Paper 025-2008

Reinvent Legacy Software with SAS®, the Web, and OLAP reporting

Andrew P. Ford , PJM® Interconnection, Norristown, PA
Troy B. Wolfe , Qualex Consulting Services, Inc., Miami, Florida
Shiva Srinivasan, PJM Interconnection, Norristown, PA

ABSTRACT

Generation adequacy calculations and assessments have been used at PJM since the early 1970s. The models and calculations are derived from statistical values, based on years of historic performance data. The applications implement probability theory to predict future PJM generation reserve requirements. Recently, the legacy systems based on Fortran and rudimentary database systems have been implemented using SAS and Java web applications. This paper describes our experience in inventing new ways to implement the old legacy capabilities using new SAS code, SAS Tables, web interface and SAS® Integration Technologies. Topics discussed include:

- Measurement of correct calculations
- Measuring code performance; arrays and modular design
- SAS Tables and Database
- Design of the web interface; interaction of the web interface and SAS Integration Technologies
- Use of Futrix® OLAP reporting in the process.
- Coordinated Maintenance and adhering to Standards.

INTRODUCTION

The PJM control area has employed the use of probabilistic models and statistical techniques over its long history of reliable electric service. Please see the PJM web site for further details about its organization at www.pjm.com. These models and techniques are based on fundamental concepts first described in the professional society of Institute of Electronic and Electrical Engineers (IEEE) papers as early as 1947. Throughout the more recent Industry trends to transition to a competitive marketplace, a clear focus on adequacy and reliability has been maintained at PJM. New rules and processes are under development to further change the adequacy model to embrace new market structures while honoring the fundamental tenets which have served PJM, its members and the public over the years. This paper describes a part of this transition process whose focus is on generation Adequacy in PJM

Reference 2 presents further discussion about PJM, its organization, project management methods, and the generation Adequacy analysis performed which provides a flexible and accurate analysis environment in performing forward looking adequacy assessments.

Several projects were involved, over a five-year period, involving up to 22 people at one time in the process. The projects started by performing a proof of concept, to determine if SAS and java based set of products were the best alternative for implementing the project deliverables. Professional disciplines of the team members included computer programming, windows systems engineering, computer networks, electrical engineering in power systems, mathematics, statistics, data warehousing, GUI development and design, computer graphic arts, and load forecasting. The project team met weekly to discuss status, details and to measure project deliverables. A weekly working meeting was implemented so that many members, with different disciplines, could work in the same room to share insights and collaborate on individual aspects of group assignments. Each team member could come and go as his schedule and work assignments dictated. This working environment insured that all aspects of the group deliverables were considered while the various individuals were working on their individual assignments. This improved the efficiency of all team members. This fostered a collaborative environment that solicited and respected each team member's perspective in providing the best solutions in the invention process. Those in leadership on the project team considered all perspectives, to make choices and decisions on what specific methods to implement, typically based on available limited

resources, funding, and meeting project schedules. The intent of this paper is to provide a summary of the more than 320 SAS programs and Macros, 150 JSP programs, over 16,000 lines of SAS code, over 10,000 lines of Java code, more than 25 SAS data mart tables, more than 10 Oracle tables and 3 schemas, and over 15 documentation manuals involved in these project activities.

MEASUREMENT OF CORRECT CALCULATIONS

The initial and driving force for the project activities were the growth in the PJM geographical footprint and membership as well as the effort needed to maintain the legacy FORTRAN computer applications. To determine the exact equations being used in the FORTRAN code, a detailed flowchart was developed. This was done by specifically describing each subroutine and common area parameters, iteratively inserting debug statements, and updating the existing documentation. A small representative model was developed to exercise all calculations performed in FORTRAN and was implemented using Excel. For each aspect of modeling and calculations used in FORTRAN, a worksheet in Excel was developed to mimic the calculation. In this manner the detailed and complex calculations were expressed in an Excel spreadsheet. Once each worksheet was developed the SAS coding development was started. The SAS programmers could review the spreadsheet; collaborating with the power engineers, mathematicians, and statisticians to invent a robust and flexible structure for correctly implementing the calculations for loss of load expectation (LOLE). The cumulative capacity probability table calculations shown in appendix B, represent the majority of the solution time of the calculations but is only one specific aspect of many involved in the calculation process.

Appendix B has further discussion and shows sample calculations concerning the cumulative capacity probability. For a region that has over 1,100 units, such as the PJM RTO, these values can become very small. This shows how attention to detail was critical as the various daily LOLE values could be very small, on the order of 10^{-20} before the value could be considered not significant. This spreadsheet has over 20 worksheets that combine specific algorithms into a coordinated LOLE calculation.

SAS code was directly developed from Appendix B, Figure 1. As such, it was very modular, structured, and distinct. The objective of this part of the process was to transfer the legacy calculations, embodied in complicated, industry specific, FORTRAN subroutines, into commonly understandable and easily documented results that could be measured and verified. However this code suffered from this design process and was in need of attention to performance considerations.

