Spring 2008

A system for automating the interpretation of Analytical Ultracentrifuge data

Bradley W. Langhorst
University of New Hampshire, Durham

A SYSTEM FOR AUTOMATING THE INTERPRETATION OF ANALYTICAL ULTRACENTRIFUGE DATA

BY

BRADLEY W. LANGHORST

B.S., University of Connecticut, 1997

DISSERTATION

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy

in

Biochemistry

May, 2008

This dissertation has been examined and approved.

Dissertation director, Thomas M. Laue, Professor of Biochemistry

Rick H. Cote, Professor of Biochemistry

Thomas P. Moody, Research Associate Professor of Biochemistry

R. Daniel Bergeron, Professor of Computer Science

Borries Demeler, Associate Professor of Biochemistry

David B. Hayes, Postdoctoral Research Fellow

Walter F. Stafford III, Senior Scientist

9 April 2008

Date

ACKNOWLEDGMENTS

This work was generously supported by a dissertation fellowship from the UNH Graduate School, Biomolecular Interaction Technology Center funding and NIH grant #R01RR022200. In addition, I wish to thank the members of my thesis committee for their advice and encouragement. Finally, I wish to acknowledge my wife's generous support and editing assistance.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

A SYSTEM FOR AUTOMATING THE INTERPRETATION OF ANALYTICAL ULTRACENTRIFUGE DATA

by

BRADLEY W. LANGHORST
University of New Hampshire, May, 2008

In order to accelerate the development of knowledge about protein associations, further improvements to acquisition, sharing, and analysis of Analytical Ultracentrifugation (AU) data must be made. XML data formats have been defined to allow complete exchange of the information associated with an AU experiment. Extensible formats to record solution identity, instrument setup parameters, acquired data, and analysis of that data have been developed. A normalized relational database to allow storage and searching of this data has also been created. A computer program called PANDaS (Protein Association Network Data Server) was built to interact with any software conforming to the data exchange standard. Conforming software will be able to access centrifuge data with the complete context required for analysis. It is also possible to import existing data files to the database and convert them to the exchange format using the import tool described in this dissertation. Aggregation of multiple analysis results through PANDaS allows for machine-assisted interpretation of results from heterogeneous analysis techniques and multiple experiments. This common data exchange format and attendant software are necessary underpinnings that enable the diverse developers of AU analysis software to collaborate more effectively. Such collaboration facilitates the construction of a system of interacting software tools that will improve our understanding of molecular associations as the effectors of interesting biology.

CHAPTER 1

INTRODUCTION

1.1 Problem Description

Interesting biological systems are driven by molecular interactions. To understand how a particular biological process works, we must identify the molecules involved. The various sequencing efforts are populating databases with lists of all the possible molecular parts. This huge, ongoing effort produces a valuable tool, but tells us very little about any specific biological process. The bottleneck on understanding is shifting from the building of a parts list to understanding how those parts work together.

There are a number of techniques that provide qualitative information about molecular interactions, but only a few that yield quantitative information about the subtleties of molecules in solution. The analytical ultracentrifuge is a powerful tool for understanding the details of interactions between molecules, but the technique requires specialized knowledge and careful data analysis. If we are to increase our rate of knowledge acquisition, we must improve the centrifuge and analysis software so we can acquire and interpret more data with less effort.

The Laue lab is working to improve Analytical Ultracentrifugation in multiple areas. The new fluorescence optical system (FDS) expands the range of conditions that can be observed with the centrifuge. The Advanced instrument Operating System (AOS) has been developed to allow easier training and instrument control. Plans for a rapid scanning absorbance system will increase the rate of data acquisition. Higher density rotor and cell designs will allow more data to be collected during a single experiment. These improvements shift the bottleneck from data acquisition to analysis of the coming torrent of data.

1.2 Background

1.2.1 Centrifuge overview

The analytical ultracentrifuge spins a rotor populated with 1–7 cells of solution and a counterbalance (Figure 1-1). A solution component's movement in the synthetic gravitational field is governed by its mass and hydrodynamic radius. The purpose of the experiment is to elucidate the molecular interactions between the solution components. In simple systems, mass distribution gives us a clear picture of how many molecules are associated with each other and can provide information about the nature of the association. The molecules of interest may be observed using a number of optical

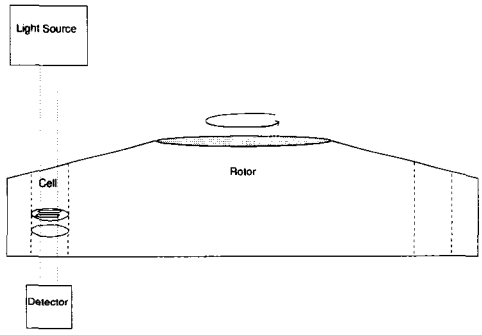


Figure 1-1: Analytical ultracentrifuge rotor

systems. An absorbance optical system is able to detect the concentration of most proteins by measuring the amount of light energy absorbed by their chromophores (aromatic amino acids, or protein backbone itself). The absorbance system is only sensitive to the components of the solution that absorb light at the chosen wavelength, so a limited ability to seletively measure only a single component is available. A Raleigh interference optical system measures the difference in refractive index ($\therefore$ solute concentration) between a reference and a sample. Interference is not as sensitive as the absorbance system on a per scan basis, but more scans can be acquired in the same period of time so it is possible to increase the signal-to-noise ratio by considering multiple scans. Since all components of a solution contribute to its refractive index, interference optics are sensitive to the

whole solution and cannot select only specific components to observe. It is possible to study higher concentration solutions with interference optics than with absorbance optics. The new fluorescence optical system is able to detect the presence of a fluorophore in a complex solution without much signal contribution from other solution components. It has high sensitivity and is usable over a wide concentration range. Molecules of interest must usually be tagged with a fluorophore to be studied with the the fluorescence optical system.

