EPRI

Asset Performance Database

Industry Database Design for Transmission Cables and Components

Technical Report

Asset Performance Database

Industry Database Design for Cables and Joints

1002133

Final Report, December 2003

EPRI Project Manager
P. Vujovic

EPRI • 3412 Hillview Avenue, Palo Alto, California 94304 • PO Box 10412, Palo Alto, California 94303 • USA
800.313.3774 • 650.855.2121 • askepri@epri.com • www.epri.com

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

ORGANIZATION(S) THAT PREPARED THIS DOCUMENT

Automation Technology, Inc.

ORDERING INFORMATION

Requests for copies of this report should be directed to EPRI Orders and Conferences, 1355 Willow Way, Suite 278, Concord, CA 94520, (800) 313-3774, press 2 or internally x5379, (925) 609-9169, (925) 609-1310 (fax).

CITATIONS

This report was prepared by

Automation Technology, Inc.
2001 Gateway Place
Suite 100
San Jose, California 95110

Principal Investigator
R. Ghosh

This report describes research sponsored by EPRI.

The report is a corporate document that should be cited in the literature in the following manner:

Asset Performance Database: Industry Database Design for Cables and Joints, EPRI, Palo Alto, CA: 2003. 1002133.

REPORT SUMMARY

EPRI's Asset Performance Database (APD) models cables, joints, and terminations in a central Common Information Model (CIM) format. The database specification includes physical asset information, historical operating information, and maintenance activity for use in failure and component analysis. The specification also allows the database to be used in maintenance and asset management optimization—including strategies for replacement, investment in monitoring systems, and testing to identify design and material problems at early stages.

Background

An abstract model, the CIM represents all the major objects in an electric utility enterprise typically involved with utility operations. It provides a standard way to represent power system resources as object classes, interrelationships, and attributes. By doing so, the CIM provides the technological basis for enabling open information exchange between market applications. For example, it allows the integration of Energy Management System (EMS) applications developed independently by different vendors or integration of an EMS and other systems, such as generation and distribution systems. CIM application to the APD will allow utilities to 1) identify which data to collect, 2) collect data in accepted, standardized industry formats, and 3) organize data according to industrywide, standardized data objects and attributes—all of which facilitates data sharing and analysis both within a company and across the industry.

Objectives

To provide a functioning industry database for cables, joints, and terminations for maintenance and failure analysis purposes, particularly maintenance and asset management optimization based on company and industrywide equipment performance analysis.

Approach

Working in collaboration with utilities, the project team developed a CIM database specification for high-voltage power cables. The specification was of sufficient quality for designers to create a functioning database using utility client data in the areas of physical asset information, historical operating information, and maintenance activity. They used Automation Technology's Aware platform as the database application for the APD. The Aware platform is a commercial off-the-shelf product and is designed based on object classes and attributes. This natural fit with the CIM concept allowed CIM definitions for cable and joint assets and failures to be configured in the application without project resources being spent on software development. Moreover, Aware is web-enabled and supports portal configuration for data analysis. This allowed the long-term "Database and Knowledge" portal vision to be implemented immediately. Finally, the team used EPRI's Maintenance Management Workstation (MMW) for data mining, data analysis, visualization, and reporting. MMW also has the ability to build and maintain a digital dashboard

for use in managing key performance indicators and metrics. Integration of Aware and the MMW data mining and reporting capabilities provided the complete performance visualization environment.

Results

The CIM approach encompasses an exhaustive definition of failure cause, root cause, failure modes, failure analyses, and other cable attributes relevant for maintenance optimization and asset management. The resulting APD allows utilities to document and track all cable, joint, and termination failures and permits update of utility-specific cable failure information. The database also permits utilities to analyze failures using the powerful portals feature, where they can chart and trend failure count and rates by failure type, vintage, manufacturer, or any other criteria needed. Finally, the database facilitates the creation of instant reports with a click of a button. Industrywide, this database is unmatched in its collection of failure data coupled with population, maintenance, and operational data. The APD's developmental basis in CIM and reliability centered maintenance (RCM) concepts is similarly unique, including for example, extension of standards and standardized naming conventions to cover maintenance and asset management data objects and attributes.

EPRI Perspective

This phase of the Asset Performance Database project focused on providing a functioning industry database for cables and joints, based on CIM and RCM concepts. The APD offers the perfect combination of physical asset information, historical operating information, and maintenance activity to enable users to analyze equipment performance and the root cause of cable failure. Thanks to the integration of Aware with the MMW in the CIM format, users can analyze utility-specific data without any additional software development. In addition, they can use predeveloped templates or create utility-specific templates applying visualization tools that permit analysis on a population of data to identify underlying trends, patterns, and statistics. EPRI plans further APD developments to extend the database to include transformers, breakers, overhead lines, and other equipment.

Keywords

Asset Performance Database
Cables
Joints
Terminations
Common Information Model
Cable Failure

ABSTRACT

EPRI's Asset Performance Database (APD) models cables, joints, and terminations using the Common Information Model (CIM) format. The CIM approach encompasses an exhaustive definition of failure cause, root cause, failure modes, failure analyses, and other cable attributes relevant for maintenance optimization and asset management.

The database specification includes physical asset information, historical operating information, and maintenance activity for use in failure and component analysis. The specification also allows the database to be used in maintenance and asset management optimization—including strategies for replacement, investment in monitoring systems, and testing to identify design and material problems at early stages.

The resulting APD allows utilities to document and track all cable, joint, and termination failures. It permits update of utility-specific cable failure and performance information, allows data sharing with other utilities, and facilitates analysis on industrywide data collected among participating utilities. The database also permits utilities to analyze failures using the powerful portals feature, where they can chart and trend failure count and rates by failure type, vintage, manufacturer, or any other criteria needed. Finally, the database facilitates the creation of instant reports with a click of a button.

CONTENTS

***1* SUMMARY 1-1**

Objectives 1-1

Approach 1-1

Architecture 1-2

***2* CABLE AND JOINT MODELING IN AWARE 2-1**

The Aware Platform 2-1

Cable and Joint Configuration in Aware 2-2

Class Definitions 2-3

Cables 2-4

Joints 2-5

System Organization 2-6

System Hierarchy 2-6

Inspection Types 2-7

Cable and Joint Failure 2-9

Failure Event 2-10

Operational History 2-10

Maintenance History 2-10

***3* MMW INTERFACE 3-1**

Aware XDAM ODBC Interface 3-1

External Data Access 3-2

Aware Connection 3-2

Schema 3-2

Query Execution 3-2

Limitations 3-2

Tools Used 3-3

Invoking MMW Templates from Aware 3-3
Interface Methods Used 3-3
Assumptions 3-4

***4* ANALYSIS 4-1**
Approach 4-1
By Failure Count 4-2
By Failure Rate 4-2
Options 4-3
MMW 4-3
Built-in 4-5

***5* SYSTEM REQUIREMENTS 5-1**

***6* DATA IMPORT AND UPDATES 6-1**
Audit Databases and Define Primary Keys 6-1
Define Tables and Columns to be Used 6-1
Mapping of APD Attributes 6-2
Develop and Test Tool 6-2
Application Program Interface (API) 6-2
ConnectionString 6-2
Connect 6-2
Open 6-3
GetSystemID 6-4
AddSystem 6-4
GetSystemTypeID 6-5
GetAllCharacteristicValues 6-5
SetAllCharacteristicValues 6-7
AddSystem 6-8
CreateActivityEx 6-9
GetSingleInspectionValues 6-10
SetInspectionValues 6-11
GetTableValue 6-13
SetTableValue 6-14

7 SECURITY 7-1
Group Rights 7-1
Types of Access 7-1
Permissions Hierarchy 7-2
Types of Security 7-2
System Types 7-2
System Type Attributes 7-2
Activities 7-2
Activity Attributes 7-2
Stored Queries 7-2
Stored Query Categories 7-2
Start Node 7-3

LIST OF FIGURES

Figure 1-1 Asset Performance Database Architecture 1-3
Figure 2-1 Example of Equipment Class Definition 2-3
Figure 2-2 Example of Activity Class Definition 2-4
Figure 2-3 Typical System Hierarchy 2-7
Figure 2-4 A Joint Failure Activity in Update Mode 2-8
Figure 2-5 A Joint Failure Activity in View Mode 2-8
Figure 4-1 Initial Analysis Portal Screen 4-2
Figure 4-2 MMW Used to Analyze Cable Failure – Pie Chart View 4-4
Figure 4-3 MMW Used to Analyze Cable Failure – Bar Chart View 4-4
Figure 4-4 Failure Count Analysis – by Manufacturer 4-5
Figure 4-5 Failure Count Analysis – by Cause 4-6
Figure 4-6 Failure Count Analysis – by Year 4-6
Figure 4-7 Failure Count Analysis – by Years in Service Before Failure 4-7
Figure 4-8 Failure Rate Analysis – by Year 4-7

LIST OF TABLES

Table 5-1 System Requirement Table 5-1

1
SUMMARY

Objectives

- Model your entire cable, joint and termination asset in a central Common Information Model (CIM) compliant database. The Asset Performance Database (APD) currently models cables, joints and terminations. In the future this will be extended to include overhead lines, transformers, breakers, etc.
- Document and track all your cable, joint and termination failures. The CIM definitions encompass an exhaustive definition of failure cause, root cause, failure modes, failure analyses, etc.
- Analyze failures using the powerful portals feature where you can chart and trend failure count and rates by failure type, vintage, manufacturer or any other criteria that you may need. Create instant reports with a click of a button.

Note: CIM is an abstract model that represents all the major objects in an electric utility enterprise typically involved with utility operations. By providing a standard way to represent power system resources as object classes and attributes, along with their relationship, the CIM facilitates the integration of Energy Management System (EMS) applications developed independently by different vendors, between entire EMS systems developed independently, or between an EMS system and other systems concerned with different aspects of power system operations, such as generation and distribution systems.

Approach

This phase of the Asset Performance Database project focused on providing a functioning Industry Database for cables and joints. Key aspects of the project were:

- Working in collaboration with utilities, a CIM database specification for High Voltage Power Cables was developed. The database specification was of sufficient quality for database designers to create a functioning database using utility client data.

  The database specification includes:

  - Physical Asset Information
    - Manufacturer
    - Design Characteristics
    - Type/Model

  - Year of Manufacturer
  - Application
  - Technical characteristics
  - Etc.
- Historical Operating Information
  - Operating conditions
  - Failure Modes
  - Failure Causes
  - Failure Dates
  - Trouble Events
  - Trouble Dates
  - Etc.
- Maintenance Activity
  - Type
  - Interval
  - Findings

- Use of Automation Technology, Inc.'s (ATI) Aware platform as the database application for the Asset Performance database. The Aware platform is a commercially "Of the shelf" product and is designed based on object classes and attributes. This natural fit with the CIM concept allowed the CIM definitions for cable and joint assets and failures to be configured in the application without project resources being spent on software development. Moreover, Aware is web-enabled and supports portal configuration for data analysis. This allowed the long-term "Database and Knowledge" portal vision to be implemented today.
- Use of the EPRI Maintenance Management Workstation (MMW) for data mining, data analysis, visualization and reporting. MMW also has the ability to build and maintain a digital dashboard for use in managing KPI and performance metrics. Users can create templates using visualization tools, without software development, that enables the specialized exploratory data analysis and visualization needed when analyzing population of data to identify underlying patterns and groupings.

Aware was integrated with MMW data mining and reporting capabilities to provide the complete performance visualization environment. Cable population and failure data from two utilities will be populated.

Architecture

The figure below shows the architecture for implementing the Asset Performance Database. This will allow expanding on the model to include overhead lines, transformers, breakers, etc. at a later date.

Note that by implementing a GDA interface for Aware, all applications will be independent of how the data management system is implemented and thus can be replaced with others systems in the future. The purple components represent features currently supported by Aware.

The database can be accessed from the Internet and users will be able to add, update and analyze cable failures.