MEASURING CODE PERFORMANCE; ARRAYS AND MODULAR DESIGN

It was of utmost concern that the SAS output matched the legacy system. In order to facilitate this and ease maintenance on the system, a modular approach to the design was chosen. Each module helped to isolate the calculations into smaller more manageable blocks, so that not only were calculations easier to handle and understand during design time, but troubleshooting issues became a much easier task, since each module's output could be easily inspected.

Figure 1.
Example of Master Code (Appendix A steps 3 - 11)

```
%Assign_Weeks(&Year, &yr1, &yr3, …)
%Get_Daily_Peaks(&Year, &startWeek, …)
%exclude_Days(&exclude_Weekends, …)
%Five_Week_Average(&oneWkAvgDs, …)
%Weekly_Sums(&dailyPeaksDs, …)
%Levelize(&fiveWkAvgDs, &oneWkAvgDs, …)
%Add_Growth(&levelDs, &growthDs);
%Per_Unitize(&growthDs, &trendsDs, …)
```

(Note ellipses added to indicate more parameters were used)

For instance, in the code written to calculate load models, a master macro was written that called all the individual modules and passed relevant information as needed to create a final model. Figure 1 shows a simplified version of this code to highlight the fact that each macro was called in succession to perform a very specific task. See Appendix A, WeekPeakFreq Logical Flow for more details about calculating load models.

Notice that macro variables (which were defined at the top of the code) were used as parameters to allow ease of maintenance. Each module called returned a table to be passed into another module further down in the process.

IMPROVING PERFORMANCE

This approach worked very well, especially during the design and creation phase. However, once that phase was complete, testing revealed a dramatic problem—the new modular code was extremely inefficient. While

the modular approach is ideal for maintenance, it has a tendency to incur significant performance degradation.

As an example, once we achieved accurate results, we found the time to produce those results took upwards of 2 to 3 hours versus the 5 minutes it took in the legacy system! The problem could be found in the fact that the modules were designed to be small incremental calculations and therefore were called many times over.

CUMULATIVE CAPACITY PROBABILITY

In this case, there is a process that calculates a Cumulative Capacity Probability table. This table could be very large and needed to be created for each week of every year in our study. The process for creating these tables (as one was created for every week) was to cycle through all the generators that were considered "on" for that week and add them to the table. Generators that were considered "off" (i.e. not available due to maintenance or some other reason) would be discarded for the purposes of this calculation.

Using the macro approach described above was extremely inefficient for this purpose. Originally, a macro received a weeks worth of generator information and built the table for that week. The problem is that each pass took significant time to complete and it was done for each of the 52 weeks in a year.

The answer to this problem was found in the use of arrays in the data step. Instead of passing over the same data 52 times, we passed over the data once and saved the answers in arrays. A separate array was created for every week and each array retained all of the necessary calculations. The result of this action created a table that was wide (many variables) instead of long (many rows). We found that the time it took to create 52 weeks worth of calculations with arrays could be accomplished in about the same time it took to do one week's calculation individually without arrays**. This change was the most significant performance improvement in reducing the run time from 2 hours to 5 minutes.**

CALCULATING A TWO AREA SOLUTION

One other area that needed to be addressed was what we referred to as the two-area automatic solution. In this portion of the process, it requires merging two areas, using the large cumulative capacity probability tables, eight different times. With the straight modular design it meant that each of these eight different runs were done individually. The array structures were once again called upon to more efficiently perform these calculations.

In merging the two areas to calculate a solution, originally data merges were used (since the information was in tables). However, instead of doing individual merges, we found importing all of the tables into an array of a single data step was very fast. We would read each individual table within a single data step, resulting in a very wide data set. Now instead of merging tables, we could simply access the data via its array index number to do our calculations. This is how the legacy system, using FORTRAN, was able to achieve a faster response time than our initial modular design. The FORTRAN code was able to utilize arrays that resulted in maximum efficiency. Instead of having to look up rows in a table, the program was able to go directly to an array index and obtain the information it was seeking.

In addition to reading tables into arrays, it was necessary to assign our arrays as temporary arrays. This saved time, since temporary arrays are not written out, so they could be stored in memory and not incur the cost to write to disk. This explains why we are able to read tables in so quickly.

PARALLEL PROCESSING

The final step in our performance enhancement was to fully utilize our server environment when running a calculation. In general, SAS will use one CPU per session unless the new THREADS option is invoked. However, that option does not apply to everything in your SAS session. To attain full parallel processing we called a separate SAS session for each solution year using the SYSTASK COMMAND as follows:

```
/* Execute batch process                                              */

systask command %sysfunc(quote(&&&startCmdVar))
              taskname=&batchTaskName
              status=&batchStatusVar;
/* waitStmt is a concatenated list of all batch processes run */
waitfor _All_ &waitStmt;
```

The WAITFOR command was used to wait for all years to complete. When all years completed processing, the controlling program (which is the process that submitted the above code for each year) compiled the re-

sults, created a report, and stored details out in tables accessible to the users via SAS® Enterprise Guide and Futrix OLAP reporting environment. Related details and figures are shown in reference 2, Figures 1- 3.

