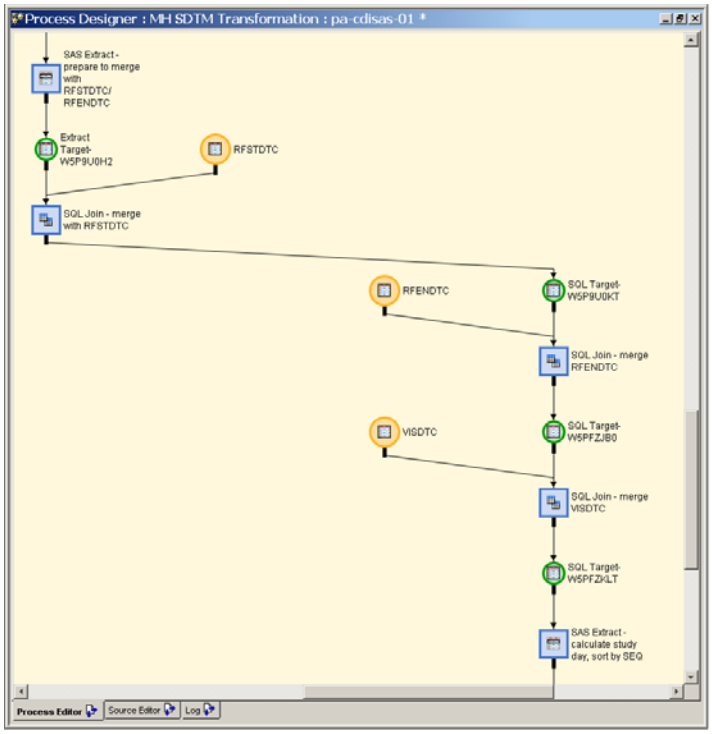
The screenshots in Fig.8 give a feel for the environment the programmers are working in – point and click interfaces with jumping between tabs to set the correct configurations and derivations. Initially daunting, it soon becomes a very quick task to navigate around and ensure the variable mappings are defined as required.

Fig.8 More complex working in a Clinical DI environment



PROCESSING WITHIN MAPPING

Fig.9 Using SQL joins



Within any data transformation step, there is, by necessity, more than just structural mappings going on.

Variables need to be derived, formats set and IF/THEN/ELSE clauses programmed up. Generally this kind of content processing is achieved via a CASE expression set within the properties of any given Node.

So, for example, a job step may include a number of SQL joins, as on the left.
The circles represent the datasets being merged in, or output, and the boxes the “SQL join” nodes.

To understand what is happening, in each of these steps, a programmer needs to drill down into the respective properties of each node to see how the data is being handled.

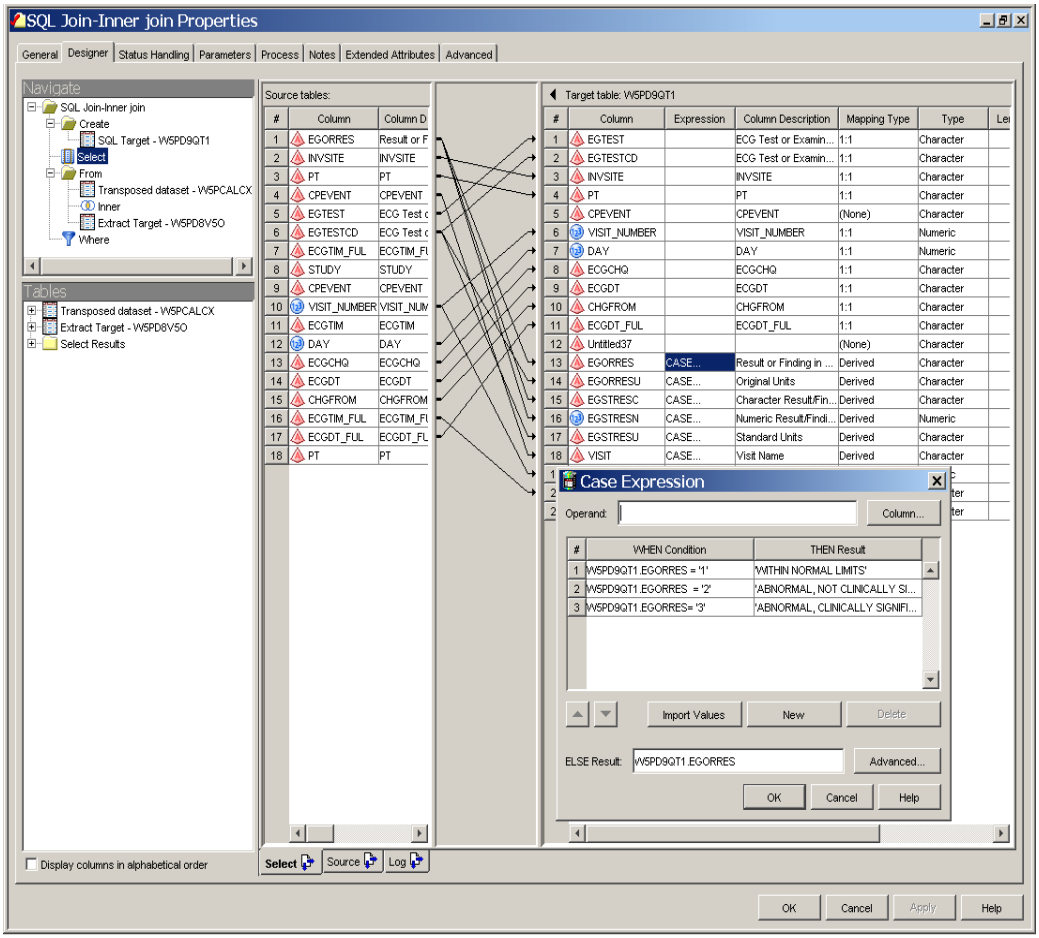
It is within these ‘properties’ windows that the formats are defined and the CASE expressions set.