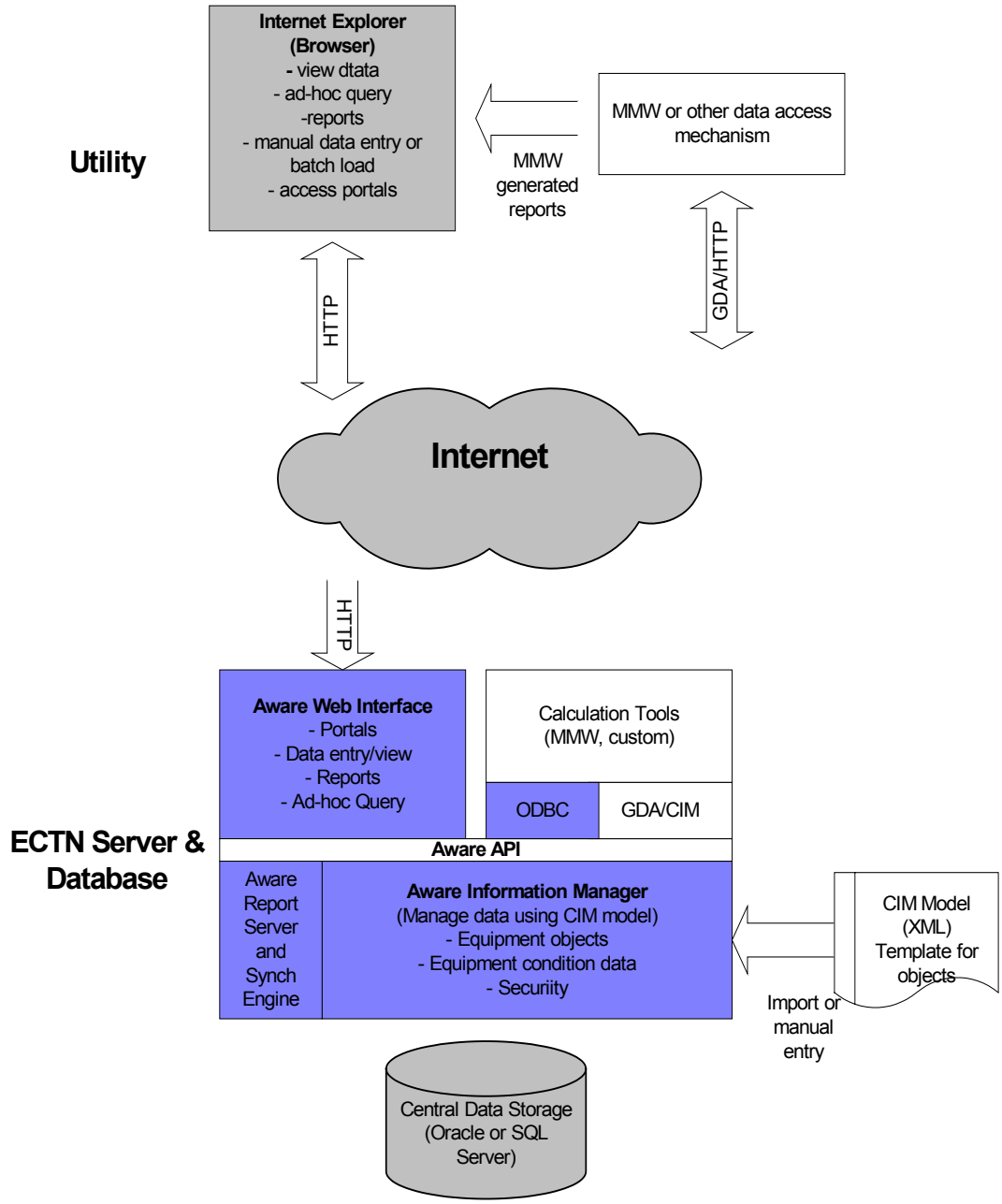

Figure 1-1
Asset Performance Database Architecture

2
CABLE AND JOINT MODELING IN AWARE

The Aware Platform

Aware is a commercially available off-the-shelf product, developed by Automation Technology, Inc. that has been used to configure the Asset Performance Database. The software has the following characteristics that have helped configure the CIM model definitions for cables and import utility information into the database.

- Aware enables you to instantiate a CIM-compliant database based on the CIM model stored in the Rational Rose environment.
- Provides all the user tools needed to manage the data including security, manual data input, import from external sources, reporting, etc.
- Automatically creates & manages the database schema based on the CIM model as this model evolves.
- Has an easy to use interface for users to build a web-based application, like the "Failure and Trouble Reporting Systems"
- Has a well established, documented, and tested API to easily interface with over modules used in this project (MMW, PRO, etc.)
- Has been installed around the world as a platform and repository for power plant equipment.

Aware is designed as a platform to manage equipment and equipment related data using objects that are user configurable based on the customer's adopted modeling standards. It can host objects of any type – including the ones defined in the Common Information Model. The view to users and to external application modules is that of objects. The application is configured by defining classes that represent the various equipment types and attributes that define the various data collected on these equipment. These classes are then used as templates for creating instances of these objects.

The concepts behind building a model in ***Aware*** are as follows:

- Classes – the different equipment types (cables, joints, transformers, etc.). Each class consists of attributes.
- Documents - activities that can be performed on systems (Cable Failure, Joint Failure)
- Inheritance / Association - for classes, defining new class based on another similar class
- Instances - specific equipment (e.g. a specific feeder FDR001 in a network) of a defined type. Each instance is defined in the three dimensions defined above.

Cable and Joint Configuration in Aware

There are four dimensions to the Aware configuration that together define the model the Aware application will use for information management. The four dimensions are:

System Types (Classes) - objects that will be part of the application domain are categorized into various system types. For example, in the Asset Performance Database application, we would have system types such as cables, joints, and terminations, etc. In simple layman terms, a system type is a definition of a type of equipment. In Aware, a system type can have a unique set of attributes. These attributes can take on the following forms:

- Reference attribute which is a static value or external association that is the same for all instances of the system type (maybe a common replacement procedure that you have defined for cables)
- Characteristic attribute which is a static value or external association that is unique for each instance of the system type (maybe the specific material and design cable insulation thickness for each unique cable)
- Inspection attribute which is a dynamic value or external association that will change overtime anytime an activity (any inspection, test, or survey) is performed on that instance of the system type (maybe a cable failure).

Inspection Types - all the different types of activities that may be performed on the different system types. Each inspection type can have its own set of attributes. A given system type can have associated with it any number of inspection types and this controls what kind of data can be collected for that type of system. For example, you may have defined that Maintenance History and Operation History inspection types (each having its own set of attributes that are recorded when performing that activity) can be performed on system type Cables. This means that an end user can store and manage Maintenance and Operational inspection data on any cable in the system.

System Organization - the rules that govern how all the different system types in your application should physically exist. These rules will be used when the user defines the actual instance of how the equipment is physically laid out in your plant. For example, a cable can appear as part of a feeder, but it would not appear as part of a joint.

System Hierarchy - the actual representation of cables and joints in your system. This is built by creating “instances” of system types in the system organization defined above.

These four dimensions provide the flexibility to configure Aware for different applications and then to easily configure these applications at different sites where the physical system is different. For example, in this application we have defined the System Types, Inspection Types, and the System Organization to allow modeling of the each utility cable and joint population. Each utility hierarchy can be different (network vs. non-network) and having a different System Hierarchy for each of the utility but the underlying definitions being the same easily accommodates that.

Class Definitions

Aware has been configured with the classes (system types and activity types) for all cable and joint related population and failures. This involved configuring the classes and attributes as defined by Maintenance & Test Engineering Company and Xtensible Solutions.

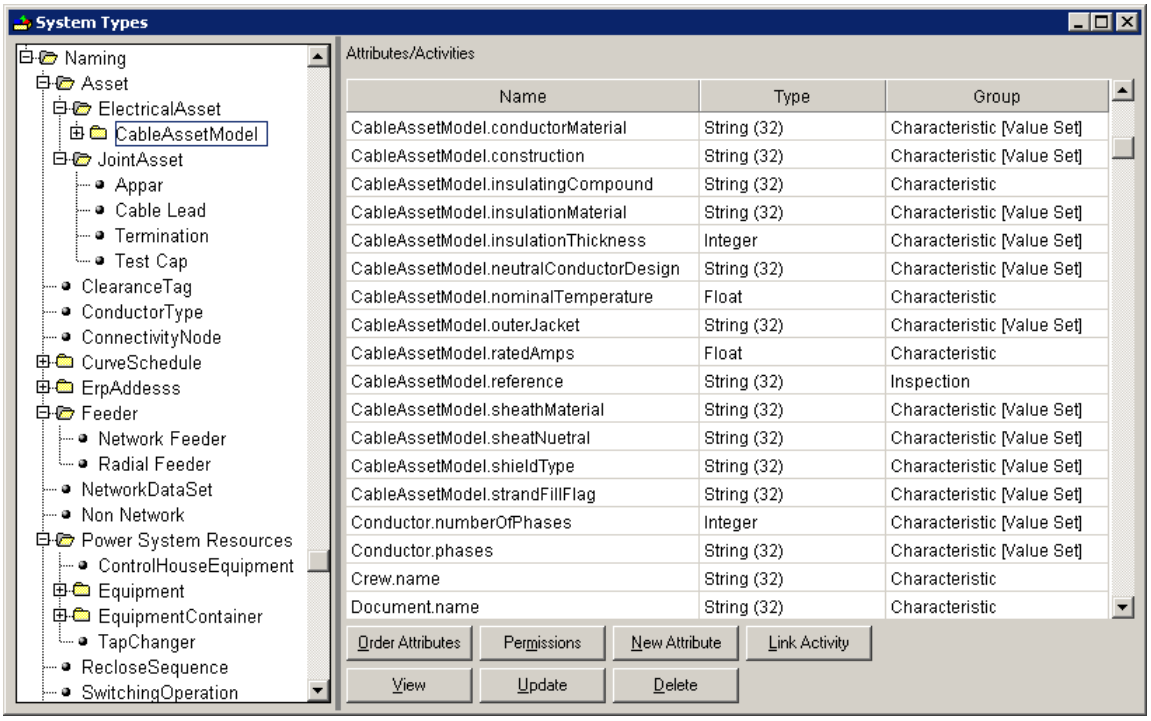

Figure 2-1
Example of Equipment Class Definition

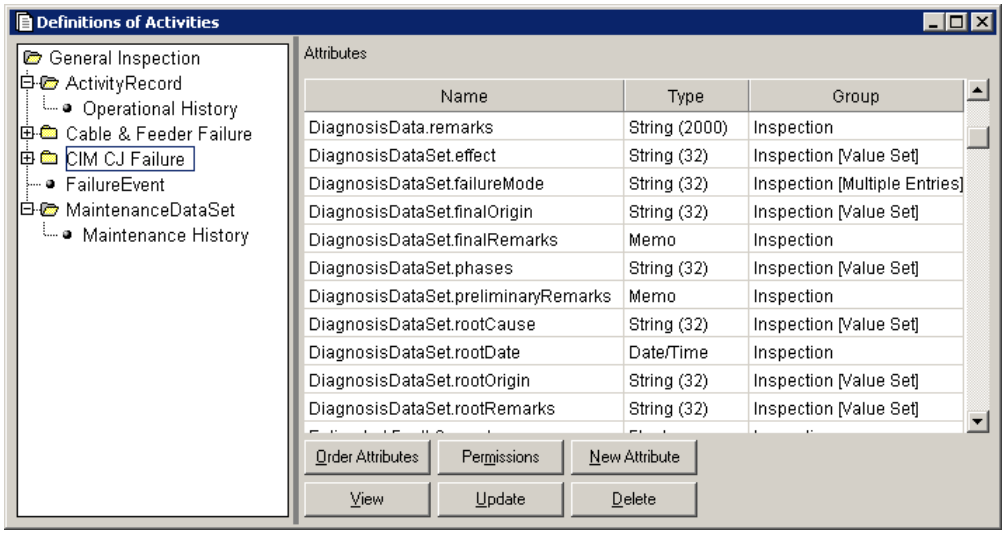

Figure 2-2
Example of Activity Class Definition

The following attributes have been configured for cables and joints:

Cables

Assets

- Nominal Voltage Rating
- Design Temperature
- Nominal Design Ampacity
- Conductor Per Cable-Phases
- Conductor Phases
- Conductor Size
- Conductor Material
- Conductor Strands
- Strand Fill
- Type of Stranding
- Conductor Shield
- Insulation Material
- Insulating Compound
- Insulation Thickness
- Sheath used as Neutral

- Sheath Material
- Jacket Type

Inventory

- Location GPS: this consists of the X and Y GPS location and the location type
- Location (non-GPS): this is a utility specific location definition for a cable and consists of X and Y locations and the location type
- Address
- Length
- Status
- Installation Type
- Manufacturer
- Manufacture Date
- Lot Number
- Installation Date
- Crew Name
- Installation Conditions
- Description
- Specification ID

Joints

- Cable IDs (1 – 5): This defines the cables that are connected for the joint
- Location (GPS)
- Location (non-GPS)
- Joint Type
- Joint Configuration
- Joint Fill
- Installation Crew
- Installation Date
- Installation Conditions
- Manufacturer
- Manufacture Date

- Model Number
- Lot Number
- Nominal Voltage Rating
- Description
- Address
- Status

System Organization

The System Organization lets you set the rules governing how all the different system types in your application should physically exist. For example, you can set the rule that a equipment of type Cable can only be created under a system of type Feeder. Consequently, when you are in the configuration mode you will not be able to put a cable under the network system. In addition to controlling how a system can be modeled, system organization also limits the list of system types one has to choose from at any level. When you go to add a system, your selection is limited to the system types allowed by the system organization.

