ABSTRACT

From its inception the SDF System was designed to tackle many of the difficulties that have plagued statistical programming ever since the first programs were developed to analyze clinical trial data. The system scope includes the following areas:

- Automating front-end and back-end data security
- Providing rapid access to raw and derived datasets to all sites within the organization
- Implementing large scale systems design and software engineering principles in the clinical trials programming domain to reduce the software development effort required for each statistical analysis
- Modeling the complex relationships inherent within clinical data to provide sophisticated impact analysis tools with the capability to manage and cascade change
- Implementing an object-oriented (OO) approach to the definition and reuse of clinical data standards
- Improving review and quality control processes by coupling the specification of a program together with it’s source code to create one logical entity
- Reducing the validation burden (through large-scale reuse) while at the same time increasing overall quality

INTRODUCTION

The Submission Data File System (SDF) was designed to support the development, validation, maintenance and execution of SAS programs required to create Submission Data Files (SDFs).

The system leverages a number of key technologies (SAS, Java and Oracle) to bring the following benefits:

- Enables collaborative working when Study Programmers are located in different sites (and countries)
- Enforces a hierarchical directory structure that is guaranteed to be consistent across development sites, data standards, therapeutic areas, products, indications, studies and analyses
- Accelerates the extraction of study data from the Clinical Data Management System (CDMS) by utilizing parallel processing technology
- Controls data access by automating the management of UNIX security groups
- Encodes data standards in metadata to drive the generation of SAS programs
- Includes a powerful Publish and Subscribe feature, which enables Standard Templates to be set up for use on many analyses
- Resilient to changes in both internal and CDISC standards
- Reduces the time and resources required to produce SDFs
- Automatically generates quality control checks into each program
- Provides a common environment and a consistent technical architecture which facilitates the transition of staff from one clinical project to another
- Ensures no lasting dependency between the system itself and the deliverables it produces. Once an analysis has been locked, all the components generated by the system are completely independent from the SDF application itself. In other words, the system is not required to review and/or execute the programs in the future.
A BESPOKE CLINICAL DATA EXTRACT, TRANSFORM AND LOAD TOOL (ETL)

**Extract** refers to the process of getting the raw data from your CDMS in a form that is accessible by those that are going to use it.

**Transform** refers to the process of transforming the data into structures that can be sent to the FDA as well as deriving key variables for analysis.

**Load** refers to the process of loading the raw and transformed datasets into a structure that is suitable for long-term storage and access.

HOW DOES THE SDF SYSTEM ACCOMPLISH THESE 3 TASKS?

The **extract** process is handled by the Clinical Data Extraction Tool (CDMS2SAS). Using parallel processing technology, it extracts study data from both Oracle Clinical and Medidata’s Rave EDC system and stores it in the SDF System’s directory structure. Not only is the directory structure standard across therapeutic areas, products, indications and studies, but the system also automates front end and back end data security using role based security profiles and automatic UNIX security group management. The system is not limited to using raw CDMS data. SAS datasets can also be imported into the SDF directory structure.

The **transform** process is handled by a number of system components:

- **The Macro Specification Editor (MSE)** allows complex data derivations to be specified, developed, tested and released into the SDF System. The MSE creates an XML document that encapsulates the, requirements, design, run time parameters and macro source code. During the development process it is possible to place the macro under source code control and check it in and out as development proceeds. Once the macro has been tested, the Macro Specification file is released into the SDF directory structure. Once released, it can be used to create many Derivation Templates, which essentially generate calling programs to execute the macro for a given...
analysis. Executing a Derivation Template creates a derived dataset, which can then be used in the creation of SDFs.

- **SDF Templates** are also used to transform data. They contain information that describes both the structure and content of SDFs. Amongst other things SDF Templates contain column mappings, CDISC metadata and row-based transformations. A row-based transformation (called a Map Function) is a transformation that can only operate on the columns in a single row of a dataset. Because SDF Templates can join many datasets together, all the columns, in the datasets which participate in the mappings, can be referenced in Map Functions. Additionally, Map Functions can use any SAS Function that can be called from the Select Clause of a PROC SQL Statement. This makes Map Functions reasonably powerful and a lot can be accomplished without having to resort to macros.

- The system also introduces the concept of **Publish and Subscribe**, which enables Standard Templates to be set up which can be used on many analyses. This gives us the power to control key attributes from a single template, yet at the same time, allow individual studies or analyses to take care of what’s different. The SDF Templates are then used to generate SAS programs that are executed within the system to create the physical SDF datasets.