Fig.10 Setting the Join properties

In the Fig.10 example, EGORRES is set to the appropriate character string based on the numeric result in the data

This processing, within the mappings, means that a programmer never really sees the code in its entirety laid out as a continuous code stream. This can be a difficult concept to deal with and is the reason why jobs need to be built and tested as small discreet units.

The code itself is generated at run time and it is not uncommon for programmers to run a *job purely to create a log file* – because, of course, the log file gives a view of the SAS code that has actually been submitted to the system. Experienced programmers need to resist the temptation to use the log to try to build the job using some kind of backward engineering.



TRANSFORMATIONS THAT DON'T EXIST

As stated above, Clinical DI comes out of the box with a number of transformations which should cover the majority of mapping needs. Inevitably, though, there will be complex pieces of coding required that can not be covered by the generic tools.

This has not caused a particular issue because the tool provides what is, effectively, a blank node for running user written code. This allows the programming team to use a "back door" to perform any highly specific coding that may be required.
By nature this coding should be a one off and plugging open SAS code into a job stream should be used with caution as it does not make the job easy to reuse – different metadata coming from a different study will likely cause the job to fail.

If the piece of code will be required on multiple studies then it is possible to define company specific transformations which can be loaded into the Process Library and then made available to all users in the system. These can then be reused as if they were normal SAS provided transformation.
In essence this becomes analogous to the Macro library and therefore processes and controls need to be designed around how these will be validated and used on other studies.

It would, theoretically, be possible to take an existing macro library and port all of the macros into Clinical DI transformations and then associate the relevant macro variable to the metadata. Although whether there would be benefit in this would be dependent on whether the current transformations fail to meet company needs.

This piece is key in the successful implementation of Clinical DI. It is relatively easy for experienced programmers to become concerned about a piece of syntax that may no longer be available to them rather than to step back and analyse what task needs to be achieved and how that can be done using the tools available.

Sometimes the way of achieving the goal may seem more circuitous but often, as a result, the job that is constructed has a higher level of reusability and structure.

CLINICAL DI AND THE METADATA APPROACH – Some food for thought

As has been highlighted, a number of times in the examples above, taking the metadata approach requires a programmer to think a lot more on a structural level than on a data level. Approaching the task of data transformation, from this perspective, is predicated on the fact that the data itself is clean.
Effectively the programmer is working a step away from the content of the data and therefore errors that have potentially slipped through from the database will not necessarily be picked up during this process and may not become apparent until further downstream – for example during statistical programming or reporting.

Of course, this is not always the case, and in some instances, mappings will fail as a result of unexpected data but inevitably this approach to data transformation does not have the same inherent ongoing data checks that are a result of programming data steps while viewing content.
Clinical DI does provide the ability to factor in data checks but this assumes that the programmer is looking at performing some defensive coding up front and has a feel for where some of the errors may creep in (for example missing visit dates). Whilst this is good practice it is not as thorough as a programming approach that is looking at the data as the code is being constructed.

This leads to a slightly different nuance to the roles being performed by programmers involved in data transformation. Those using base SAS could be thought of more as Data Analysts whereas those using Clinical DI are more akin to Data Modelers.
Both roles are, of course, important but the usage of Metadata driven processes should drive an organisation to think about how it wishes to use its resources and what some of the key skill sets need to be.

There are other elements of Clinical DI which can frustrate the more traditional SAS Programmer.
The inevitable specification changes, which happen after transformation have been set up, can prove difficult to handle, dependent on the nature of the change.
So, for example, adding an additional step in the middle of a job can be cumbersome as all mappings further downstream need to be recreated. It feels far easier in Base SAS where a Data Step can be added in the middle of a program without (seemingly) having to alter any of the subsequent code.
Clinical DI brings more rigour to the process as it forces the programmer to revisit the impact of midstream code changes. Dependent on the complexity of the task this can be a good thing or it can be seen as a hindrance.

CONCLUSION

Metadata concepts shift the focus away from simply 'programming' to understanding how data is structured and how it needs to be processed.
So the use of Clinical DI, within an organisation, - or indeed any tool based on metadata driven processes - will necessitate the rethinking of how job roles and responsibilities need to be structured. There will always be a need for standard SAS programmers but those working in groups focusing on data transformation may need to be 'meta-programmers'.

The term here has been used slightly tongue in cheek to highlight the fact that these are programmers who are programming on a metadata level.
In fact the users of Clinical DI are more 'Data Modelers' than they are 'programmers' – this distinction becomes more important as regulatory authorities move towards requesting data in specific ways.

The distinction is also key from a CRO perspective where multiple clients are requesting to receive data in their own internally defined data standards. For a CRO delivering the data, to the client, to a pre-agreed specification is paramount and often represents a large proportion of the data processing activity. Understanding data and data models is essential to providing customer satisfaction.

Therefore organisations, recruiting into a data transformation group, will increasingly need skill sets around data modeling rather than traditional SAS programming. The role becomes one of understanding data, and the relevant industry data standards, and not the language that processes it.

This should be seen as a good thing because, in the pharmaceutical industry, it is the data and its interpretation that is really important – not the effort that goes into getting it into shape.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Simon Wilcock
Associate Director, Data Integration and Standardisation
ICON Clinical Research
2 Globeside
Globeside Business Park
Marlow
Buckinghamshire
SL7 1HZ
UK

Web: www.iconplc.com
Email: Simon.Wilcock@iconplc.com
Phone: +44 (0)1628 496488
Fax: +44 (0)1628 496301