A major advantage of Analytical Ultracentrifugation is that no standards are needed for calibration; the molecular parameters are determined from first principles. An analytical ultracentrifuge is typically operated in velocity mode or equilibrium mode.

1.2.2 Velocity

In a velocity experiment [44] [42] (Figure 1-2), the centrifuge is spun fast enough that molecules of interest are moved quickly from the inside of the rotor to the outside, but not so fast that they are difficult to observe. A molecule's movement is governed by the centrifugal force from the spinning rotor, the drag it experiences as a result of solution interactions, and diffusion. By measuring the rate at which the molecule moves down the cell and the amount of diffusion that has taken place, it is possible to determine the the molecule's mass and some information about its shape. A velocity experiment usually takes a few hours to complete.

1.2.3 Equilibrium

An equilibrium experiment [43] [47] (Figure 1-3) provides more detailed information about the interactions and stoichiometry of a multimeric complex, but can take days to complete. Component analysis is performed by fitting combinations of exponential equations to the experimental results. Given the inherent ambiguity in this type of curve fitting, it is necessary to have source data at multiple concentrations to eliminate incorrect models. In the example of a monomer–dimer system, the observed exponential will be the distribution of the monomer component overlaid with that of the dimer. By determining the relative amounts of each component under various conditions, it is

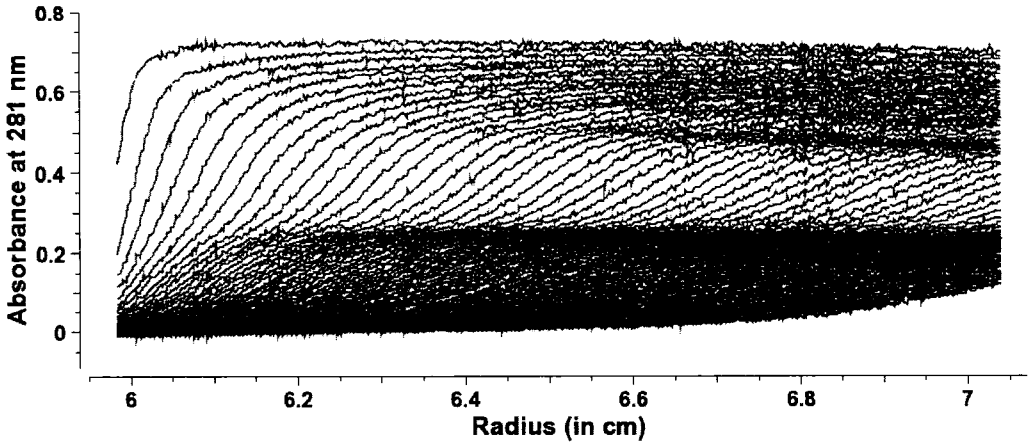


Figure 1-2: Typical Velocity Experiment

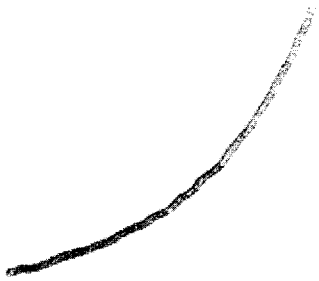


Figure 1-3: Typical Equilibrium Experiment

possible to learn about how tightly the two components interact with each other.

1.2.4 Other techniques

Other quantitative techniques are available to probe molecular interactions, but all suffer from similar data analysis bottlenecks. Some of the tools described here may be applicable to those sorts of problems as well. By grouping results from light-scattering, mass spectrometry, isothermal calorimetry, and other techniques with results from Analytical Ultracentrifugation, a more complete and reliable picture of a molecule's behavior can be obtained.

1.3 Related Efforts

Other software has been written in an effort to unify data analysis and experimental information.

1.3.1 Sednterp

Sednterp [21] is a program that stores information about a sedimentation experiment and estimates solution density and viscosity using solute concentrations and solvent properties. It encapsulates some of the specialized knowledge a centrifugation expert has and simplifies data interpretation. It lacks a flexible method of data storage or a way to group and co–analyze similar experiments. Sednterp cannot send or receive results from distributed users and must be manually updated with new data and corrections.

1.3.2 Ultrascan

Ultrascan [16] encapsulates a large number of analysis techniques into a monolithic program. The system proposed here is of a more modular design. While there is significant overlap of goals between Ultrascan and this project, it should be possible to take advantage of the significant effort that has been put into Ultrascan over the years by integrating its analysis and browsing tools with this data store using the interconnection software described in this document.