- Code generation techniques allow **quality control checks** to be automatically generated into each program (e.g. data truncation checking, primary and candidate key violation checking and valid values checking), which would otherwise require detailed programming standards to be followed by convention.

The **load** process is handled by the **Program Execution Manager**, which executes all the generated programs in the correct order. It dynamically determines the dependencies between programs and executes programs concurrently whenever possible to ensure maximum performance. The resultant SDF datasets are then loaded into the SDF directory structure becoming both a deliverable for the submission and another data source.

**ADVANTAGES OF METADATA AND CODE GENERATION**

It is important to realize that just because a program is generated it does not lessen the responsibility for ensuring that it is correct and produces the required results. Exactly that same programming skills are required to construct an SDF using the SDF System as would be required to produce the code by hand.

Given the last statement, a good question would be why bother generating the code if the metadata describes everything anyway; why not just develop it by hand. There are many reasons why code generation makes sense; some of them are listed below:

- In the SDF System the generated code and the program specification and are essentially the same thing. This means that any changes to the program specification are automatically reflected in the generated code.

- The high-level design of every SDF Program is the same. They have the same sections in the same order. The only differences between SDF Programs are the:
  - Structure of the SDF that is being produced
  - Column mappings and map functions required
  - How datasets (raw and derived) are filtered and joined

- Because code is generated it is possible to ensure that certain checks are automatically performed in every SDF program. Currently the system checks for truncation, valid values and key integrity.

- The metadata makes it possible to perform sophisticated impact analysis. The system records exactly where every raw and derived column is used and can therefore display the interrelationships graphically.

- Using metadata and code generation we are able to better define and share standard data structures. Additionally we can encode some of the implementation details in the standards themselves (e.g. default mappings and map functions) and yet still make it possible to modify any specific implementation where it differs from the norm.

There are limitations to code generation. Firstly, it is not possible to provide all the functionality of a programming language in a system. After all, that is why languages exist. They provide the ultimate interface to program whatever is required. The SDF Template provides mechanisms to implement row-based transformations and to join all the lower level components (raw and derived datasets) together. More complex derived variables are produced using the Macro Specifications and their associated Derivations Templates as described in the previous section.
THE INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

The SDF System provides a standard IDE that supports the complete Software Development Lifecycle (SDLC). Using the IDE it is possible to specify, develop, maintain and execute SDF programs in a multi-user environment.

Figure 2: The SDF IDE

2. Tool Bar  4. Execution Manager  6. Editor Pane

Notice the Navigator Tab (# 8. in the screenshot above). This tab shows the SDF Directory Structure which is described in detail in the next section. As objects are selected for editing using the Navigator Tab, the Editor Pane (# 6. above) displays the editor appropriate for the object selected. For example, in the screenshot above, the SDF Template named core is currently being edited (see 5. Open Editor Tabs – notice that tab named core has the focus). Looking at the bottom of the screen you can see that the SDF Template Editor has a number of tabs (# 11. Editor Tabs). The ColMap tab has the focus. This tab shows how each column in the final SDF dataset is mapped. For example the study column (the first column in the list) is to be populated using oc.demo.study (this is a three level qualifier of the form <libname><dataset><column>).
The SDF Directory Structure consists of two components, the SDF Tree (displayed in the IDE) and the SDF UNIX Directory Structure. The SDF Tree is a logical directory structure, which is a combination of information (metadata) stored in the SDF Database and the underlying SDF UNIX Directory Structure. Each Amgen site running the SDF System has its own local SDF UNIX Directory Structure but share a common SDF Tree view and a single SDF Database.

The system synchronizes the individual sites’ SDF UNIX Directory Structures in real-time as directories (with the exception of Analysis directories which are collectively known as Abstract Nodes) are created, modified or deleted using the IDE. This ensures that a consistent clinical study classification scheme is maintained across sites, which will result in better reuse of SDF standards. Analysis directories (also known as Analysis Nodes) only exist in the SDF UNIX Directory Structure at the site that created them. However, they are always visible in the SDF Tree at all sites. This means that Statistical Programmers can work on an Analysis at a remote site as though it was local (with the exception of a slower system response time).