SAS TABLES AND DATABASE

A formal SDLC (Software Development Life Cycle) model is adopted within the PJM SAS Environment. The SAS Environment at PJM is divided into a Development (DEV), Test (TST), Production (PRD) AdHoc, and a Production Batch server. All applications going through a formal Change Management process are run on the SAS PRD Batch Server and all other programs/processes are run on the SAS PRD AdHoc Server. A separate folder tree exists for each SDLC phase observed in the SAS Server environment. Files in these environments are organized functionally by appropriately named folder/subfolder hierarchies, each of which is further divided by the owning department. Each environment's table structure is as follows:

SAS Adhoc: Web application input files, Debug Folders, SAS Enterprise Guide Projects

SAS Apps: Location of SAS Programs used by ARC, categorized into sub folders Probabilistic Reliability Index Study Model (PRISM), Week Peak Frequencies (WkPkFq), and Administration (Admin).

SAS IO : Common Tables, Data mart Tables, Control Tables, Parameter Tables, Summary Tables

TABLE MAINTENANCE

Beyond identifying the different table types, these tables need to be maintained by using the administration function of ARC. To do this, a generic web interface was designed to allow administrators to execute any SAS code or macro. This interface, which was driven by a table itself, could be customized to present users/administrators with input boxes for required and optional parameters of the requested job. Among the tasks that could be performed in this interface include:

- Updating maintenance tables,
- Running both the legacy application and new application with an input file,
- Loading Oracle warehouse data to the SAS Datamart, and
- Loading data from other systems

Of great interest in this discussion is the process used to prepare the data to create a PRISM case for the annual Reserve Requirement Study (RRS). This maintenance item is called "The RST programs" and consists of several phases to get the data in proper shape and more importantly to verify its correctness. The integrity of the application ultimately comes down to having accurate data. The RST programs address the validity of the data relating specifically to all the generators within the PJM RTO footprint. Generators considered outside the PJM RTO footprint are also brought in during this process.

RST PROCESS

The process to bring in and validate generator data consists of eight phases. This process was created to automate and provide structure, and also to allow users to "step in", validate, and update information as the data moves through the process. With this approach, the users were given the best of both worlds: automation and a structured process, with full flexibility to intervene and correct information along the way. The eight phases and purpose for each phase are summarized as: 1) Obtain updated Capacity market list of generators, 2) Create new internal list of generators, 3) Merge statistics with list of generators, 4) Read in data from www.pjm.com/Rstudy, 5) Read in external generators to identify duplicates, 6) read in external generators from public data source, 7) Obtain data for future generation changes (https://www.pjm.com/planning/project-queues/queue-gen-active.jsp) 8) Read in future generation changes for external generators.

Note that each phase was designed so that it could be run separate from the others (although they need to be run in sequence). Phase 1 can be run over and over until the users are satisfied it is correct. Then phase 2 can be executed multiple times until that data is considered accurate. As the data moves through each phase, it is validated and put in the format needed to create PRISM cases for the annual RRS.

The RST process is just one aspect of the data that is read into the SAS Datamart. The administration screen function is used to import all other necessary information (i.e. load history, load forecast, generator statistics, etc.). Once the verification process has been completed, the web interface can be used to read the data and build PRISM case studies. Related details and graphics are shown in reference 2, Table 2 and Figure 4.

DESIGN OF THE WEB INTERFACE; INTERACTION OF THE WEB INTERFACE AND SAS INTEGRATION TECHNOLOGIES

SAS Integration Technologies with Enterprise Guide was used to build from an existing installed technology. The webAF™ Software using SAS® AppDev Studio™ was used to create a servlet which set up a communication link to SAS Integration Technologies via an IOM bridge in order to access the power of SAS. Establishing the communication link enabled the ability to write JSP code with scriplets to build our screens, execute SAS commands and call SAS macros to retrieve data.