1.3.3 Data Flow

As part of this dissertation the following tools were built:

- An extensible, standard format for sharing experimental results (section 2.3)
- An extensible, standard format for sharing analysis results (section 2.5)
- A scalable, normalized database with full referential integrity constraints (Chapter 4)
- A fine–grained security model for regulating access to this data (section 3.3)

- A utility to import existing centrifuge experiment data to the database (section 5.1)
- A network data server to allow aggregation of analysis results from programs written by many authors and to interact with Sednterp (Chapter 3)
- A Tableau [31] workbook for use as meta-analysis tool to group and view analysis results from multiple experiments (section 5.3)

With this project's tools, centrifuge data flows from the centrifuge, through AOS, to the legacy Beckman directory layout (Figure 1-4). Users may then use the Sucker (section 5.1) to import the data and specify solution details for each channel. PANDaS (Chapter 3) makes this data available to any program implementing the common XML data format. As results of analysis flow back into the database via PANDaS, they are available for meta-analysis (section 5.3). In a future revision of AOS, data will flow directly to PANDaS instead of requiring the Sucker.

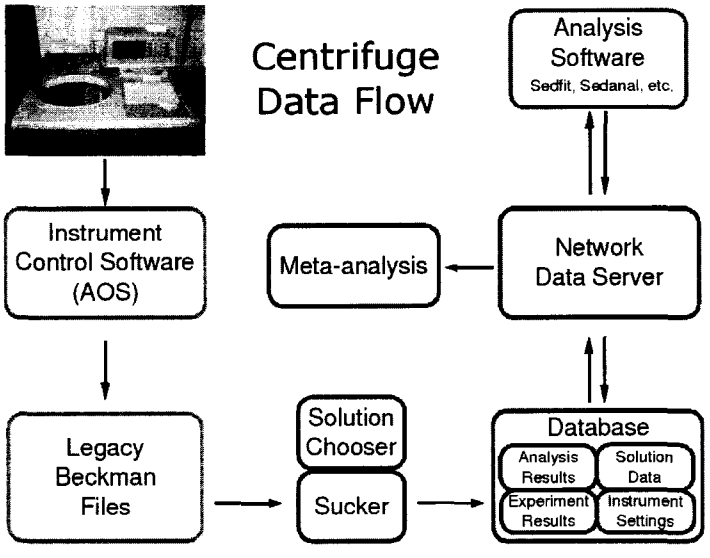


Figure 1-4: Centrifuge Data Flow

CHAPTER 2

INTERCHANGE DATA STRUCTURES

2.1 Reasons to Implement

2.1.1 Efficient binary format

When analytical ultracentrifuge data is acquired using the Beckman data acquisition software or with the current version of the Advanced instrument Operating System (AOS) [29], data is stored in ASCII text files. Storing a position and absorbance value as ASCII characters requires approximately 20 bytes. It is far more efficient to store this data in a binary format using only 4 bytes per data point. Each optical system is best equipped to decide on a data storage structure, so this schema does not specify any particular format for scan data but includes a raw representation of the data directly in the file. Since new tools will be required to deal with this data interchange format, it makes sense to simultaneously switch to a binary data representation even for legacy Beckman data files.

2.1.2 "Decoratable" tree

It is possible to "decorate" a data tree with transformations identifying key features, notes about a run, analyses of data, and other modifications as an experiment progresses through a set of analyses. The final decorated tree contains a history of what has happened to a particular set of data files over time.

2.1.3 Extensible format

XML [38] is inherently extensible, and this format has been explicitly designed for easy extension. New optical systems, analysis methods, arbitrary references to external information, and other extensions can be accommodated by combining this schema with an extension schema or by simply adding site-specific structures to the XML document.

2.1.4 Searchable and transformable with standard tools

Technologies such as XSLT [10] and XQuery [37] allow for automatable data extraction and file modification for XML documents conforming to a well-defined schema.

2.1.5 Easy exchange with colleagues

Instead of sending compressed archives of data and text descriptions of run conditions, this work allows the exchange of a single file (or a link to it). Since analysis results may be attached to an experiment data tree or exist as independent documents, collaborators have the flexibility to communicate how data was acquired and analyzed with minimal effort and decreased possibility for misunderstanding.

2.1.6 All important information in one file, easy to archive

Without an interchange data structure, a researcher must correctly identify all experiments belonging to a project and build an ad-hoc archive using compressed files. Each of these archived experiments must be associated with a description of its run conditions and be reliably stored together - probably requiring references to lab notebooks to allow for successful re-analysis in the future. Archive errors may be eliminated if it is only necessary to archive one file per experiment.

2.2 XML Schema Background

2.2.1 Purpose of schema

A standard for data interchange is of limited utility if only a few programs are able to read the format. The schema serves the important purpose of defining and documenting the data format. In addition to documentation, a schema allows for automatic validation and parsing of a potentially conforming document. By having the documentation for the data format integrated with a validation tool, any ambiguity caused by different readings of a text-based standard definition is minimized.

2.2.2 Schema language choice (RelaxNG)

RelaxNG [12] was chosen over other schema languages such as DTD [38] and XML Schema [45] because of RelaxNG's simplicity, narrow scope, and expressive power. RelaxNG schemas can be converted to other formats using the existing Trang [9] conversion utility.

2.2.3 Abstract vs. physical objects

To reduce duplication of data across a set of documents, properties of objects like rotors and centerpieces are split into "abstract" and "physical" sub-trees. Abstract objects contain properties that are consistent across all members of that type of object. For a Beckman An60Ti rotor, abstract properties include material and number of holes. Physical objects contain properties such as serial numbers that are unique to the particular object. Physical objects may incorporate an entire abstract object for maximum portability and archive purposes. However, they may also refer to abstract objects that are shared in a central database and made available to client software via a URI [6]. Avoiding duplication of information is worth the extra complexity of managing distinct abstract and physical representations.