This is available from the Windows version of Aware.

System Hierarchy

System hierarchy configuration; i.e. system building, is performed after the types of systems and the rules for assembling these systems into a hierarchy have been defined through the System Types and System Organization configuration. This is the final configuration step of creating a model that resembles the physical nature of your application domain. An example system hierarchy for a power system resource is shown below. This example exposes a power system model in terms of the various physical components that are part of it and the relationships between these parts. This is the model the users are used to thinking in terms of when putting together a report of inspections or failures in a power system. In the following figure, a file folder indicates more items beneath it and a filled in circle at the end of a “branch.” The “00947-01396” cable segment is the end of the branch and its parent is “Cables.”

System building capability is available over the web.

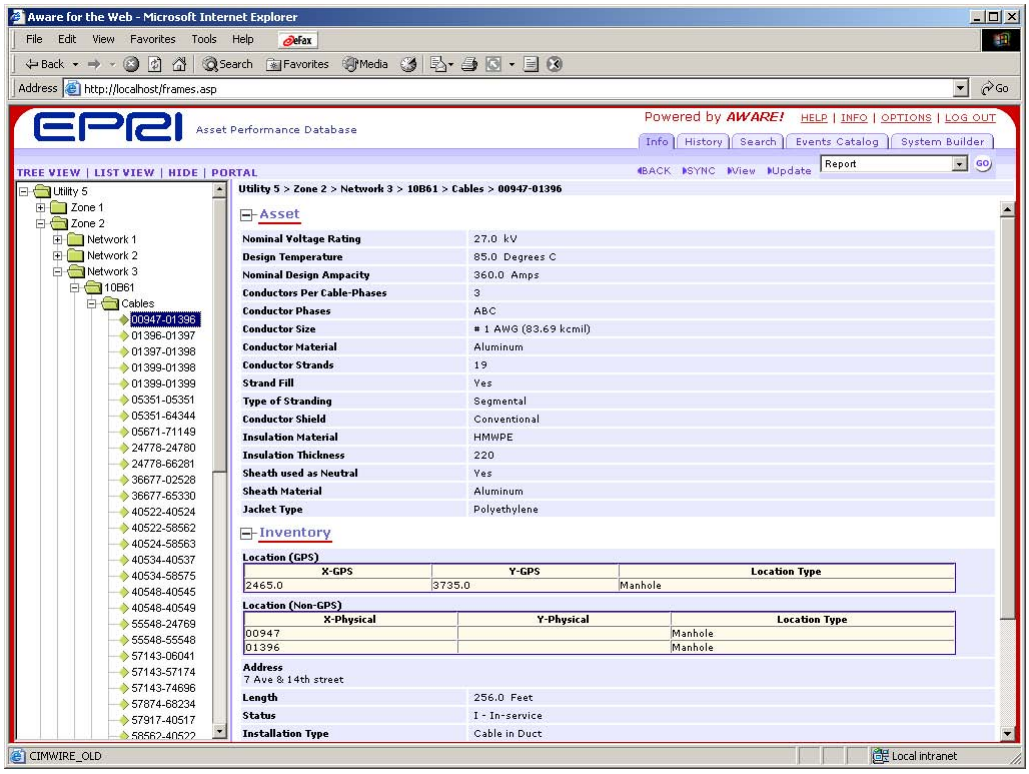

Figure 2-3
Typical System Hierarchy

Inspection Types

The following activities have been configured in the database:

- Failure event
- Cable and Joint (Splice) failure. This also includes documenting failures for other accessories like terminations, cable caps, etc.
- Operational history
- Maintenance history

The following figures show sample screens of a joint failure in update and view modes.

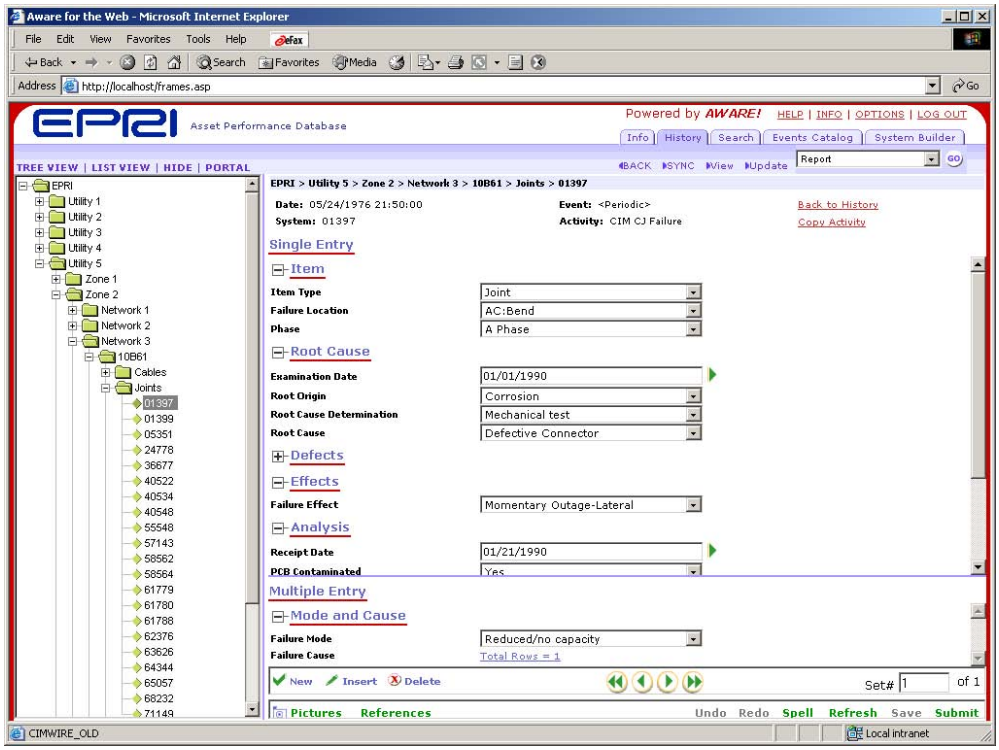

Figure 2-4
A Joint Failure Activity in Update Mode

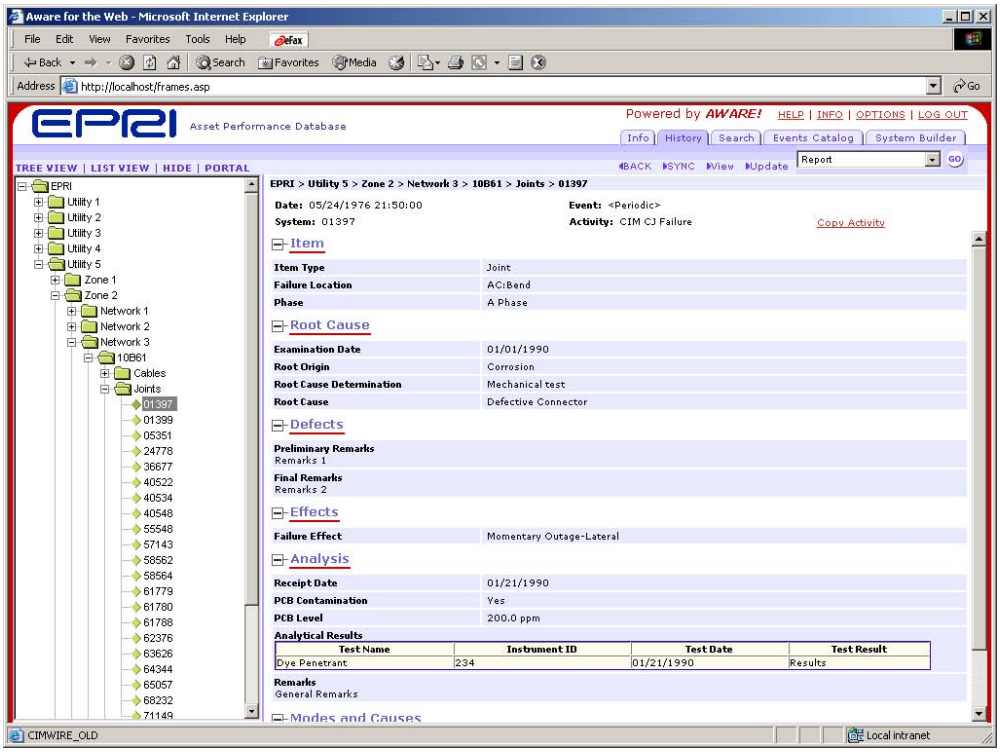

Figure 2-5
A Joint Failure Activity in View Mode

The following attributes have been configured for cables and joints activities:

Cable and Joint Failure

- Item
  - Item Type
  - Failure Location
  - Phase
- Root Cause
  - Examination Date
  - Root Origin
  - Root Cause Determination Method
  - Root Cause
- Defects
  - Preliminary Remarks
  - Final Remarks
- Effects
  - Failure Effect
- Analysis
  - Receipt Date
  - PCB Contamination
  - PCB Level and Units
  - Analytical Results consisting of
    - Test Name
    - Instrument ID
    - Test Date
    - Test Result
- Remarks
- Modes and Causes
  - Failure Mode
  - Failure Cause consisting of:
    - Examiner Name (Last, middle and first names)
    - Examination Date

  - Type
  - Origin
  - Cause

This model allows documenting:

- Multiple analytical tests and results per failure
- Multiple failure modes per failure
- Multiple failure cause per mode

Failure Event

- Event Type
- Fault Locating Method
- Fault Location
- Event Details / Remarks

Operational History

- Type
- Duration
- Remarks

Maintenance History

- Type
- Test Frequency
- VLF Voltage
- Results / Reports

3
MMW INTERFACE

Aware has been integrated with the EPRI Maintenance Management Workstation (MMW) to provide a complete performance analysis and visualization environment. The interface consists of two components:

- Aware XDAM ODBC driver that allows MMW to query the Asset Performance Database
- MMW automation server methods that allow invoking MWM templates from Aware and creating web files using web publishing. The output results in html in then displayed within the Aware portal frame.

Aware XDAM ODBC Interface

MMW uses ODBC to access data from various data sources. The Aware External Data Access module (XDAM) module is utilized to access cable failure data from Asset Performance Database. All queries configured within the database are exposed as tables that are accessed using ODBC from MMW. Queries can be configured within Aware using the web interface.

The External Data Access module for Aware allows the use of database connectivity standards that include ODBC, JDBC, OLE DB, and .NET to allow various applications to interact with Aware using SQL. Examples of applications that use ODBC are MMW, Microsoft Excel, Crystal Reports, Microsoft Access, and hundreds more. JDBC is primarily used by Java based applications that include Web Servers and Applications servers such as IBM webSphere and BEA WebLogic.