ESTABLISH A WEB SESSION

The servlet created and invoked a web session, initiating a connection with SAS, and authenticated users. Consider how the hook to SAS was set up. The servlet has two public methods that are called from the web, doPost and doGet. The doGet method simply passes control to the doPost method. The focus will be on the doPost method as shown below. A session needs to be created utilizing the Java class HttpSession from the servlet package.

```
public void doPost(javax.servlet.http.HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response)
    throws javax.servlet.ServletException, java.io.IOException{
    // Retrieve the identifier of the current session
    // or create one if it does not exist
    HttpSession session = request.getSession();  //create the session
```

ESTABLISH A CONNECTION WITH SAS

After a web session is established a connection to SAS can be made. To create a connection, classes supplied by webAF Software were used as shown. Note that there are four distinct classes instantiated and each is saved to session variables to be recalled in JSP scriplet code to access the active SAS connection.

```
Connection connection1 = new Connection();  //instantiate connection object
// Set attributes for the connection object to the SAS IOM Bridge
connection1.setHost(hostName);
connection1.setPort(portNum);
connection1.setServerArchitecture(serverArchitecture);
connection1.setAccessMethod(accessMethod);
connection1.setInitialStatement(initialStatement);
// Save out to session variables
session.setAttribute("connection1",connection1);
BoundConnection bc = new BoundConnection(connection1);  //bind connection ob-
ject
session.setAttribute("bc", bc); // Save out to session variables
rocf = new Rocf();  //instantiate ROCF class
// ROCF is responsible for maintaining all of the connections
// in order to gracefully terminate any sessions upon application exit.
session.setAttribute("rocf", rocf);  // Save out to session variables
br = new BoundRocf(rocf);  // bind rocf object
session.setAttribute("br", br); // Save out to session variables
```

RUN SAS CODE

The last thing the servlet did was authenticate user credentials. This was accomplished using a SAS dataset that stored usernames and encrypted passwords. To authenticate a user a SAS macro for that purpose was run. To do this, two objects were supplied by webAF Software:

1. SubmitInterface – to provide a facility to submit SAS code, and
2. SclfuncsV2Interface – to allow SAS to pass macro variables back to a web session (in this case a variable that determines if a user was successfully authenticated)

Both of these objects needed to first be instantiated and then set to talk to rocf as follows:

```
com.sas.sasserver.submit.SubmitInterface submit = null;

com.sas.sasserver.sclfuncs.SclfuncsV2Interface iscl = null;
try {
  submit = (com.sas.sasserver.submit.SubmitInterface)rocf.newIn-
           stance(com.sas.sasserver.submit.SubmitInterface.class,connection1);
  iscl = (com.sas.sasserver.sclfuncs.SclfuncsV2Interface)rocf.newIn-
           stance(com.sas.sasserver.sclfuncs.SclfuncsV2Interface.class,connecti
           on1);}
```

Once these objects were set up, the ability to make calls to SAS and retrieve results was accomplished opening up the power of SAS. The following is an example of how to execute SAS macro code:

```
// Create the string used to include the SAS code
String runCmd = "%include '" + sourceDir + "Verify_User.sas';";
// Submit the sort statement
submit.setProgramText(runCmd);
runCmd = "%VerifyUser(".concat(username).concat(",")
runCmd = runCmd.concat(password).concat(",").concat(newPassword).concat(");");
submit.setProgramText(runCmd);
```

INTERACTIVE WEB INTERFACE

The web tier was initially started as a small application to meet the needs of the users who needed to view the snapshot of SAS datasets created by the many SAS® ETL type processes. After some successful implementation it was decided to explore the power of the web interface and expand it to be the graphical interface for our analysis. The users could now create a study definition, select a load model, select the geographical zones to be included in the study, add/modify/remove generators for a particular study, perform administration type activities such as maintain studies, add users, add/revoke privileges to users, run ETL type SAS programs, and submit SAS code as batch processes. In brief it became a robust web application fully utilizing the potential of SAS analytics. The ARC web Application maintains the same format with respect to the web interface for both Week Peak Frequency Model (WkPkFq) and PRISM (Probabilistic Reliability Index Study Model). Once the user logs into the application he is given the option to either run the WkPkFq application, the PRISM application or perform Admin type tasks. Both the WkPkFq and PRISM model follows the same flow of information; 1) Case definition, 2) Model Parameters and defaults, 3) Submission of SAS batch proecess, and 4) E-mail confirmation of process status.

The WkPkFq application is within the ARC suite of applications. Week Peak Freq is an application that allows a user to build a load model based on historical peak loads. A load model is expressed in a table that has a mean and standard deviation for each of 52 weeks. Hourly peak loads are read from the SAS data mart based on the criteria defined by the user to create the weekly mean and standard deviations.

Once a user selects their criteria for the study, they may submit it. Week Peak Freq will run SAS in the background. While it is running, a status will be displayed to the screen to show the user progress. An email is sent to the user when the load model calculation is complete, so the user does not have to wait for the process to complete before creating and running another model or logging off the system. To get a better understanding of the Logical Calculation flow that the SAS macros perform, refer to Appendix A: Week Peak Frequency Application Logical Flow.

The various GUI items act as parameters to the SAS programs for creating the load model. Most of the GUI fields i.e. text items, radio selection, list items and check boxes have self explanatory names. For example, Report Type radio selection can either be a TREND or a PLOTS model, the Start Year is the first year for historical loads to be included in the study and the duration is the amount of years to be included from the Start Year.