2.2.4 Internal and external references

In some cases it is necessary to refer to the same object in multiple places in a single document. Rather than duplicating the entire object's definition in each location, XML has the inherent capability to make these references using the ID/IDREF data types. ID attributes must be unique within an XML document, but they do not need to be globally unique. To avoid conflict between IDs of different types, each ID is derived from a database primary key and prefixed with an abbreviation indicating what type of object it is (e.g. "opt6653" for an OpticalSystem object). In some cases, a Globally Unique Identifier (GUID) [30] is associated with an object to allow references to be generated in multiple environments without concern that they will overlap.

2.2.5 Extensibility features

This schema has been designed to be extended as new optical systems become available. It defines a format for storing viscosity, density and other common "meta" information about a solution, but allows for extension to include storage of unanticipated properties as well. Where appropriate, this schema allows for arbitrary content in conforming documents. Extension information is not standardized, so it may be ignored by software using this schema to validate documents. Custom tools could take advantage of the additional information with or without a custom schema.

Many components used in this schema have been broken into independent schemas so it is possible to combine them into one large schema that can validate an entire experiment, or use the individual components alone.

This is a "flattened" schema, in which most elements contain references to collections of elements, instead of references to a single element with children. The flat format was chosen to allow similar elements to share a common definition and to allow new objects to be defined that extend existing ones.

2.2.6 Backward compatibility

Analysis software will gradually be updated to take advantage of the information stored in an experiment document, but it must be possible to analyze data using existing software. A simple program to convert a document in this XML format to a series of files in the traditional Beckman file system layout could easily be written using XSLT [10].

2.3 Experiment Schema Structure

2.3.1 Experiment

Each document conforming to this schema contains a single experiment which corresponds to a run of the instrument, possibly at multiple speeds and using multiple optical systems. At the top of the tree is the experiment node (Figure 2-1). This node contains a GUID, the begin time of the experiment, an optional experiment name and an optional end time of the experiment. It also contains references to a series of sub-trees representing the instrument, operator, lab, rotor, cells, centrifuge protocol, and analyses. Cells have been kept at the same level as the rotor to decrease unnecessary tree depth and allow for easier querying.

2.3.2 Instrument

The Instrument sub-tree (Figure 2-2) contains information about the instrument used in the experiment. The schema requires only an instrument name and GUID. It specifies the possibility of a serial number, but unspecified additional details are also allowed by the inclusion of the Other element.

2.3.3 Operator

The Operator sub-tree (Figure 2-3) contains a reference to a simple Person object. Only a person's name is explicitly required, but without specification a public access key and AuthURI, the Person object will not be useful for authentication purposes (section 3.3). Other information may be added

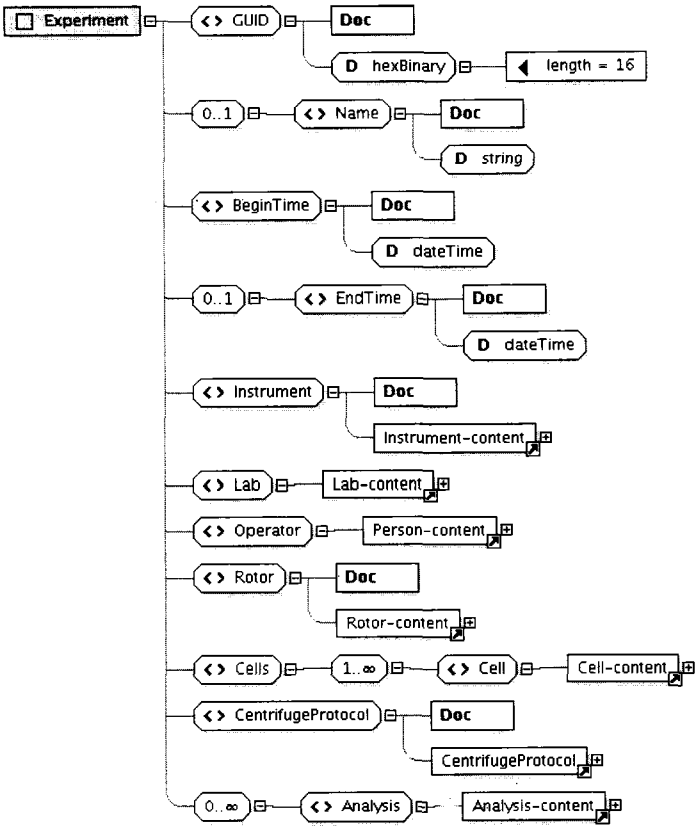


Figure 2-1: Experiment Tree

as necessary.

2.3.4 Lab

The Lab sub-tree (Figure 2-4) describes the physical location of the experiment. Only the name of the lab is required. Room and building information can also be documented here. The Other element is included to allow unanticipated information about the lab to be included.