In order to support code generation, the system needs to rely on a standard directory structure and contents and therefore the only way to manage the SDF Directory Structure is using the IDE, which runs under a privileged SDF UNIX account. The operations that any particular user can perform using the IDE are controlled through their User Profile. Provided you have the appropriate permissions in your User Profile, the SDF UNIX Directory structure, clinical data, and generated programs can be accessed in read-only mode via UNIX.

Figure 3: The SDF Tree Structure
A CLOSER LOOK AT THE EXTRACT PROCESS – THE CDMS2SAS TOOL

CDMS2SAS was developed to expedite the extraction of clinical study data from Amgen's principle Clinical Data Management Systems (CMDS). It uses parallel processing technology to extract raw study data from both Oracle Clinical and Medidata’s EDC System (RAVE).

Extraction times are slower for locations that are geographically remote to the CDMS (e.g. UK, Australia and Japan). Prior to the rollout of this tool it would take more than 12 hours to extract study data for large phase 3 trials in the UK. By implementing parallel processing technology CDMS2SAS has reduced these times to approximately 1 hour.

Until recently data extraction times were not seen as much of an issue in the US. However, with the arrival of mega-trials things changed. Traditional data extraction methods were taking approximately 8 hours. By implementing CDMS2SAS extraction times have been reduced to a matter of minutes (with the exception of NORMLAB2, the normalized lab view, for Oracle Clinical studies).

Users have the option of performing an Interactive Extraction (where it is possible to select one or more datasets for immediate extraction) or alternatively setting up a Batch Extraction to extract data according to a specified schedule.

The CDMS2SAS tool has been integrated into the SDF System to allow the importing of raw data directly into the Data>CDMS folder belonging to the appropriate Analysis Node in the SDF Directory Structure.

DEFINING EXTRACTION PARAMETERS FOR ORACLE CLINICAL STUDIES

1. The Study selection list will only display studies for which you have been granted permission in Oracle Clinical. Choose the study from which you wish to extract data.

2. Once a study has been selected the Schema selection list will only display schemas for the selected study. Choose the schema from which you wish to extract data. The <STUDY> CURRENT schema represents the most up to date view of the study data. For a detailed explanation of the different schemas consult your Oracle Clinical System Administrator.

3. Select the type of views that you wish to extract data from. The system will use the SAS Extract Views by default.

4. Set the Output Directory path. This will be the directory to which the views get extracted. Pressing this button will display a standard file chooser window. Simply navigate to the required location and ensure that you select the directory before pressing the OK button. The full path name will be displayed in the Output Directory field on the main screen and the Extract button will be activated.

5. Once you have selected the Output Directory you can either run the extraction process in Interactive Mode or Batch Mode.
DEFINING EXTRACTION PARAMETERS FOR RAVE STUDIES

1. The Study selection list will only display studies for which you have been granted permission in RAVE. Choose the study from which you wish to extract data.

2. Once a study has been selected, the Schema will be set to dbo which cannot be modified.

3. Select the type of views that you wish to extract data from. The system will use the Conformant Views by default. See your EDC Administrator for information regarding the differences between the different view types.

5. Once you have selected the Output Directory, you can either run the extraction process in Interactive Mode or Batch Mode.

Note: Selecting a previous extraction job from the File menu provides a convenient mechanism to automatically set your extraction parameters.

4. Set the Output Directory path. This will be the directory to which the views get extracted. Pressing this button will display a standard file chooser window. Simply navigate to the required location and ensure that you select the directory before pressing the OK button. The full path name will be displayed in the Output Directory field on the main screen and the Extract button will be activated.

PERFORMING AN INTERACTIVE EXTRACTION

1. Once you have selected the Study and Schema, the Defined Views selection list displays all the associated views. See the section entitled Data Differences Between CDMSs for a complete description of the views extracted from each CDMS.

2. Use the dual list box selector buttons to select/deselect one or more views from the Defined Views panel and put them in the Selected Views panel. Drag & drop and multiple selection are also available.

3. Once you have selected the required views, initiate the extraction process by pressing the Extract button.

4. The system displays a progress bar and notifies the user of any SAS errors or warnings encountered.
SETTING UP A BATCH EXTRACTION

1. Once you have defined the extraction parameters, selecting the Manage Batch Definitions tab from the main screen will display the following screen. You can now setup a batch job to extract data according to a predefined schedule. All views in the selected study will be extracted.