The Zones option allows a user to select the geographical regions that will be included in the study. The user can add or subtract zones by pressing a "Zone Selection" button. The zone selection screen allows the users to select the zones to be added or subtracted either as check boxes with names of the zones/subzones or by clicking on a geographical map. The selections (shown in reference 2, Figure 6) are dynamically linked to the SAS server, and passed back to the zone selections GUI shown to the user. Not only is the user shown a graphical rendering of his selections in the top portion of the screen, the list in the bottom portion of the screen is dynamically created, based on the SAS data mart, allowing for a true representation of latest available data. One of the useful pieces of information displayed is the time frame of zonal load data available or

stored in the database. This information helps validate invalid time frame selections made by the user. This also serves as a check between the underlying database changes and the experienced user's knowledge of assessing modeling needs.

A forecast growth factor must be supplied in order to create a load model. The forecast growth factor is based on the anticipated growth between the first and last winter of a given PJM load forecast report. A user can supply a forecast growth factor or have Week Peak Freq calculate it based on the seasonal load forecasts defining the Forecast report Year, Growth Start and End Year. The calculation will be based on the selected geographical zones, in the add and subtract list. A user can select either a 5 days (Week days) or 7 days period. Once all the parameters are in place then a user can display, by clicking the "Create Model" button, the user selections and the calculated growth factor (if automatic is selected) in order to verify requested selections before submitting the parameters for a SAS batch process to run in the back ground. Once the "Submit" button is clicked control is passed to SAS for the load model calculations. The user has the option to wait for a confirmation screen to appear after the SAS process has been completed successfully or to log off and get the confirmation via an email. The results are stored in the Summary tables as mentioned in the SAS TABLES and DATABASE section.

These applications also have on-line help which serves as very useful user manual documentation for how best to use the various and numerous capabilities. These on-line documents are updated by the user community to share the maintenance tasks for supporting the application and to allow the most experienced users to document the intricacies of the application. The application used for this is efficient for user documentation purposes. By clicking on the link WKPKFQHELP, the user can navigate the on-line help.

Another highly used user tool is the DirectoryHelp link. This link, on the upper right hand GUI of all screens, allows correct navigation to all the various tables across all environments. This is an efficient navigation tool for users to find any and all inputs and result data, related to a given set of analysis. This link, is on all GUI screens due to user feedback of its usefulness. Related details and graphics are shown in reference 2, Figures 4 – 7.

USE OF FUTRIX OLAP REPORTING IN THE PROCESS.

The decision for implementing an Online Analytical Processing (OLAP) and reporting environment was based on the need for efficient assessment of data quality used in the applications and technical analysis. PJM sought a robust, flexible, user changeable, environment where a known set of data could be repetitively processed to assess the data quality and to ease assessment work. The ability to produce aggregate summaries, drill into specifics to prove correctness of data, and transpose the data for a related geographic area or time period are tasks that can take considerable resources. Needed was the ability to define and specify the relationships among varied and complex SAS tables. Only the experienced users knew the relationships and how to correctly manipulate the data. The key issue was that the relationships and manipulation techniques were not documented or specified clearly. The improvement in moving to an OLAP was metadata, or storing data about data.

The SAS data mart and relational Oracle tables in the PJM information warehouse have a defined structure that is in a matrix, stored by rows and columns. The data used in the various OLAP data queries, transpositions, and various aggregations requires the data to be transformed into many views. These views (i.e. summaries) are then used to define the OLAP metadata so that the relationships are formed for production data. The transformation process is performed by a series of SAS macro programs. This gives the users flexibility to control which data sources get updated and when. This process is performed by the ARC Administration GUI. About ten structured data mart and oracle tables are transformed into over eighty summary views for use in the OLAP.

Once the SAS macros have successfully run, the OLAP is used by the user to perform the assessment of data integrity. The ability of the user to point and click to transpose, aggregate and drill down into data, keeping all data relationships intact allows for an unprecedented abilty to assess data not previously possible due to time limitations and limited resources. Another significant benefit is in the clear documentation of the data relationships that exist in the underlying data models.

The Futrix administration menus are used to create and control the OLAP report and metadata specifications. The administration menus allow many important features to be specified such as security, access definitions, report definitions, and metadata. The metadata management defines the various data table relationships that before were only available to those with a high level of experience.

The rows and columns of metadata information show how each piece of data relates to the other data views.

Another aspect of the Futrix OLAP is that when the user is clicking thru viewing the data, the most efficient data view, or the view that displays the fastest, is always displayed, per the table relationships described in the metadata.

The user reporting screen defines the options available in viewing the data. There are several selections on the top tool bar. They help to navigate among views, redefine the rows and columns in the display, and to filter the report. The “Drill with” option is to show a more detailed data for the current data view. The “drill new meta item” allows for changing the index for viewing the same data. In other words, if you are viewing data for a given time period you can change it to view the data by geographical region. Consistency across all views is assured as production quality SAS code, and a defined data relationship, is the source of the underlying views.