2.3.5 Rotor

The Rotor sub-tree (Figure 2-5) describes the physical rotor used in the experiment. It refers to an AbstractRotor (Figure 2-6) object to encapsulate the characteristics of the rotor type (number of holes, speed rating, material type, etc.). A name for the rotor and a serial number are required

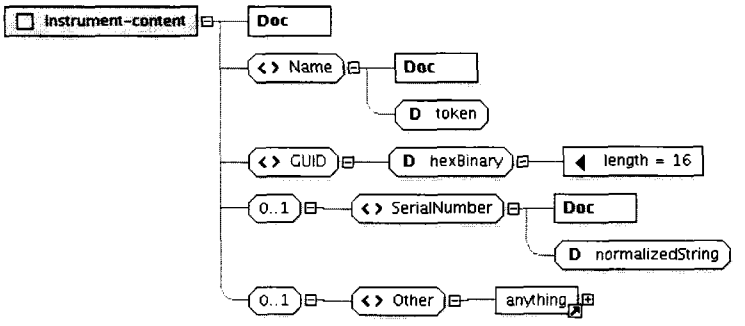


Figure 2-2: Instrument Sub-tree

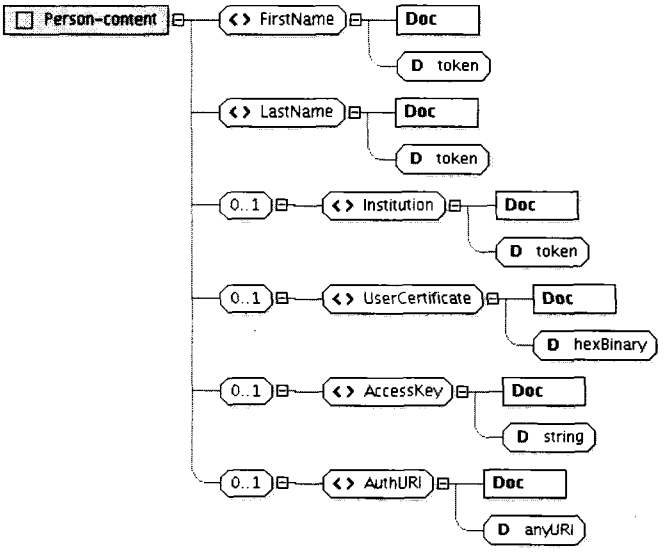


Figure 2-3: Person

elements. It is possible to store a series of stretching functions and/or a log of the rotor's hours along with the physical rotor so that changes in its material properties may be evaluated in later analysis.

2.3.6 Cells

The Cells sub-tree (Figure 2-7) is a collection of one or more cell nodes. While it would be valuable to check that the number of cells matches the number of holes in the rotor, this is not possible with RelaxNG alone. Schematron [11] provides more complex rule-based validation and could be incorporated if necessary. Each cell indicates the hole number it was placed in during the experiment,

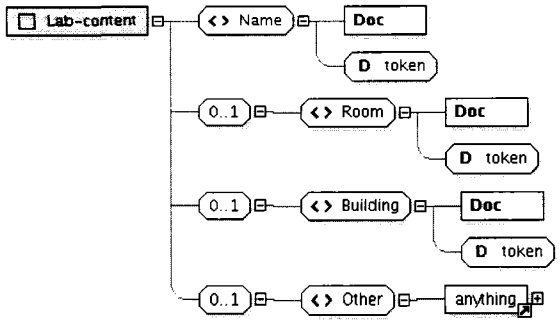


Figure 2-4: Lab Sub-tree

the type of window used, and a reference to the centerpiece (Figure 2-9). The Centerpiece sub-tree records the position, identity, and relevant properties of all the solutions in the experiment. The primary data acquired in the experiment resides in the Channel sub-tree (Figure 2-13). Scans are the atomic unit of data acquisition and may be contained in their own document containing references to their parent experiment and channel or in arbitrary EditGroup objects. For these references to be useful, the master experiment document must contain all the channels sampled in the experiment. As scans are added to an ongoing experiment, the channel GUID and experiment GUID are known by the acquisition software and may be passed with the scan so it can be properly associated with the experiment.

2.3.7 Centrifuge protocol

The CentrifugeProtocol sub-tree (Figure 2-8) describes the instrument settings in use during the experiment. It contains a record of all the optical systems used, scan frequency, acceleration profiles, etc. Existing optical systems have been specified in detail in Appendix A (figures A-1, A-2, A-3, A-4 and A-5). RelaxNG provides an "interleave" mechanism to combine a new schema file with details about a new optical system and this schema. It is also possible to include an optical system in this schema as an "OtherOpticalSystem" (Figure A-6) that allows any set of elements and attributes in its description.