Aware is inherently an Object-Oriented platform with data stored in objects of various types. SQL on the other hand is geared towards tabular view of data. The mapping of object view to tabular view is accomplished through the use of Aware's Query engine. Aware's Query engine allows the definition and execution of queries to extract equipment and activity data based on searching the entire database or localizing it to a sub-set of equipment using the equipment hierarchy.

Queries in aware are defined by going through the search tab. Through the search GUI, the user can build a query that can include attribute from the equipment and attributes from activities. The tables exposed in the SQL view of Aware data contain the set of fields that are defined for display in the query builder. Any query added to Aware is immediately accessible as a TABLE on which SQL can be applied.

Support for SQL in the External Data Access module empowers the end user to perform any operation supported by SQL on the data set returned by the query the table is mapped to. This includes filtering based on WHERE clause, grouping, ordering, and aggregating.

External Data Access

Client/Server version of OpenAccess SDK 5.x is used to implement a OpenRDA Server for Aware. The OpenRDA Server will make use of EasyCOM for all data access. The glue between the OpenRDA Server and Aware is referred to as the Aware IP and is coded in C++.

The Aware IP performs the following functions:

- Establish connection with Aware
- Expose the schema
- Execute the associated query and retrieve the data

Aware Connection

A connection request from a client will result in a connection to Aware using the same user name and password passed in from the client.

Schema

The schema exposed to the consumer application is the set of all queries accessible to the logged in user. Each query is exposed as a Table. The name of the table is the same as that of the query. The SCHEMA of each table corresponds to the attributes returned in the query.

The list of tables is cached during connection in order to speed up the access to each table. Columns for each table will consist of the set of output and parameter columns defined for the query and columns such as the EQUIP_ID, FORM_ID, Date, and other system fields. The name of the column is the alias name specified in the query.

Indexes and foreign key relationships are not exposed.

Query Execution

Execution of the query involves extracting the parameter values from the WHERE condition in the SQL query and passing them to the query execution engine.

Limitations

- Only data exposed through queries is accessible.
- Queries must include the prompt columns as part of the output columns in order for the user to get unambiguous result sets when OR conditions are involved.

- Tables with "_" in names fail to return column information when accessed from MS Excel.

Tools Used

Some additional third party tools have been used for ODBC/SQL enabling Aware.

OpenAccess ODBC SDK 5.x – OpenAccess is the SQL middleware solution from Automation Technology, Inc. for building an ODBC, OLE DB, .NET and JDBC driver for any proprietary, legacy, ***object*** or relational database system or application. It complies with Microsoft ODBC, OLE DB, ISO RDA, ISO SQL, and ISO CLI specifications.

Invoking MMW Templates from Aware

An option to invoke MMW has been added to the Aware portal configuration. An optional tag (DSN for Data Source Name) defines the data source where the MMW templates are defined. The default value is *WS_PageBuilder*. When the user selects the MMW option from the portal page, Aware retrieves the list of templates from the *Template* table in the DSN. The Template description is displayed as a link. However, if no description is found then the template name is displayed instead.

Interface Methods Used

Clicking on a Template link invokes MMW to perform the following operations:

1. Create the Performa Automation Object.

   Set objPerforma = Server.CreateObject("Performa.Document")

   Where 'objPerforma' is the MMW Object

2. Login to MMW.

   objPerforma.Login("Aware", "aware").

   Note that the user ID and password has been pre-defined. This user needs to be configured in MMW for the interface to work.

3. Open the selected Template.

   objPerforma.OpenTemplate(sMMWTemplate)

   Where 'sMMWTemplate' is the template selected by the user from the Aware portal page to be executed by MMW

4. Run the specified Template.

   objPerforma.RunTemplate(sMMWTemplate)

5. Update the 'PlaceHolder' table in the Page Builder database.

objPerforma.UpdatePlaceHolder

6. Create the Web Publisher automation Object.

   Set objPublisher = Server.CreateObject("Puwebtpl.Application")

   Where ‘objPublisher’ is the Web Publisher Object

7. Publish the Html template and display it.

   objPublisher.PublishTemplate (sTemplate)

   where ‘sTemplate’ is the Html template file to be published.

Aware displays the HTML file generated by MMW within the portal frame.

Assumptions

The following assumptions have been made in defining this interface:

1. The DSN for the Page Builder table is WS_PageBuilder. This is installed by MMW by default during MMW installation. User can override the default from the portal configuration.
2. The templates to be displayed are configured in the 'Template' table in the Page Builder database.
3. The Html and MMW template are assumed to reside in the same directory and have the same names except their extensions. This implies that there is a one to one mapping between the Html and MMW templates.
4. The html template and the output html files have the same name.
5. The 'Control' table in the Page Builder database has absolute paths for the html template and html output file.
6. The output html file resides in the Easyweb/temp folder.
7. A default user ‘Aware’ needs to be created in MMW. Aware will use this user ID to interface with MMW.

4
ANALYSIS

One of the objectives of Asset Performance database is to allow analysis of one's failure data and to allow comparing against the industry. This is accomplished by providing a tabular and graphical representation of the data. Many of the commonly used analysis are pre-configured and the administrator can build additional ones.

Approach

Since Aware is already web-enabled, it provides the environment necessary for customers to access data. It has also been interfaced with MMW and can be invoked to produce reports of complex analysis from the analysis portals page.

Portals provide a systematic approach to presenting analysis material to the users. No single existing tool discussed provides for the collection, management, or presentation to the diverse types of knowledge that will be generated. A simple html based system will not enable the types of content management support that will be needed with the large quantities of restricted-access data expected. Combined Aware and MMW provide a complete content management solution. MMW and its query and reporting capabilities are focused on data driven reports, tables, and graphs. And the MMW dashboard features can be used to generate the hierarchical KPI and industry wide benchmarking dashboards with drill-down capabilities. However, MMW isn't designed to present the more unstructured date that will also be collected.

The Aware content management web portal platform has been used to define the analysis portals. Aware and MMW provide the tools needed to present the results from this project.

Automation Technology, Inc. interviewed utility users and industry experts to determine the types of analysis that will be required. Based on these discussions, the analysis portals defined in this section have been defined.

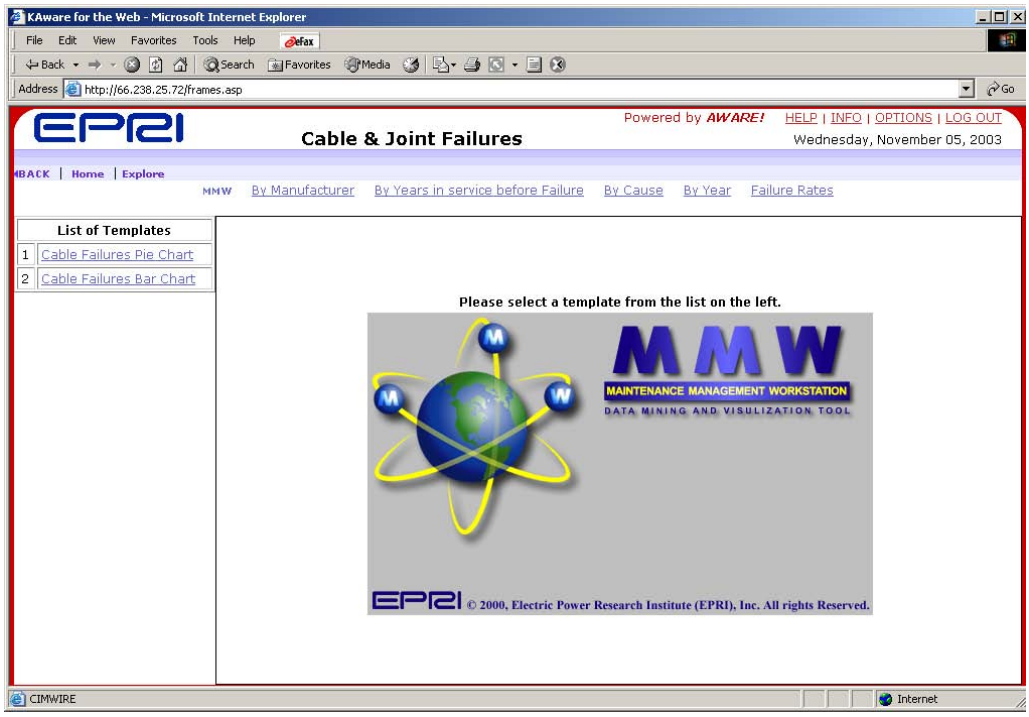

Figure 4-1
Initial Analysis Portal Screen

By Failure Count

Failure count analysis is available on a per utility basis. The user will be able to analyze failure information across the installed base. This is available for equipment that the analyzer owns.

Failure counts are useful when a utility is analyzing it's own cable failure data. Failure data can be presented in various combinations including:

- By manufacturer
- By vintage
- By type
- By years before failure
- By failure cause
- Any combination of the above

By Failure Rate

Failure rate will be calculated based on the population information. Since cable segments and joints information is populated in APD, aggregations will be done to determine the total population. Alternatively, users can enter aggregate values manually.

Failure rate analysis will be available for users to compare performance with different population within their install base or compare against the industry standards. For instance users can compare failure rates based on manufacturer or cause within their own population or across the industry. At no point will users see utility specific information for equipment not owned by them.

Failure rate calculations depend on aggregation of the population. The aggregations can vary depending on the type of analysis required. The aggregations methods that are required have not yet been finalized.

Options

Users have two options to perform their analysis:

- MMW
- Built-in

MMW

The Asset Performance Database can invoke MMW templates and present results in the portal page. MMW can be used for advanced analysis where data needs to be correlated across multiple databases. MMW can also be used to analyze APD failure data.

The MMW templates that users can select have to be configured using the Page Builder. APD queries MMW for all the templates thus configured and displays a list for the user to select from. Once the user selects a template MWM is invoked using the automation server methods and results displayed within the page. The following figures show sample MMW outputs integrated within the Aware portal screen.