The most detailed increment for time is hourly, an integrated hourly average peak. The most detailed increment for the geographic areas is by sub-zone. But there can be many different summaries, whose basis is on different hourly time periods and different combinations of geographical sub-zones.

The overall objective is to provide the ability to report data for a given period of time for a given area of interest. However, the difficulty is in either double counting or missing information due to various summary methods. For example, consider two individual areas that have a daily peak at 3:00 pm and 5:00 pm, respectively, with the aggregate of these two areas having a peak at 4:00 pm. Summing the individual peaks will not yield the correct aggregate total as neither peak value is at the time of the aggregate peak (4:00 pm).

Therefore, peak loads cannot be naturally rolled up between geographic and time dimensions for the user reporting screen. Peak loads may occur at different times for different sub-zones within a given region and different aggregations of areas may include the same sub-zone area. For this reason, moving from a sub-zone level and rolling up to a zone view of the data will provide incorrect results if the peak loads are simply summarized together. Therefore, the zone level of information needs to be calculated straight from the details and stored as a separate summary level table. Futrix makes it possible to define a different table for each dimension crossing that is summarized appropriate to that view of the data (these tables are created as defined above in the ARC administration GUI). Each of these tables is defined inside a Futrix Hybrid Online Analytical Processing (HOLAP) structure that allows a user to view the data as if it were one table. From a users perspective, it all appears as a seamless roll
up of data that otherwise would be very difficult to view as such.

The metadata table names are shown in the column headings, for example one table name is “Study Area Peak Loads”. Each row in the matrix represents a description of an important piece of information in the metadata. Column names from each table are then “aligned” to each metadata description. All reports can then be built off of these descriptions. In this way, columns in different tables, even with different names, can be treated in the same way for all reports. For example, each of four individual tables have a column named “ARC_Year” that align to the metadata description labeled “Planning Year”, but only two tables contain a column that aligns to “Region”. Related details and graphics are shown in reference 2, Figure 8.

COORDINATED MAINTENANCE AND ADHERING TO STANDARDS

SAS programs, for the ARC application, were carefully written adhering to “SAS Principles and Standards” developed during the course of the projects. Consistent use of good programming standards helped improve readability, efficiency and reduced maintenance.

A simplified Software development life cycle (SDLC) approach was adopted to fully utilize the potential of implementing SAS standards. The design of the SDLC model used is shown below which includes design, development, change management, implementation and coordinated maintenance.

a. ***Collecting User requirements***: This is to define deliverables and measurements for judging successful completion. This is a statement of need from the user’s perspective. These requirements are usually a “Verbal” description of what the program should do (clearly explaining the calculations and analysis involved in data collection terminology)
b. ***Analysis and Planning***: This involves generating the high level program specifications and documentation of statements containing the functional specifications, feasibility of the project, proposed timeline, the resources required and a consideration of alternative solutions.
c. ***Design***: The Developer creates program specifications that take the requirements and translates them into database and programming terminology. Collaborating with the users and gaining their buy-

in is invaluable. This phase determines how the system or program will get the work done. A development work plan and a validation protocol are documented during this phase.

d. ***Coding***: Programming activities are now performed using the requirements, the specifications, and the validation protocol as references. Programs are developed using formal programming guidelines which are geared towards producing readable and maintainable code. Iterative general reviews with users to gain feedback keeps deliverables on target.
e. ***Testing***: Programs are then tested for all possible bugs by the developer while ensuring minimal reliance on the validator to identify errors. Regression testing is also performed to ensure that the functionality of the program has not been affected due to modifications.
f. ***Integration***: This phase involves integrating the program to new/existing systems/programs and verification of overall integrity.
g. ***Acceptance***: . With consistent user community involvement in the development process, significant issues for gaining the users' acceptance and sign-off are mitigated.
h. ***Implementation***: Any changes to a validated program must follow a well-defined Change Management process. In this phase, documentation regarding why a change is needed (is it a bug fix or program enhancement) is performed. Once the change is made, the program must be revalidated through extensive testing before it is put back in production.
i. ***Maintenance***: A change in the business logic or an enhancement to an existing process presents opportunities for program modifications or development of new programs. All maintenance activities should again follow the steps a – g of the Software development life cycle.

Regardless of the number of steps involved in a project, documentation, testing, and user involvement are indispensable stages required to ensure desired outcomes. For more examples and SAS code to help develop SAS standards, see reference 1 titled "A perfect recipe to maintain SAS programs".