2. First, set the time that you wish the extraction process to commence.

3. Next, either select the Weekly radio button and set the day using the Set Day selection list or alternatively choose the Daily (1-5) radio button if you wish the extraction to execute every week day. Selecting the Daily (0-5) radio button will schedule the extraction to execute every day including weekends.

4. As you set the time and select the extraction schedule the system automatically builds a UNIX CronTab entry in the New Cron Definition field. When you are happy with the time and schedule selections pressing this button will add the New Cron Definition into the Unix CronTab Entries panel.

5. To remove a scheduled job simply select the job in the Unix CronTab Entries panel and then press the Delete button. Remember to press the Save button (at the bottom of the screen) for the deletion to take effect.

6. If you delete a job in error, press the Undo button.

7. You need to press the Save button in order for the New Cron Definition to be written to your CronTab file. If you do not press the Save button the job will not be scheduled and will not appear in the Unix CronTab Entries panel the next time you access this screen.

8. The Unix CronTab Entries panel shows all your active data extraction jobs. The first time you access this screen it will be empty as you have not defined any batch jobs. You can use the scroll bar at the bottom of the panel to see the entire command. Most of the command is encrypted, however, the last section of the command will show you the selected Extraction Parameters.

9. To confirm that the data extraction process completed successfully review the SAS data extract log file (data_extract.log). This file can be found in the directory to which the data was extracted.

Note: Immediately after you change your UNIX password you will need to remember to press the Save button on the Manage Batch Definitions tab otherwise batch jobs defined using your old password will fail.
A CLOSER LOOK AT THE TRANSFORM PROCESS – SDF TEMPLATES

SDF Templates stored in Analysis Nodes (as opposed to those stored in Abstract Nodes) are called Concrete SDF Templates. Only Concrete Templates can be used to generate SAS programs that create physical SDF datasets. An SDF Template stored in an Abstract Node (called an Abstract SDF Template) simply describes the structure of the SDF along with its default column mappings. Abstract Templates do not create physical SDF datasets until they are copied or imported into an Analysis and subsequently have their columns mapped and contributing tables joined.

DEFINING COLUMNS IN THE COLUMN EDITOR – SUPPORT FOR CDISC STANDARDS

The Column Editor captures column definition metadata. The metadata contains physical attributes, such as Data Type and Length, and logical attributes such as Role, which is a CDISC attribute.

Although the SDF System is not limited to any specific data standard, it is proving to be an ideal vehicle with which to deploy CDISC SDTM 3.1.X and ADaM data standards across teams at Amgen. In addition, it also stores all the information necessary to automate the generation of define.xml (see the CDISC Attributes panel in the screenshot below).

Figure 3: The Column Editor – Includes CDISC Metadata
MAPPING COLUMNS

The ColMap tab is used to provide the column mappings and map functions required to populate the physical SDF dataset.

1. An SDF Column can be dropped from the final SDF dataset by checking its Drop checkbox. See the next section entitled Dropping Columns for a complete discussion.

2. The Map Type pull down list is used to set the Map Type for the corresponding SDF Column.

3. Column contains the list of SDF columns that have their Selected checkbox checked on the Columns tab.

4. Mapping contains the column mappings. Each SDF column that has a Map Type of Single Mapping can be mapped to a source column. If the SDF Column is not mapped it will be null (have a value of missing) in the final SDF dataset unless it is given a constant value in its Map Function. Mappings provide a fully-qualified, three-level, column reference of the form: <libname>.<dataset>.<column>. If the SDF Column has a Self mapping then only a two-level mapping of the form: self.<column> is used.

5. The Move Column Up/Down buttons reorder the current (highlighted) SDF Column within the SDF dataset.

6. The Automap button attempts to automatically map columns based on previous mappings (if the column has previously been mapped) or default mappings (if the column has never been mapped). Only unmapped columns will be affected. The automatic mappings are only created if the system can find a physical implementation for the Default Mapping Type, Origin and Default Mapping attributes. If the default mapping can be honored, the system will set the Map Type, Mapping and Map Function attributes.

8. The Map Same Name and Attribute Columns button attempts to automatically map SDF columns to the currently selected source dataset in the Source Data Panel (on the right-hand side). Mappings are based on matching column names and data types.
The **Map Function Editor** is used to enter PROC SQL code that can be used to perform row-based transformations. A row-based transformation is a transformation that can only operate on the columns in a single row of a dataset. However, because SDF templates can join many datasets together in the process of creating the SDF dataset, then all the columns, in all the datasets, which participate in the mappings, can be referenced in a **Map Function**.