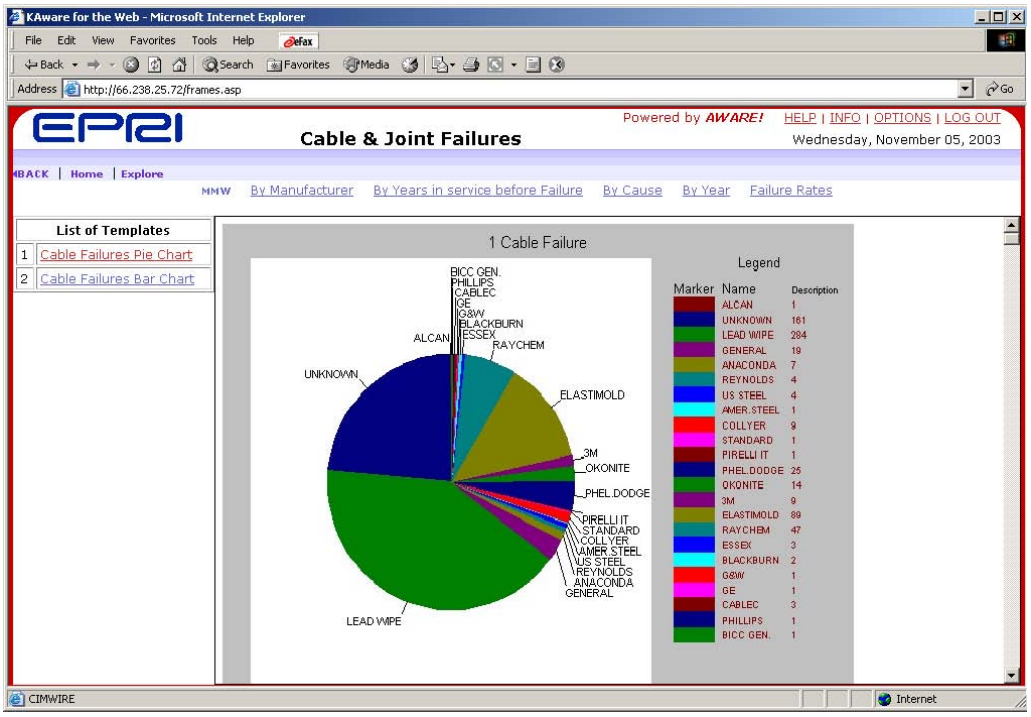

Figure 4-2
MMW Used to Analyze Cable Failure – Pie Chart View

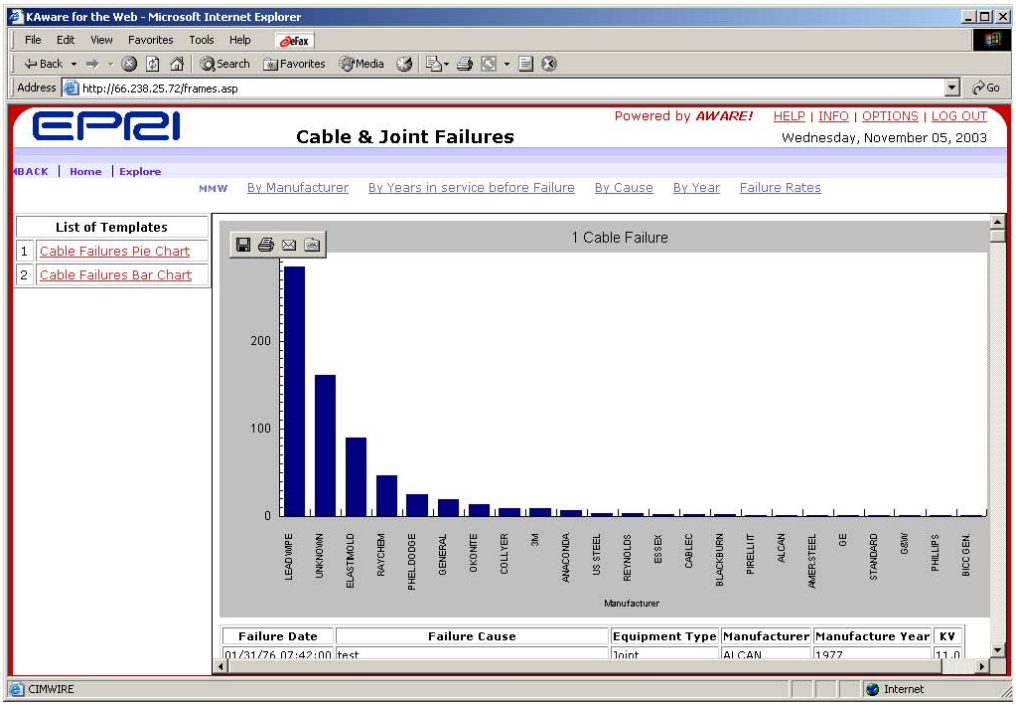

Figure 4-3
MMW Used to Analyze Cable Failure – Bar Chart View

There is no limit to the number of templates that can be configured in Page Builder. All the templates configured will be available to the users.

Built-in

Aware has a built-in feature to analyze data and display them in pre-defined graphs or in a tabular format. Aware utilizes Microsoft's Excel for doing the analysis. Aware uses the automation methods to push the search results into Excel and have it display the graphs using pivot tables. This provides the users with an interactive method of displaying various analyses with changing inputs.

Each analysis has a query associated with it. The query is built using the software. The portals are configured using XML. Users can build their own portal pages.

A few sample analysis screens are shown below.