The essential ingredients used to develop SAS standards are Clarity, Consistency, Maintainability, Transfer Capability and readability which could be broadly categorized as follows:

1. ***Naming Conventions***: Use appropriate, meaningful and unambiguous names for libraries, program files, datasets, macros, datasets, variables, formats and macro variables
2. ***Compatibility***: Deals with the flexibility of SAS programs to be shared across platforms, across members and across different environments. For example, Windows environment variables were used to define the SAS environment in which the programs were executed. Standardization of the CodeRoot, DataRoot, LogRoot and ReportsRoot included LIBNAME statements in the driver program so that the SAS programs could be moved across environments from DEV to PRD.
3. ***Documentation***: Numerous explanatory comments were added to explain module description, conditional and subsetting logic, statistical calculations, and database queries. Every SAS program included a prologue which gave the maintenance programmer valuable information such as macro variables used, databases accessed, calling and called programs, and history of changes.
4. ***Appearance***: A single style of writing SAS programs was adopted and used such that the visual appearance of the SAS programs mirrored their logical flow.
5. ***Efficiency***: Adopting a linear and modular style reduced debug time and increased efficiency. The SAS programs were exchanged among programmers and reviewed for efficiency. Efficiency techniques such as Indexing, Sorting, WHERE rather than IF/ELSE, use of RETAIN for initializing constants, LENGTH statements, and arrays were used.
6. ***Maintainability***: The programs were written keeping reusability in mind. Use of macros, nested calls, autocall facility, and adopting a program flow approach rather than a single program does it all approach.

Based on experience one can conclude that adopting a new standard increases development time. However, not adopting good standards can cost more in terms of debugging, maintenance and in re-coding. Hence, spending some extra time initially saved considerable time later.

CONCLUSION

The development process of the various applications provided opportunities for all team members to be innovative while always being challenged to complete deliverables on schedule, with limited resources and constant oversight by those in leadership of the project process. The integrity of the application ultimately comes down to proper manipulation of accurate data.

Realized Improvements include;

- A robust SAS server environment, with capable and experienced staff to maintain and make appropriate system enhancements.
- Documentation that is clear and understandable by experienced professionals in various disciplines.
- Standards and appropriate techniques, including a Software Development Life Cycle, in a production environment.
- Appropriate levels of security profiles based on business needs, appropriateness and separation of duties.
- An increased efficiency and productivity implementing assessments for changing Power Industry needs.
- Audit trails and suitable documentation of methods used in Adequacy assessments.

REFERENCES:

1. A Perfect Recipe to maintain SAS® programs, Shiva Srinivasan, PJM Interconnection LLC, Norristown, PA, Shailaja P Ramesh, Tech Dynamics LLC, VA, NESUG 2006, Philadelphia. , Administration and Support Section, http://www.lexjansen.com/nesug/
2. Reinventing a Legacy System with SAS®, the Web, and OLAP reporting, Andrew P. Ford and Shiva Srinivasan, PJM Interconnection LLC, Norristown, PA; Troy B. Wolfe, Qualex Consulting Services, Inc. , Miami, FL, NESUG 2007, Baltimore, Applications Big and Small Section, http://www.lexjansen.com/nesug/
3. PJM Reserve Requirements, PJM Manual 20, http://www.pjm.com/contributions/pjm-manuals/pdf/m20.pdf.
4. Reserve Requirement Study Report, http://www.pjm.com/committees/planning/downloads/20070822-item-03-2007-reserve-requirement-report.pdf.
5. Microsoft Project (MSP), project management software, http://office.microsoft.com/en-us/project/default.aspx.
6. IEEE Standard Definitions for use in Reporting Electric Generating Unit Reliability, Availability, and Productivity, IEEE Power Engineering Society, IEEE Std 762-2006, March 15, 2007.
7. Power System Reliability Evaluation, Roy Billinton, 1970, Gordon and Breach, Science Publishers.

ACKNOWLEDGMENTS

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.

Futrix is a registered trademark of Futrix Software Limited in the USA and other countries. ® indicates USA registration.

PJM is a registered servicemark of PJM Interconnection, Norristown PA. ® indicates USA registration. Used with permission, all other rights reserved.

Eric W. Mayhew and Diane L. Smith of PJM Interconnection and Steve E. Wagar of Qualex Consulting Services Inc, made significant contributions in the deployment of the applications discussed in this paper.

CONTACT INFORMATION

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

Author Name	Andrew P. Ford	Troy B. Wolfe	Shiva Srinivasan
Company	PJM	Qualex Consulting	PJM
Address	955 Jefferson Ave	423 Castle St	955 Jefferson Ave
City State ZIP	Norristown, PA 19403	Geneva, NY 14456	Norristown, PA 19403
Work Phone:	(610) 666-8964	(315) 781-0243	(610) 666-4524
Fax:	(610) 666-2296		
Email:	ford@pjm.com	troy.wolfe@qlx.com	srinis@pjm.com
Web:	www.pjm.com	www.qlx.com	www.pjm.com

Appendix A

A portion of the process used in week peak frequency calculations is shown in appendix A. There are a total of 15 steps in the logical flow of this calculation process. A compete logical flow is shown in reference 2, appendix A.