1. Clicking on the [...] button on the right-hand side of the **Map Function** field in the **ColMap** tab opens the **Map Function Editor**.

2. Enter the Map Function code. A Map Function tutorial is provided in **Appendix A**.

3. If the Map Function is empty and the column has a **Default Map Function** defined, then pressing the **Use Default Function** button will copy the Default Map Function into the Editor.

4. A special syntax is used to reference columns in Map Functions. The **Source Data** panel (on the right) allows a Source, Dataset, and Column to be selected and the syntax required to reference the column is then displayed in the **Translated Value** field at the bottom right-hand side of the screen. The value can be copied (Ctrl-c) and pasted (Ctrl-v) into the Map Function text area.

5. Press **OK** to save the Map Function and close the **Map Function Editor**.

6. When a Map Function is saved (by pressing the **OK** button) the system will automatically perform a syntax check if it is possible to do so. Any errors or warnings will be displayed in the window below. Fix the errors in the Map Function and save it.
JOINING DATASETS

The Joins tab shows i) the list of datasets that require joining and ii) the joins tree, which is a graphical representation of the datasets already joined.

Each time an SDF Column is mapped on the ColMap tab the system determines if the new mapping introduces a new dataset that must be added to the list of tables to be joined to create the SDF dataset.

When a column is un-mapped, the system determines if the mapping represented the last reference to the dataset. If it did, the dataset will be removed from the Joins tab. If the dataset has subsequently been joined to other datasets then this will result in the other datasets being moved into the list of datasets that require joining so that they can be rejoined to a different dataset.

1. The Joined Data Files panel displays a diagrammatic representation of the joins that have been specified.

2. The Un-Joined Data Files panel displays the list of datasets that still require joining in order to create the SDF dataset.

3. The selected (highlighted) row in the joins tree shows the following Configurable Join Parameters:
   - SQL Join Type: set to Left Outer in selected join
   - Joining Key: set to Patient in selected join
   - Filter: <None selected>
   - No Duplicates: Not Checked in selected join

4. The tree indentation shows how tables are joined. Any table in the tree is directly joined to each of the tables below it that are indented one level to the right. Therefore the selected (highlighted) row above indicates that oc_demog participates in two joins. Firstly, oc_demog is Inner joined to oc_investigatordetails on the Center key (Note: The Inner join means that only rows in oc_demog for which a matching Center number can be found in oc_investigatordetails will be retained). Secondly, the result of that join then forms the right-hand side of a Left Outer join with sdf_sv_core_common (Note: The Left Outer join means that the table on the left, which is sdf_sv_core_common in this case, will not lose rows even if the table on the right does not have a row with a matching key). Join nesting directly impacts the order in which joins are executed. This is discussed in the section entitled Join Order Precedence.
EXECUTING AN SDF PROGRAM

There are two methods of executing an SDF Template. It can be executed in isolation or it can be executed with dependencies. Both of these methods can be triggered from the Code tab.

1. Every time the Code tab is selected, the SDF Program named <SDF Template Name>.sas is regenerated in real-time and written to the Source Code > SDF folder in the corresponding Analysis Node. This ensures that the program is always up-to-date with respect to the latest changes made to its meta-data.

2. Pressing the Run Code button performs the following operations:
   a) Executes the SDF program producing a SAS log file named <SDF Template Name>.log which is written to the Source Code > SDF folder. The log file can be viewed by selecting the Log tab.
   b) Displays a message indicating the success or failure of the execution. Error and Warning counts are displayed if appropriate.
   c) Writes the resultant SDF dataset to the Data > SDF folder. The dataset can be viewed by selecting the Data tab.