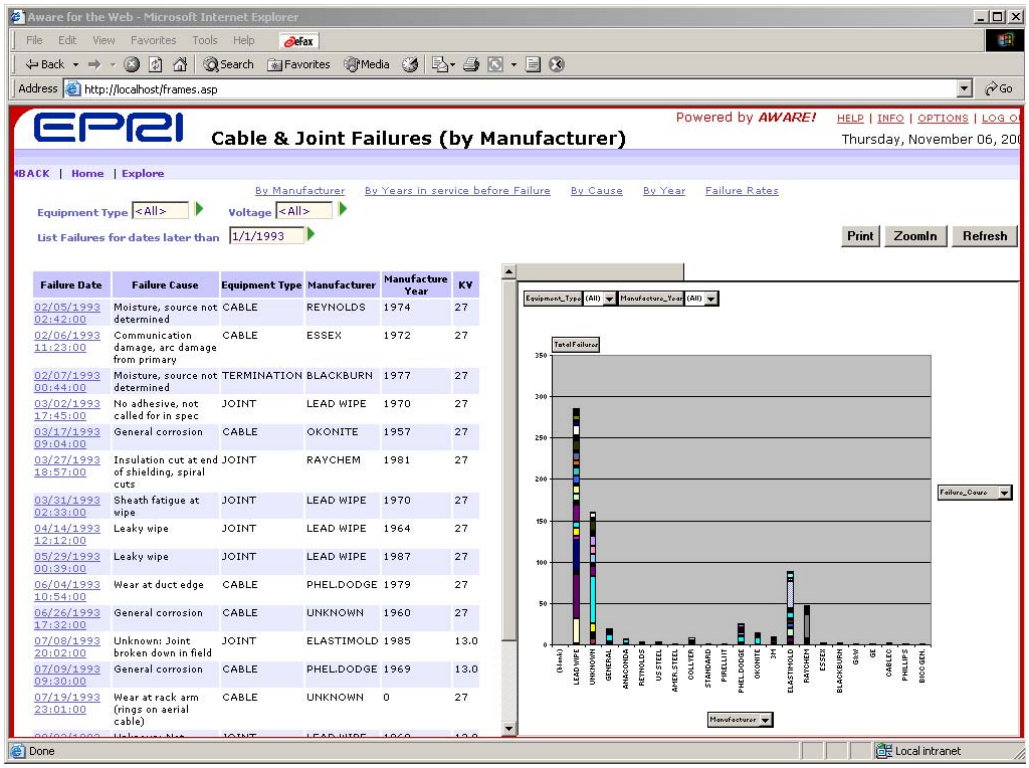

Figure 4-4
Failure Count Analysis – by Manufacturer

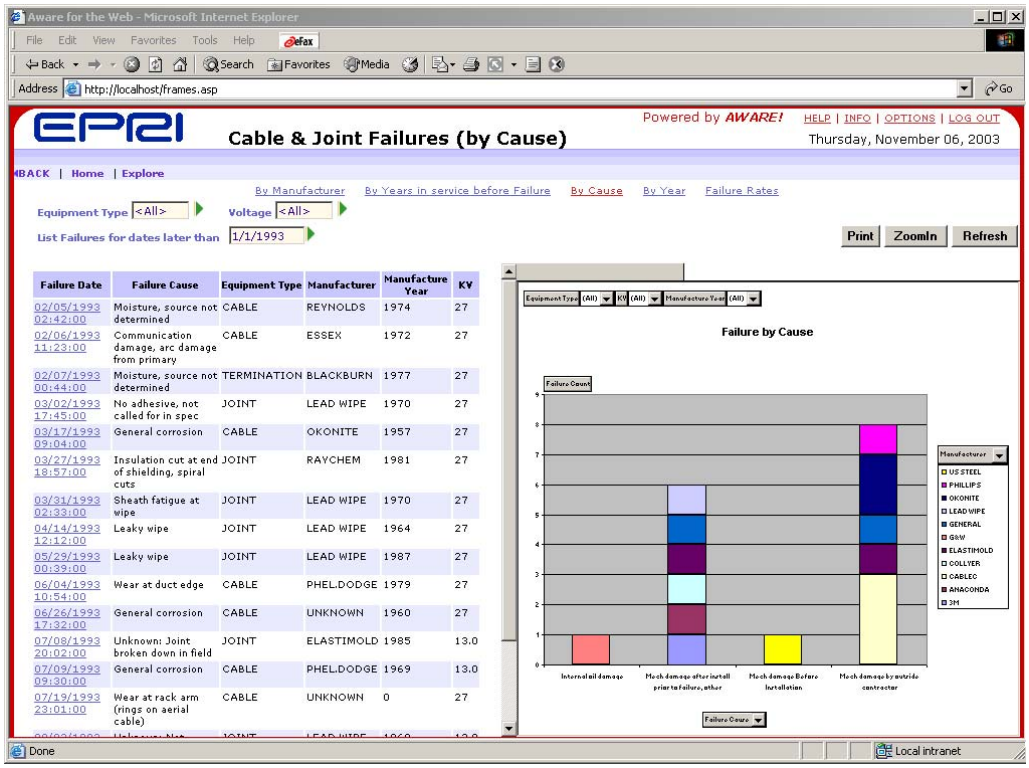

Figure 4-5
Failure Count Analysis – by Cause

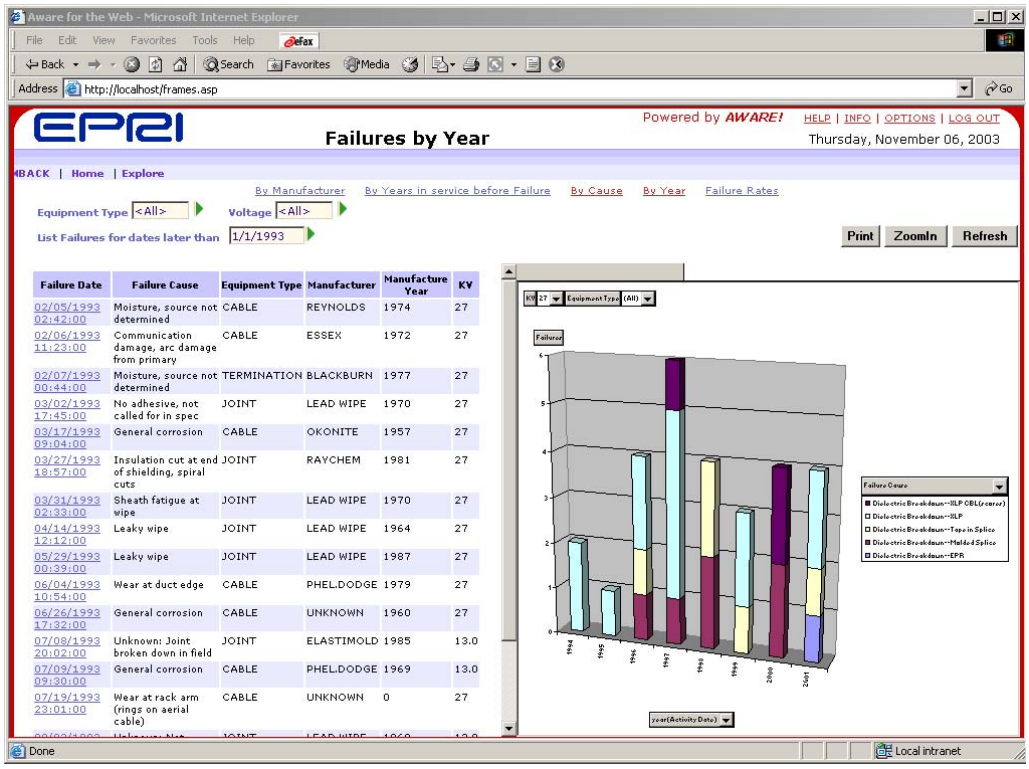

Figure 4-6
Failure Count Analysis – by Year

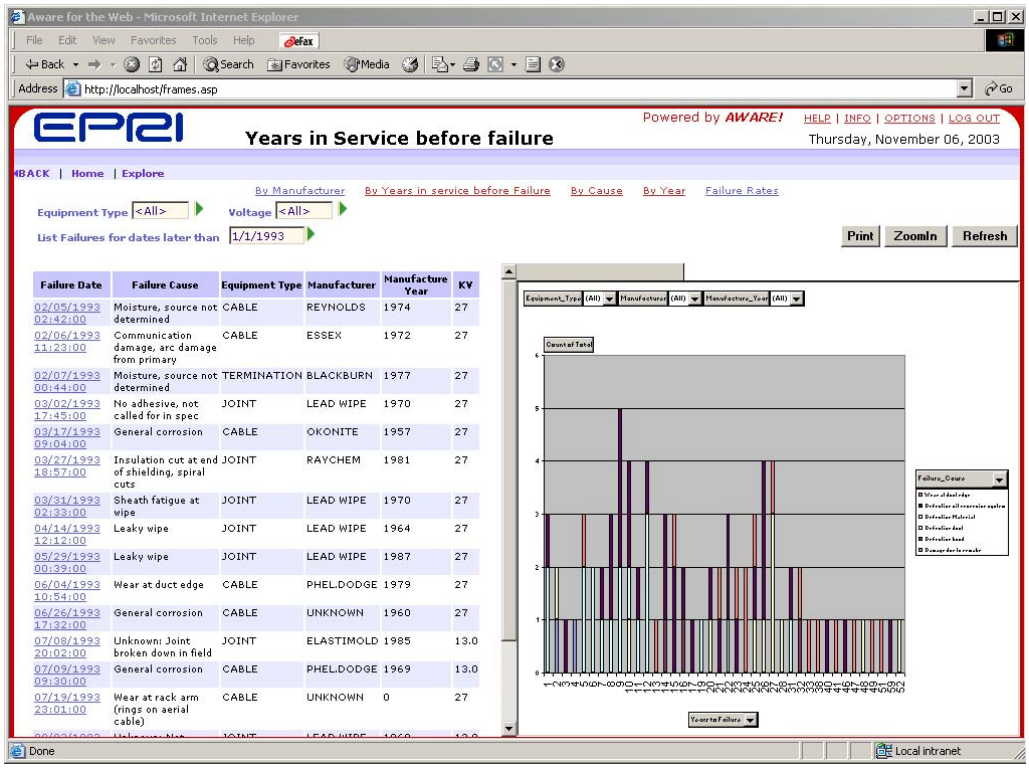

Figure 4-7
Failure Count Analysis – by Years in Service Before Failure

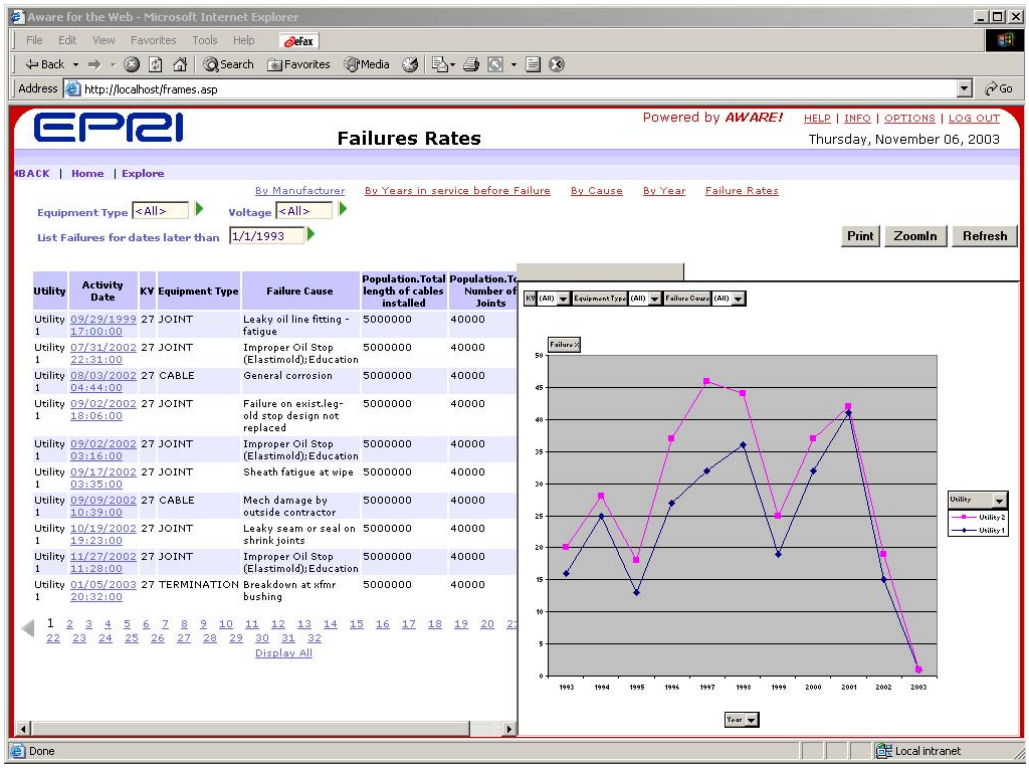

Figure 4-8
Failure Rate Analysis – by Year

5
SYSTEM REQUIREMENTS

The Asset Performance Database is a web-based application that requires a web-server and a database server. The web and database servers could run on the same hardware or on different platforms. The web-server needs to be Microsoft's Internet Information Service (IIS). The database can be either Oracle or SQL Server.

Whereas end users would require a thin client (IE 5.5 or later), the Aware desktop (windows) version of the software is required to administer the application. The administrative functions include:

- User administration: maintaining users and user groups for the application
- Configuration: maintaining the classes and attributes, if required.

The following table provides the system requirement for the different components involved.

Table 5-1
System Requirement Table

Component	Recommended Specifications	Minimum Specifications
Database Engine	Oracle 8i+ or later OR MS SQL Server 2000	Oracle 7.3+ or MS SQL Server 7
Database Server	Pentium IV 800 MHz or higher 256+ MB RAM 10+ GB Hard Drive Tape Backup	Pentium III 500 MHz or higher 128+ MB RAM 2+ GB Hard Drive

Component	Recommended Specifications	Minimum Specifications
Web Server	Pentium IV 800 MHz or higher 512+ MB RAM 8+ GB Hard Drive Windows 2000 MS IIS 5.0 or higher	Pentium III 500 MHz or higher 256+ MB RAM 1+ GB Hard Drive Windows 2000 MS IIS 5.0 or higher
Administrator Workstations (Client application is installed on workstation for administrative functions – e.g. managing user rights, etc.).	Pentium IV 600 MHz or higher 256+ MB RAM 100MB of free hard disk space VGA with 256 colors CD ROM Win 95, Win 98, NT4.0, 2000 or XP Oracle SQL*NET client MS MDAC 2.6	Pentium III 300 MHz or higher 128+ MB RAM 60MB of free hard disk space VGA with 256 colors Win 95, Win 98, NT4.0 Oracle SQL*NET client ODBC Driver for Oracle
Client Workstations	IE 5.5+ VGA with 256+ colors Win 2000 or XP	IE 5.5 VGA with 256+ colors Win 95, 98, NT4.0, 2000, or XP.

6
DATA IMPORT AND UPDATES

The Asset Performance Database is modeled to store asset and failure information from multiple utilities. The objective is to have the database populated with data from a number of utilities so as to provide good industry averages. The volume of information to be populated is prohibitive for manual entry. Automated tools / filters are required to transfer data to the APD. These data import tools have to be written specific to each utility as the data resides in different databases with different schemas.

There are two phases to populating the database:

- Phase 1: One-time import of utility specific information. Filters have to be written to import population and failure data provided by the utilities into APD. This will be a one-time effort.
- Phase 2: Automatic update of APD to import incremental changes at the utility end. New population and failure data from the utilities have to be imported into APD on a regular basis. This will require enhancing the tools developed in Phase 1 to automatic push changed data into APD.

This project addresses phase 1 only. The steps involved to import data into APD are listed below.

Audit Databases and Define Primary Keys

Cable and joint population and failure information reside in multiple databases within a utility. In many cases these are islands of information where each database fulfills a particular function and not necessarily tied in with other databases. The utility must first audit all the relevant databases within their system and evaluate what information should be used to populate APD and from which database.

In conjunction with this audit, the utility must also define the primary keys across databases, if possible. In cases where primary keys do not exits, mapping information should be provided to correlate data between the databases.

Define Tables and Columns to be Used

The databases may contain more information than what needs to be transferred into APD. The Utility must provide a list of tables and columns that contain the necessary information. The columns from the various tables must match with the attributes defined in APD.

In some cases, the utility may have to join multiple tables to provide the information in a more usable format.

Mapping of APD Attributes

The vendor has to work with the utility to ensure that the data mapping is defined correctly. A mapping table should be created that will list the APD attributes and the mapping to the database, table and column. The tool to populate the relevant population and failure information will use this mapping.

Develop and Test Tool

Aware is an object-oriented platform. Unlike a SQL application where data can be populated into tables and columns, Aware operates with objects and attributes. A rich set of Application Program Interface methods are available that can be used from any application that support COM. The developer must use the mapping table created in the previous task to populate the relevant objects and attributes. A sample program can be provided, if required.

The tool must be tested with a small set of data that would be a reflection of all different data sets. The developer must test and ensure that all necessary information is being populated into APD. A sample ADP should be used for testing. Once the program is well tested, the tool should be run against the central APD.

Application Program Interface (API)

A complete list of APIs can be got by contacting Automation Technology, Inc. EasyCOM, the COM component of Aware must be registered in order to work with it. The common APIs are listed below:

ConnectionString

Purpose: This is a property. It must be set to the string containing the connection information.

Usage: obj.ConnectionString = "<ODBC Data Source>,<User ID>,<User Password"

Connect

([out, retval] LONG *plRetCode)

Purpose: Connects to the data source specified in the property ConnectionString. It will also apply any required data updates

Parameters: None

Return Values: Error codes as follows:

0 (ECRC_SUCCESS) if successful

1 (ECRC_UNSPECIFIED) for unspecified error

20 (ECRC_INVALIDAUTHENTICATION) invalid authentication (user ID or password)

21 (ECRC_ERRORCONNECTING) Unable to connect (Invalid data source!)

22 (ECRC_INVALIDUSER) Invalid user; contact administrator

23 (ECRC_INCORRECTPASSWORD) Incorrect password

24 (ECRC_ACCOUNTDISABLED) User account is disabled;

25 (ECRC_PROMPTFORUPGRADE) Wrong version; prompt for upgrade

26 (ECRC_WRONGVERSION) Wrong database version

29 (ERCR_CANNOTUPDATE) the database needs to be updated

30 (ECRC_NOAPIAUTHORIZATION) not authorized to use API.

47 (ECRC_NOSYSTEMROOT) No system root defined for the user

Open

([in] BSTR sTempFilePath, [in, optional, defaultvalue(FALSE)] BOOL bReportServerMode, [out, retval] BOOL *bSuccess)

Purpose: Connects to the data source specified in the property ConnectionString if not connected by a previous call to Connect, and initializes the COM object by fetching some data objects. Attributes, System Types, Employees and Events are cached from DB.

Parameters:

sTempFilePath – path for the file download location. This must be a valid location or an empty string. If an empty string is specified, then the TEMP directory (the one setup in the environment variables) is what will be used.

bReportServerMode – An optional parameter telling COM that this is the Report Server calling it. The default value is FALSE.

Return Values: BOOL – TRUE if successful, FALSE otherwise.

GetSystemID

([in] LONG lRootSystemID, [in] BSTR sSystemName, [out, retval] LONG *plSystemID);

Purpose: This API gets the ID of the system with name sSystemName and root system ID lSuperSystemID. It will also work if the system name passed in is the same as the root system ID, in that case it'll return that same root system ID. Sub-systems in a tree can have similar names, so this API will return the first one it finds with that name.

Parameters:

lSuperSystemID – The root system ID of the system tree to search.

sSystemName – The system name.

Return Values: A pointer to long, which is the system ID whose name is sSystemName and root ID lSuperSystemID. In case of an error, 0 is returned.

AddSystem

([in] LONG lParentEquipID, [in] BSTR sEquipName, [in] LONG lEquipClassID, [out, retval] LONG *plResultID)

Purpose: This function adds a new child system with the specified name and class type to a specified parent system. It will also create and initialize the system attributes. It returns the ID of the new system or one of several error codes.

Parameters:

lParentEquipID – the parent system ID to which the new system is to be added.

sEquipName – the name that is to be used for the new system.

lEquipClassID – the ID of the class to use for the new system.

Return Values: An integer with one of the following values:

Positive – The ID of the new system.

-1 - An unspecified error has occurred.

-2 - The name already exists as a child of the specified parent.

-3 - The operation would violate organization rules.

-4 - The parent is not valid.

-5 - The class is not valid.

-8 - A transaction could not be started.