Week Peak Frequency(WKPKFQ) Logical Flow

Step	Program	Line #	Description
1	WPF_Main		Accept input for the following: ▪ Growth Factor ▪ Growth Start Year ▪ Growth End Year ▪ Report Selected ▪ Add Zones ▪ Subtract Zones ▪ Exclude Holidays Flag ▪ Holiday List ▪ Exclude Dates ▪ Exclude Weekends ▪ 5 Days or 7 Days per week
2	WPF_Main	289 - 318	Create subset data for the range of years specified including the Add Zones and excluding the Subtract Zones.
3	Assign_Weeks		Assign week numbers to the data by determining the first Monday of the Growth Start Year, assigning that week as Week 1, and then assigning week numbers to the remaining weeks.
4	Get_Daily_Peaks	132 157 205 245 261	Determine the daily peaks in the data as follows: ▪ Subset the data to include an 85 week period and only the zones specified by Add Zones and Subtract Zones. ▪ Summarize the data for each hour of each day. ▪ Roll the data up by date. ▪ If a day is missing, use the average of the previous and the subsequent days as the day's value. ►load_mwh = round((last_mwh+load_mwh)/2) where last_mwh = mw value for 2 hours ago load_mwh = mw value for current hour ▪ If multiple consecutive days missing, generate error.

Appendix B

Cumulative Capacity Probability Calculations

The primary equation for the cumulative capacity probability calculations is shown at the top of Figure 1. Also each of the individual state values shown in columns E to H, for rows 7 to 16 were essential in measuring what values are correct and determining any sources of errors considering both a detailed data model and detailed calculation process. Rows 3 - 6 are necessary only for the spreadsheet processing, to easily automate the spreadsheet calculations.

Figure 1

Cumulative Capacity Probability table used to measure SAS code results

A	B	C	D	E	F	G	H
	Area 2	**4 unit test sample**		**$P_{i,j}=P_{i,j-1}*(1-EFORd)+P_{i-c,j-1}*EFORd$**			**Final**
		8/8/2007		i=row # on spreadsheet= MW out			**Cumulative**
				j= column # on spreadsheet = unit number			**Probability**
				c= capacity/table step size			
Row	**number of**	**Units**	**0**	**1**	**2**	**3**	**4**
#	**adder**	**Unit MW**	**0**	**10**	**20**	**30**	**40**
		Total MW	**0**	**10**	**30**	**60**	**100**
		Unit/10	**0**	**1**	**2**	**3**	**4**
1	(or more)	**EFORd**	**0**	**0.115**	**0.1125**	**0.1135**	**0.1145**
2	**MWOUT**	**INDEX**	Probability	Probability	Probability	Probability	Probability
3	0	0	1.000	1.000	1.000	1.000	1.000
4	0	0	1.000	1.000	1.000	1.000	1.000
5	0	0	1.000	1.000	1.000	1.000	1.000
6	**0**	**0**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
7	10	1	0	1.150E-01	2.146E-01	3.037E-01	3.834E-01
8	20	2	0	0.000E+00	1.125E-01	2.132E-01	3.033E-01
9	30	3	0	0.000E+00	1.294E-02	1.250E-01	2.252E-01
10	40	4	0	0.000E+00	0.000E+00	2.435E-02	1.361E-01
11	50	5	0	0.000E+00	0.000E+00	1.277E-02	4.608E-02
12	60	6	0	0.000E+00	0.000E+00	1.468E-03	2.572E-02
13	70	7	0	0.000E+00	0.000E+00	0.000E+00	1.431E-02
14	80	8	0	0.000E+00	0.000E+00	0.000E+00	2.788E-03
15	90	9	0	0.000E+00	0.000E+00	0.000E+00	1.462E-03
16	100	10	0	0.000E+00	0.000E+00	0.000E+00	1.681E-04

In words, the spreadsheet equations for a given cell (E7 to H16) are: For the given level of capacity out (rows 7 to 16) shown in column B, a summation of

- the previous unit(s) probability on forced outage (Same Row) times the given unit's availability plus
- the given unit's probability of being forced out times previous unit(s) probability of being forced out.

Example spreadsheet calculations are:
Cell F7 = E7*(1-F1)+E5*F1
Cell F8 = E8*(1-F1)+E6*F1
Cell F9 = E9*(1-F1)+E7*F1

The EFORd statistic represents the forced outage rate for an individual unit. A unit's availability is 1 (or 100%) minus the unit's forced outage rate. In a two state model, a unit forced out or available, all probabilities need to sum to 100%.
The Table step size is equal to 10, i.e.the values in the column B are in increments of 10.
From these clear, concise and defined calculations the SAS code could be developed to replicate the results for each individual cell in the matrix.