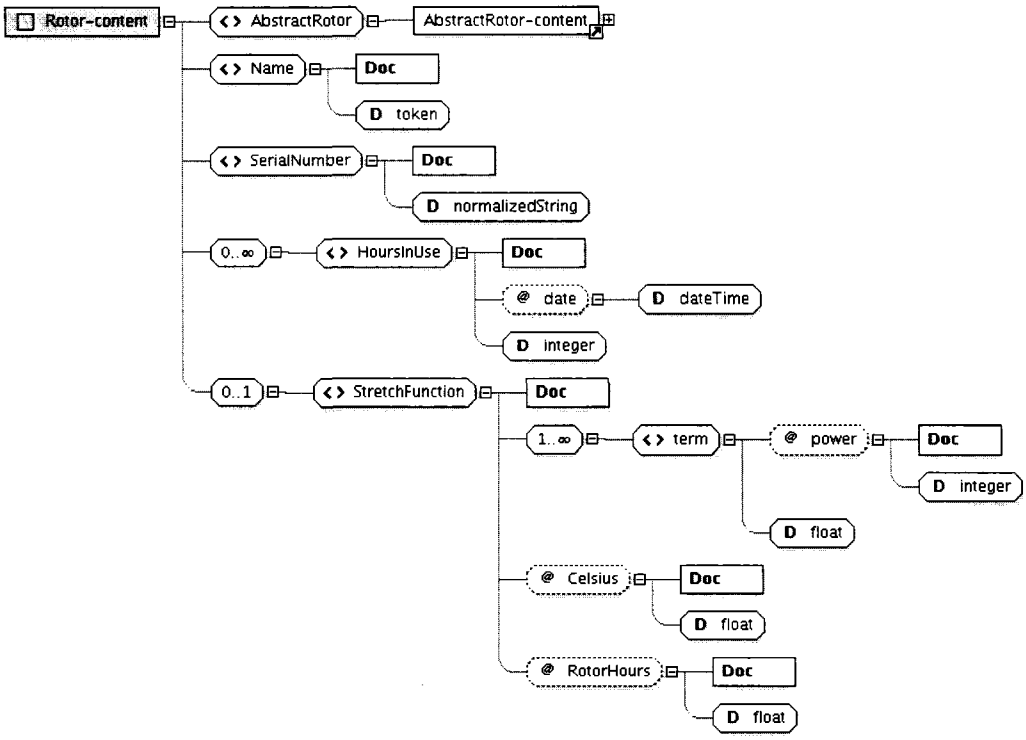


Figure 2-5: Rotor Sub-tree

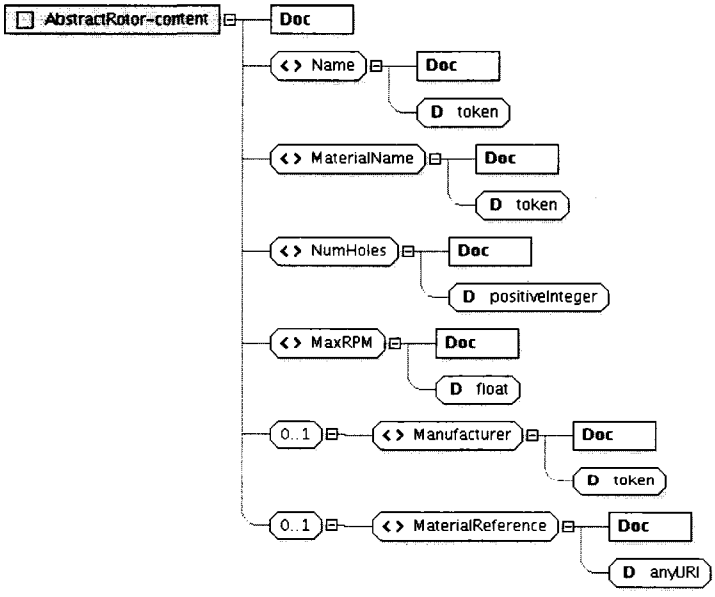


Figure 2-6: Abstract Rotor Sub-tree

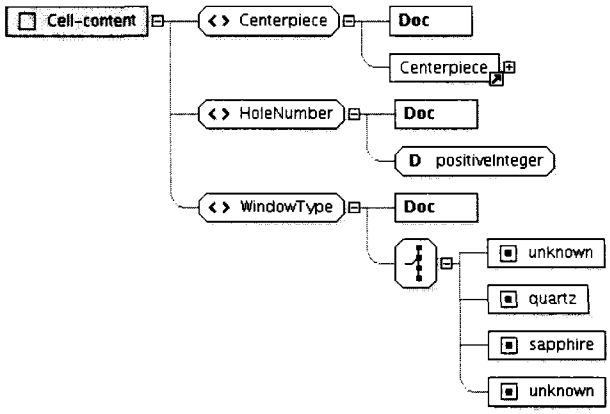


Figure 2-7: Cell Sub-tree

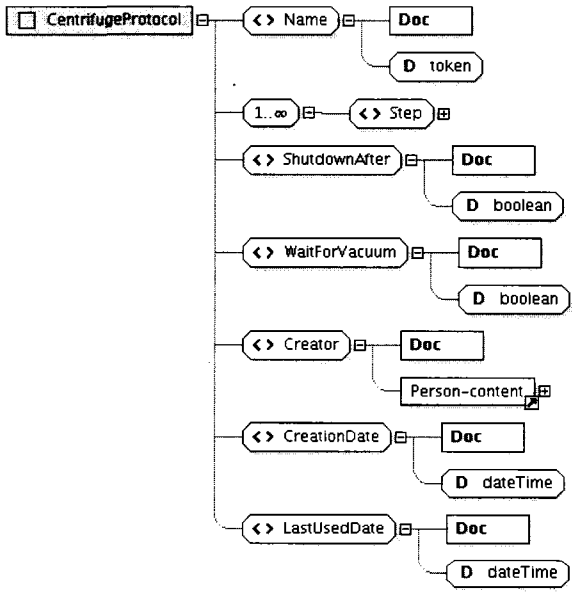


Figure 2-8: Centrifuge Protocol Sub-tree

2.4 Cell Detail

Each Cell sub-tree contains all the information about the organization and identity of the solutions to be studied in this experiment. It is organized in a hierarchy of Cell (figure 2-7), Centerpiece (figure 2-9), Channel (figure 2-13), and Scan (figure 2-26). It is split into two branches. Software requiring information about physical dimensions and positions of landmarks in cells (e.g., data acquisition software) will primarily use the AbstractCenterpiece sub-tree. Analysis software will primarily use the non-abstract branch of the Cell tree to determine the contents of channels and access acquired scan data.