-9 - An error occurred calculating the path and ancestors for the system.

-10 - An error occurred creating the system.

-11 - Error initializing the system attributes.

GetSystemTypeID

([in] BSTR sSystemTypeName, [out, retval] LONG *plSystemTypeID)

Purpose: Given a system type name, this function returns the system type ID (fetched from the cached list of system types). In case of error or if the system type doesn't exist a 0 is returned. The name lookup is case insensitive.

This API is part of the "LookupInterface". The interface can be accessed from the CWorkspace object as follows:

Dim Lookup As LookupInterface

Set Lookup = cworkspaceObj.LookupInterface

Result = Lookup.GetSystemTypeID ("Custom Object")

Parameters: sSystemTypeName – The system type name.

Return Values:

The system type ID is returned on success.

ECRC_INVALIDNAME (-6) is returned if the system type doesn't exist.

ECRC_UNSPECIFIED (-1) is returned for any other error.

GetAllCharacteristicValues

([in] LONG lSystemID, [out, retval] IUnknown **rsValues)

Purpose: This function will get the list of characteristic attribute and values of the system with the ID passed in. The attributes are returned ordered just like the user ordered them in Aware (desktop version).

The attributes are retrieved from the AttributeOrder table (if ordered), else from the attribute table. For the best efficiency, the user must order the attributes prior to this call. Also if new attributes were added, the user must order them again to avoid missing the new attributes added. The characteristic attributes are cached in memory for this system, so subsequent calls to this API with the same system ID will make use of the cache.

The values are returned as is for Integer, Float, Date/Time, String (32), String (255), String (2000), Memo, HTML and Link. For Query Table, Report Output, and File, the description is returned. For Picture, a blank is returned. And for Table, the string "Total Rows = x" is returned. So all the values look like what you see in the attribute-value list in Info Man. The rest

of the values for Table, Picture, Query Table, Report Output, and File can be retrieved by their specialized functions like “GetFileValue”, “GetTableValue”, etc.

The result is returned in a record set with the following columns:

The attribute ID (EZAPI_COL_ATTRID="Attr ID", long)

The attribute name (EZAPI_COL_ATTRNAME="Attr Name", char, MAX_LONGNAME_LEN+1)

The attribute type as an enum value (EZAPI_COL_ATTRTYPE="Attr Type", long)

A flag “does the attribute have a value set?” (EZAPI_COL_ATTRHASVS="Attr Has Value Set", Boolean)

The value ID: This is 0 if value doesn’t exist. (EZAPI_COL_VALUEID="Value ID", long)

The value string: (EZAPI_COL_ATTRVAL="Attr Value", char, MAX_LONGSTRING_LEN+1)

The value set #. This is always 0. (EZAPI_COL_GROUPNUM="Group Num", long)

Changed flag. This is initially FALSE. The user will change this later when preparing to write the values to DB. (RS_FLAGS="Changed_Flag", Boolean)

A flag “Show Attribute in Info Man?” (EZAPI_COL_SHOWATTR="Show Attribute", Boolean)

A flag “does the attribute have an external value source?” (EZAPI_COL_EXTERNALVALUESOURCE=" External Value Source", Boolean)

A flag “is the Attribute read only?” (EZAPI_COL_ATTRREADONLY= " Attr Read Only", Boolean)

A flag “is the Attribute required field?” (EZAPI_COL_ATTRREQUIREDFIELD= " Attr Required Field", Boolean)

A flag “does the value have a real value?” (RS_HASVALUE = "Has Value”, Boolean). For String (32), String (255), String (2000), Integer, Float, Date, and Link, the value string must be of length > 0. For File, Query Table, and Report Output, the storage type must be set to internal or external. For Pictures, at least one picture must exist. For Tables, the # of rows must be > 0.

Parameters:

lSystemID – the system ID of which we need the characteristic values.

Return Values: rsValues – The record set containing the values.

Important:

The Date/Time and Float values are returned formatted according to the attribute setting.

SetAllCharacteristicValues

([in] LONG lSystemID, [in] IUnknown *rsValues, [out, retval] BOOL *bSuccess)

Purpose: This function will save a list of characteristic values to the database. The values are passed in as a record set and only the values that are marked changed or that don't have a value ID (meaning new values) are saved to DB. So it's the user responsibility to mark the values changed if he needs them to be saved to DB.

This function will fully write values of type Integer, Float, Date/Time, String (32), String (255), String (2000), and Link. For Query Table, Report Output, Table and File it will write out NULL values (creating a NULL entry) if the value ID is 0. The actual value must be saved using the specialized APIs for Table, Picture, Query Table, Report Output, and File like "GetFileValue", "GetTableValue", etc. This is the only API that will allow you to write out NULL values (new values).

The record set passed in must have the following columns:

The attribute ID (EZAPI_COL_ATTRID="Attr ID", long)

The attribute name (EZAPI_COL_ATTRNAME="Attr Name", char, MAX_LONGNAME_LEN+1)

The attribute type as an enum value (EZAPI_COL_ATTRTYPE="Attr Type", long)

A flag "does the attribute have a value set?" (EZAPI_COL_ATTRHASVS="Attr Has Value Set", boolean)

The value ID. This is 0 if value doesn't exist. (EZAPI_COL_VALUEID="Value ID", long)

The value string. (EZAPI_COL_ATTRVAL="Attr Value", char, MAX_LONGSTRING_LEN+1)

The value set #. This is always 0. (EZAPI_COL_GROUPNUM="Group Num", long)

Changed flag. This is the changed flag set by the user determining whether to write the value to DB or not. (RS_FLAGS="Changed_Flag", boolean)

A flag "Show Attribute in Info Man?" (EZAPI_COL_SHOWATTR="Show Attribute", boolean)

A flag "does the attribute have an external value source?" (EZAPI_COL_EXTERNALVALUESOURCE=" External Value Source", boolean)

Parameters:

lSystemID – the system ID of which we need the characteristic values.

rsValues – The record set containing the values.

Return Values: BOOL – TRUE if successful, FALSE otherwise.

Important:

Use this function to write out the NULL (new) values for Table, Picture, Query Table, File and Report Output, before using the specialized functions to set their values (SetFileValue, SetTableValue, etc.).

AddSystem

([in] LONG lParentEquipID, [in] BSTR sEquipName, [in] LONG lEquipClassID, [out, retval] LONG *plResultID)

Purpose: This function adds a new child system with the specified name and class type to a specified parent system. It will also create and initialize the system attributes. It returns the ID of the new system or one of several error codes.

Parameters:

lParentEquipID – the parent system ID to which the new system is to be added.

sEquipName – the name that is to be used for the new system.

lEquipClassID – the ID of the class to use for the new system.

Return Values: An integer with one of the following values:

Positive – The ID of the new system.

-1 - An unspecified error has occurred.

-2 - The name already exists as a child of the specified parent.

-3 - The operation would violate organization rules.

-4 - The parent is not valid.

-5 - The class is not valid.

-8 - A transaction could not be started.

-9 - An error occurred calculating the path and ancestors for the system.

-10 - An error occurred creating the system.

-11 - Error initializing the system attributes.

CreateActivityEx

([in] LONG lEquipID, [in] LONG lEventID, [in] LONG lActivityType, [in] BSTR sDate, [in] LONG lEmployeeID, [in, optional, defaultvalue(FALSE)] BOOL bCreateDefaultFileValueFromTemplate, [in, optional, defaultvalue("")] BSTR sDefaultFileValues, [out, retval] LONG *plResultFormID)

Purpose: Creates an activity record and NULL values for all the single attributes and one set of multiple attributes. It will do everything that you get from creating an activity record from EasyDOC. This includes copy values from previous activity records if the option was set on the attribute.

This API can also be used to set additional file values after the activity record was created already. So it works as if the user is updating the activity record instead of creating it. So the user can call this API multiple times and can keep changing the sDefaultFileValues parameter.

Parameters:

lEquipID – the system ID where the activity record to be created.

lEventId – the event id of the activity record.

lActivityTypeID – The activity type ID of the activity record.

sEntryDate – The creation date of the activity record.

lEmployeeID – The employee ID to created the activity record.

bCreateDefaultFileValueFromTemplate – If this flag is TRUE, then any file value (as long as it doesn't have Copy From Previous Record or Set) will start with the template value as a default value. This flag is FALSE by default.

sDefaultFileValues – a comma separated string of attribute-file name pairs. It lists zero or more file names to use as default values for the specified attributes. This takes higher priority than template values as the default values. The string is in the following format: "attribname=filename,attribname2=filename2" and so on. The attribname can be skipped and the following can be specified "=filename", in this case the first file attribute will get this filename value. The file format for the file name is looked up by the extension of the file, if no file format exists for that extension, then the value is not set. The caption is left blank and the storage type will be internal. This parameter is optional; the default value is "". This file value setting is done instead of the template value setting except if the file was not found or if the file extension doesn't have an application linked to it and so on. So in that case the template value is copied instead (as long as the bCreateDefaultFileValueFromTemplate is set).

Return Values: The activity record ID that we just created/modified if successful.

GetSingleInspectionValues

([in] LONG lSystemID, [in] LONG lFormID, [in] LONG lActivityTypeID, [in] LONG lEventID, [in] BSTR sFormCreationDate, [in] BSTR sFormDataDate, [out, retval] IUnknown **rsValues)

Purpose: This function will get the list of single attribute and values of the system and the activity record information passed in. The attributes are returned ordered just like the user ordered them in EasyDOC (desktop version).

The attributes are retrieved from the AttributeOrder table (if ordered), else from the attribute table. For the best efficiency, the user must order the attributes prior to this call. Also if new attributes were added, the user must order them again to avoid missing the new attributes added. The attributes are cached in memory for this activity type and system, so subsequent calls to this API or the GetMultipleInspectionValues API with the same system ID and activity type ID will make use of the cache.

The values are returned as is for Integer, Float, Date/Time, String (32), String (255), and String (2000). For Query Table, Report Output, and File, the description is returned. For Picture, a blank is returned. And for Table, the string “Total Rows = x” is returned. So all the values look like what you see in the attribute-value list in Info Man. The rest of the values for Table, Picture, Query Table, Report Output, and File can be retrieved by their specialized functions like “GetFileValue”, “GetTableValue”, etc.

The result is returned in a record set with the following columns:

The attribute ID (EZAPI_COL_ATTRID="Attr ID", long)

The attribute name (EZAPI_COL_ATTRNAME="Attr Name", char, MAX_LONGNAME_LEN+1)

The attribute type as an enum value (EZAPI_COL_ATTRTYPE="Attr Type", long)

A flag “does the attribute have a value set?” (EZAPI_COL_ATTRHASVS="Attr Has Value Set", boolean)

The value ID. This is 0 if value doesn’t exist. (EZAPI_COL_VALUEID="Value ID", long)

The value string. (EZAPI_COL_ATTRVAL="Attr Value", char, MAX_LONGSTRING_LEN+1)

The value set #. This is always 0 for single attributes. (EZAPI_COL_GROUPNUM="Group Num", long)

Changed flag. This is initially FALSE. The user will change this later when preparing to write the values to DB. (RS_FLAGS="Changed_Flag", boolean)

A flag “Show Attribute in Info Man?” (EZAPI_COL_SHOWATTR="Show Attribute", boolean)

A flag “does the attribute have an external value source?” (EZAPI_COL_EXTERNALVALUESOURCE= "External Value Source", boolean)

A flag “is the Attribute read only?” (EZAPI_COL_ATTRREADONLY= " Attr Read Only", boolean)

A flag “is the Attribute required field?” (EZAPI_COL_ATTRREQUIREDFIELD= " Attr Required Field", boolean)

A flag “does the value have a real value?” (RS_HASVALUE = "Has Value”, boolean). For String (32), String (255), String (2000), Integer, Float, Date, and Link, the value string must be of length > 0. For File, Query Table, and Report Output, the storage type must be set to internal or external. For Pictures, at least one picture must exist. For Tables, the # of rows must be > 0.