3. Pressing the Run With Dependencies button performs the following operations:
   a) Generates the execution plan (equivalent to a Make file). The plan determines the execution order of all the SDF Templates and Derivation Templates required to guarantee that the target SDF is up to date.
   b) Runs the execution plan in a threaded manner so that all programs within the same execution level run in parallel. A log file containing the execution plan and the concatenated logs of all the programs executed can be viewed by selecting the Log tab. Additionally, individual SAS log files (for each of the programs executed) are written to the Source Code > SDF folder (in the case of dependent SDF Templates) and to the Source Code > Derivation folder (in the case of dependant Derivation Templates).
   c) Displays a message indicating the success or failure of the execution. Error and Warning counts are displayed if appropriate.
   d) Writes the resultant SDF and Derived datasets to the Data > SDF and Data > Derived folders. The target SDF dataset can be viewed by selecting the Data tab.
   e) Generates a UNIX script file named <SDF Template Name>.csh which is written to the Source Code > SDF folder. This file is not executed by the SDF System and is generated so that in the event that the Analysis is unlocked in the future, and the SDF system is not available (or is incompatible with the old Analysis), the programs can be executed directly from UNIX without having to determine the execution order manually. The .csh file calls all the dependant programs in the correct order to recreate the target SDF.
THE DEVELOPMENT OF A CUTTING EDGE INFRASTRUCTURE

A CLOSER LOOK AT SDF’S SAS ARCHITECTURE

The SDF system’s business and presentation layers have been exclusively developed using Java; it’s interaction with SAS being managed using SAS Integration Technologies (IT). Whenever possible, the development of the SDF application has leveraged the extensive Java libraries provided with the SAS IT and AppDev Studio modules.

SAS code generated by the SDF system and that encapsulated in Derivation Macros is executed on one of two UNIX servers running SAS 9.1.3. The SDF application treats these SAS servers as a single load-balance cluster. The metadata (machines names, ports etc) that defines this logical cluster is stored in a SAS Metadata Repository located on one of the two UNIX SAS hosts (Fig. 4). Prior to submitting SAS code the SDF application first obtains connection information about the cluster from the SAS Metadata Repository; communication being mediated by the SAS Metadata Server (Fig 4).

We saw earlier that the SDF Application allows generated SAS code to be run with or without dependencies (see the section entitled EXECUTING AN SDF PROGRAM). When the SDF application executes code with dependencies it first determines the list of SDF and Derivation templates on which the target SDF template relies. This list is organized into execution levels that honor the interdependencies between SDFs and Derivations. Programs within a given level are executed in parallel.

In the same way that it is possible to execute an SDF / Derivation Template with dependencies it is also possible to execute and entire analysis with dependencies. The ability to execute independent programs in parallel, within a scaleable clustered SAS environment; makes it is easy to see how SDF can bring significant performance improvements over traditional Biostatistical programming techniques.

Figure 4: The SDF SAS Architecture
OTHER IMPORTANT COMPONENTS OF THE SDF SYSTEM INFRASTRUCTURE

In addition to the SAS architecture described above, the SDF System infrastructure includes of the following components:

- **Best of Breed Development Tools:** Java, Oracle, XML
- **Automated Security:** Automated UNIX security group management to reduce reliance on other departments. Application security using LDAP authentication and role based system access.
- **Metadata Repository:** Provide Biostatistics with a centralized Oracle database to hold metadata
- **Define Standards:** Encode standards in metadata whenever possible to support process improvement initiatives. Provide tools for SDF standard directory structure creation and management.
- **Application Deployment:** Implement application deployment technologies to reduce maintenance and support issues
- **Dependencies Management:** Manage system and infrastructure external dependencies to support comprehensive impact analysis when environmental changes are proposed and implemented
- **Validation and Automation:** Use IT Methodologies, Source Code Control tools, Configuration Management tools, Automated Testing tools, and Bug and Enhancements Tracking tools to consistently move the project up the Software Maturity Model

CONCLUSION

This paper has provided a high-level overview of few of the capabilities of the SDF System. This system is transforming the way Amgen approaches and implements statistical programming. The major benefits include:

- Rapid software development time scales made possible by maximizing the reuse of data standards which include metadata to create content as well as structure
- Improved quality
- Improved validation documentation
- Improved program design – modular design ensures no hidden or hard to understand code
- Consistency of implementation across individual teams
- The ability to cascade changes across multiple implementations of a given data standard
- Rapid access to CDMS data
- Enforcement of a Standard Directory Structure
- Flexible role-based front-end and back-end security

CONTACT INFORMATION

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

Marcus Bloom  
Amgen  
One Amgen Center Drive  
Thousand Oaks, CA, 91320-1799  
Work Phone: (805) 313 4477  
Email: marcusb@amgen.com

David Edwards  
Amgen  
One Amgen Center Drive  
Thousand Oaks, CA, 91320-1799  
Work Phone: (805) 313 4634  
Email: edwardsd@amgen.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.