Parameters:

lSystemID – the system ID of which we need the inspection values.

lFormID – the activity record ID of which we need the inspection values.

lActivityTypeID – The activity type ID of the activity record. If 0 is passed in, the activity type ID is fetched from the form.

lEventID – The event ID of the activity record.

sFormCreationDate – The activity record creation date. If an empty string is passed in, the activity record creation date is fetched from the form.

sFormDataDate – The activity record data date. If an empty string is passed in, the activity record data date is fetched from the form.

Return Values: rsValues – The record set containing the values.

Important:

The Date/Time and Float values are returned formatted according to the attribute setting.

SetInspectionValues

([in] LONG lSystemID, [in] LONG lFormID, [in] LONG lActivityTypeID, [in] LONG lEventID, [in] BSTR sFormCreationDate, [in] BSTR sFormDataDate, [in] IUnknown *rsValues, [out, retval] BOOL *bSuccess)

Purpose: This function will save a list of inspection (single and multiple) values to the database. The values are passed in as a record set and only the values that are marked changed or that don’t

have a value ID (meaning new values) are saved to DB. So it's the user responsibility to mark the values changed if he needs them to be saved to DB.

This function will fully write values of type Integer, Float, Date/Time, String (32), String (255), and String (2000). For Query Table, Report Output, Table and File it will write out NULL values (creating a NULL entry) if the value ID is 0. The actual value must be saved using the specialized APIs for Table, Picture, Query Table, Report Output, and File like "GetFileValue", "GetTableValue", etc. This is the only API that will allow you to write out NULL values (new values).

The record set passed in must have the following columns:

The attribute ID (EZAPI_COL_ATTRID="Attr ID", long)

The attribute name (EZAPI_COL_ATTRNAME="Attr Name", char, MAX_LONGNAME_LEN+1)

The attribute type as an enum value (EZAPI_COL_ATTRTYPE="Attr Type", long)

A flag "does the attribute have a value set?" (EZAPI_COL_ATTRHASVS="Attr Has Value Set", boolean)

The value ID. This is 0 if value doesn't exist. (EZAPI_COL_VALUEID="Value ID", long)

The value string. (EZAPI_COL_ATTRVAL="Attr Value", char, MAX_LONGSTRING_LEN+1)

The value set #. This is 0 for single attributes, or 0-indexed for multiple attributes (as in the DB). (EZAPI_COL_GROUPNUM="Group Num", long)

Changed flag. This is the changed flag set by the user determining whether to write the value to DB or not. (RS_FLAGS="Changed_Flag", boolean)

A flag "Show Attribute in Info Man?" (EZAPI_COL_SHOWATTR="Show Attribute", boolean)

A flag "does the attribute have an external value source?" (EZAPI_COL_EXTERNALVALUESOURCE=" External Value Source", boolean)

Parameters:

lSystemID – the system ID of which we need the inspection values.

lFormID – the activity record ID of which we need the inspection values.

lActivityTypeID – The activity type ID of the activity record. If 0 is passed in, the activity type ID is fetched from the form.

lEventID – The event ID of the activity record.

sFormCreationDate – The activity record creation date. If an empty string is passed in, the activity record creation date is fetched from the form.

sFormDataDate – The activity record data date. If an empty string is passed in, the activity record data date is fetched from the form.

rsValues – The record set containing the values.

Return Values: BOOL – TRUE if successful, FALSE otherwise.

Important:

Use this function to write out the NULL (new) values for Table, Picture, Query Table, File and Report Output, before using the specialized functions to set their values (SetFileValue, SetTableValue, etc.).

GetTableValue

([in] LONG lValueID, [out, retval] IUnknown **rs)

Purpose: Given a collection value ID, this function retrieves the collection value and places it in the record set rs. The record set will be NULL/Nothing if an error occurs. An empty collection value is returned as an empty record set (but the collection definition would still be in there)

The record set has three extra fields/columns at the beginning:

The first one is called “Row_Num” which is the original row number (1-indexed).

The second one is called “New_Row_Num”, which is the new row num if the user needs to modify this record set later. Initially this column has the exact values as the previous column (1-indexed as well).

The third one is called “Changed_Flag”, which is a decimal value representing which cells has changed in the row. Initially the values of this field are all set to 0. The value is the addition of (1 * (1 or 0) + 2 * (1 or 0) + 4 * (1 or 0) + 8 * (1 or 0) + 16 * (1 or 0) + 32 * (1 or 0) + 64 * (1 or 0) + 128 * (1 or 0) + 256 * (1 or 0) + 512* (1 or 0)), where it’s 1 if the cell at that column changed, and 0 otherwise. So for example, if cells 1 and 4 changed, then the flags are (1 * 1 + 2 * 0 + 4 * 0 + 8 * 1) = 1 + 8 = 9.

The rest of the columns/fields represent the actual collection columns and their values.

Parameters:

lValueID – the value ID of the collection value.

Return Values: The record set representing the collection value. This value would be NULL/Nothing on error.

Important:

The data type on the field should tell you the column type:

Integer columns are “adVarChar” with size “30”.

Float Columns are “adVarChar” with size “31”.

Date Columns are “adVarChar” with size “33”.

Short String (32) Columns are “adVarChar” with size “32”.

String (255) Columns are “adVarChar” with size “255”.

Link Columns are “adVarChar” with size “256”.

String (2000) Columns are “adVarChar” with size “2000”.

File Columns are “adVarChar” with size “142”. The value format is “<ValueID><delimiter><HasValue 0 or 1><delimiter><Caption>”. 142 is 11 + 1 + 1 + 1 + 128. 11 is is for the max value id, 1 is for the delimiter, 1 is for the has value flag (0 or 1), 1 is for the delimiter, and 128 is for the caption.

Picture Array Value Columns are “adVarChar” with size “46”. The value format is “<ValueID><delimiter><HasValue 0 or 1><delimiter>Total Pictures = <Number>””. 44 is 11 + 1 + 1 + 1 + 32. 11 is is for the max value id. 1 is for the delimiter, 1 is for the has value flag (0 or 1), 1 is for the delimiter, and 32 is for the “Total Pictures = %d”.

SetTableValue

([in] LONG lValueID, [in] IUnknown *rsTableValue, [out, retval] BOOL *bSuccess)

Purpose: Given a record set “rsTableValue” and the value ID “lValueID” of the collection table, this function will take that record set and will save its content to the collection value with the specified value ID.

The record set is assumed to have the same configuration as the record set returned by the GetTableValue. Namely, the first three columns of the record set should be the following:

The first column is called “Row_Num” which is the original row number (1-indexed).

The second column is called “New_Row_Num”, which is the new row num if the user needs to modify this record set later. Initially this column has the exact values as the previous column (1-indexed as well).

The third column is called “Changed_Flag”, which is a decimal value representing which cells has changed in the row. Initially the values of this field are all set to 0. The value is the addition of (1 * (1 or 0) + 2 * (1 or 0) + 4 * (1 or 0) + 8 * (1 or 0) + 16 * (1 or 0) + 32 * (1 or 0) + 64 * (1 or 0) + 128 * (1 or 0) + 256 * (1 or 0) + 512* (1 or 0)), where it’s 1 if the cell at that column changed, and 0 otherwise. So for example, if cells 1 and 4 changed, then the flags are (1 * 1 + 2 * 0 + 4 * 0 + 8 * 1) = 1 + 8 = 9.

The rest of the columns/fields represent the actual collection columns and their values.

Following are the guidelines the calling application should follow in modifying a record set:

The calling application should never modify the “Row_Num” field.

If the calling application needs to delete a row, it should set the “New_Row_Num” value of that column to “0”.

If the calling application needs to add or insert a row, it should use the standard methods of adding a new set to the record set, but it should also set “New_Row_Num” field to the row number where this row should be (1-indexed). If the row was inserted (meaning location of other rows need to change), then the application should set the “New_Row_Num” field on all the other affected sets.

If the calling application needs to modify a row, it should use the standard methods of modifying the fields in a record set, but it should also set the “Changed_Flag” field to the proper value which marks which fields got modified. If this value is not set properly, then the cell values will not be written to the database.

The function will start a transaction if no modify transaction was started and will also lock the required objects for the update operation. It will lock the activity record if the value is inspection; the system type if the value is reference and the system if the value is characteristic.

Parameters:

lValueID – the value ID of the collection value.

rsTableValue – the record set representing the new collection value to save.

Return Values: Boolean: TRUE if successful, FALSE otherwise.

Important:

The data type on the field should tell you the column type:

Integer columns are “adVarChar” with size “30”.

Float Columns are “adVarChar” with size “31”.

Date Columns are “adVarChar” with size “33”.

Short String (32) Columns are “adVarChar” with size “32”.

String (255) Columns are “adVarChar” with size “255”.

Link Columns are “adVarChar” with size “256”.

7
SECURITY

User Administration within Aware has been developed to address the security concerns of users. Various levels of access are used to prevent accidental damage to data and protect business sensitive information but at the same time provide the necessary information to the users.

The different levels of access to data within Aware are provided by a group concept. The Aware administrator can create the different user groups and associate the access rights to these groups. Individual users are then assigned to a group depending on the level of data access that they may need. Security is be further defined at the object and attribute levels. Permissions can be assigned to user or groups at the class and attribute levels.

Group Rights

Aware implements a security model in which users belong to groups and derive their user rights from the group rights. This allows for easy administration of many users by their work functions. This model allows for easy classification of users into groups that have:

- Full rights to configure the system
- Rights to modify the system hierarchy to add, remove or modify existing components
- Rights to enter and view information - create new events, new activities, etc.
- Rights to only view data

Types of Access

1. Group Rights – the user has the rights of the group that the user is in.

2. Read Only –The user has read only rights to the system node or attribute for which this type of security is defined. This is further defined below for each type of security.

3. No Access – The user has no access to the system node or attribute for which this type of security is defined.

Permissions Hierarchy

Permissions can be set for either the group level or the user level. Permissions are checked first at the group level and then at the user level. Therefore, the user level supersedes the group level if both are set for a particular user.

Types of Security

Aware security can be set to allow or deny access to certain areas of Aware. It can be set for the following areas:

System Types

System Type security determines the access given to system types. The results of this security are seen in the navigation hierarchy (either the tree or list view).

System Type Attributes

System type attribute security determines the security for each attribute that is defined for a system type. The results of this security setting are seen in the characteristic attributes.

Activities

Activity security determines access to an activity. The results of this security are seen in the History page.

Activity Attributes

Activity Attribute Security determines access for each attribute that is defined for an activity. The results of this security are seen in the History Update and View pages.

Stored Queries

Stored query security determines access to each stored query. The results of the security can be seen in the Search and Search Edit pages.

Stored Query Categories

Stored query category security determines access to stored query categories. The results of the security can be seen in the Search and the corresponding edit pages.

Start Node

Each group or User is assigned a starting node in the database. It determines the highest level in the hierarchy that a user or group will have access. The results of the security setting can be seen in the hierarchy.

One of the goals of the Asset Performance Database is to compare with industry averages. This implies that information needs to be retrieved from nodes that are not owned by the user. Stored Queries that use a static start node can return results that are not part of the sub-tree defined by the root node of the current user. The system name and system path columns of the results will contain a generated name and an empty system path respectively for any rows that are not part of the sub-tree. The generated system name will be of the form Sysnnnnnn where nnnnnn is a zero-filled number corresponding to the internal ID of the system. This feature allows information to be made available without providing the actual identity of the associated systems and owner information.

Program:

Transmission Systems Asset Management & Utilization

About EPRI

EPRI creates science and technology solutions for the global energy and energy services industry. U.S. electric utilities established the Electric Power Research Institute in 1973 as a nonprofit research consortium for the benefit of utility members, their customers, and society. Now known simply as EPRI, the company provides a wide range of innovative products and services to more than 1000 energy-related organizations in 40 countries. EPRI's multidisciplinary team of scientists and engineers draws on a worldwide network of technical and business expertise to help solve today's toughest energy and environmental problems.

EPRI. Electrify the World

Printed on recycled paper in the United States of America

1002133

EPRI • 3412 Hillview Avenue, Palo Alto, California 94304 • PO Box 10412, Palo Alto, California 94303 • USA
800.313.3774 • 650.855.2121 • askepri@epri.com • www.epri.com