Each cell is required to contain a Centerpiece object (Figure 2-9), which contains a list of channels, a serial number, and a reference to the abstract centerpiece type.

The AbstractCenterpiece (Figure 2-10) sub-tree stores the physical details of a centerpiece type. It specifies the speed rating, material type, number of channels, and a reference to a set of AbstractChannel objects that are found in the centerpiece. Each abstract channel (figure 2-11) specifies physical dimensions of the channel and its various properties, including information about channel types (figure 2-12). Some optical systems (e.g. interference) acquire data for an entire multi-channel centerpiece in one scan. In this case the data may be duplicated in each channel or placed in a special channel with a number of 0 to avoid data duplication.

Centerpiece objects also refer to physical channels. These objects contain information about the contents of a channel (Figure 2-14) and the data acquired during the experiment (subsection 2.4.2).

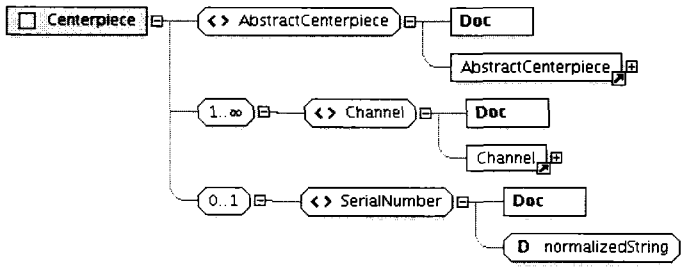


Figure 2-9: Centerpiece Sub-tree

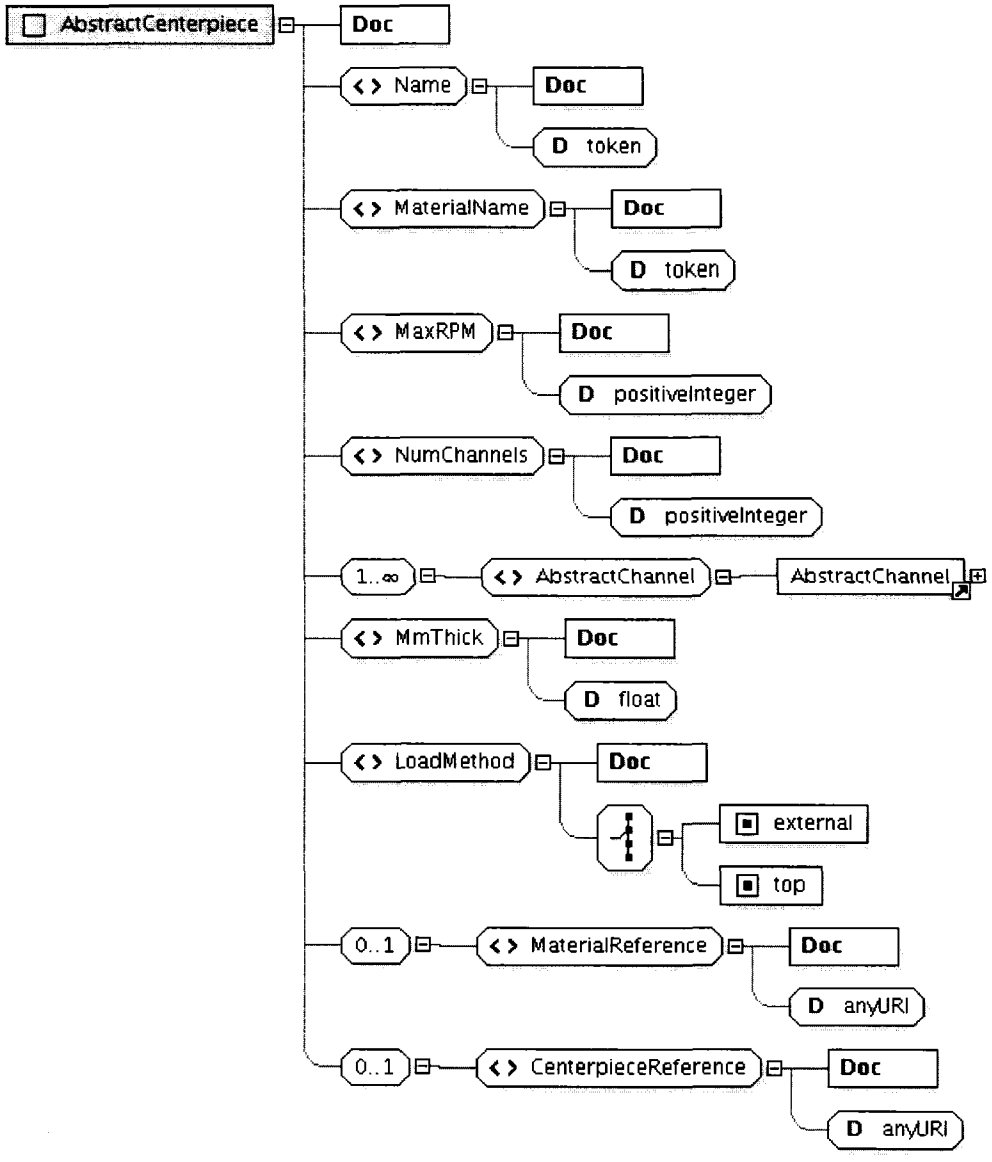


Figure 2-10: Abstract Centerpiece

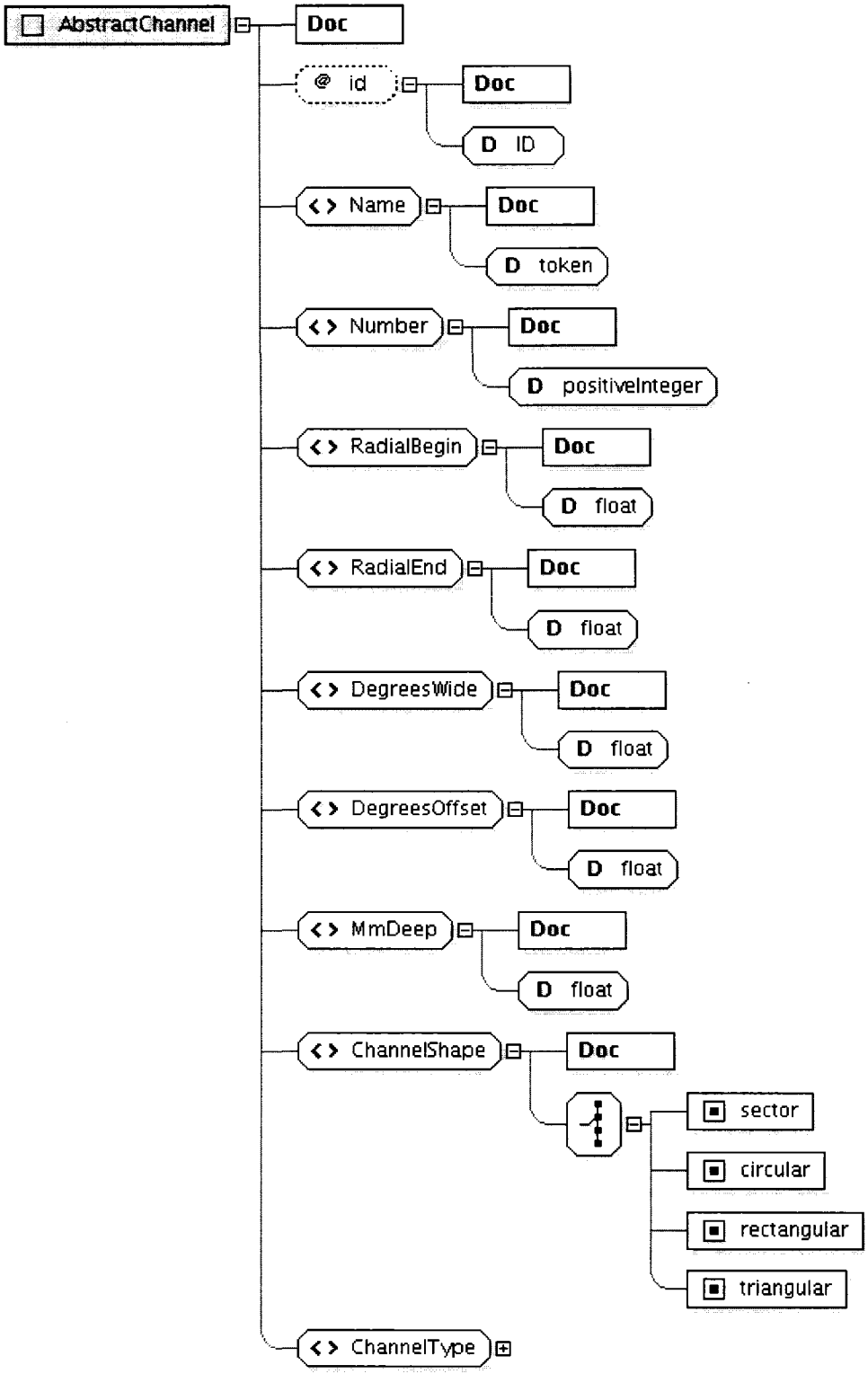


Figure 2-11: Abstract Channel

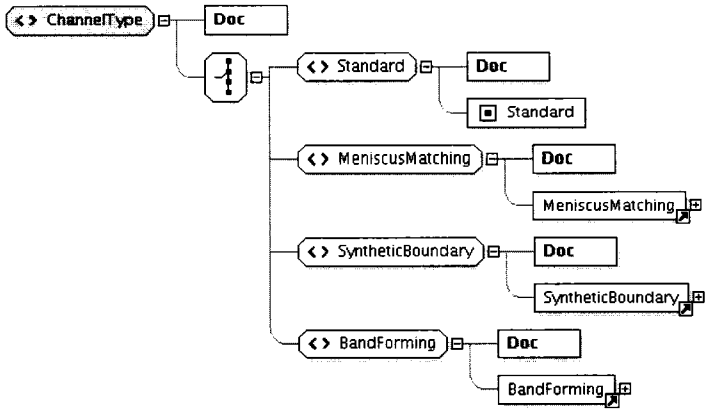


Figure 2-12: Channel Types

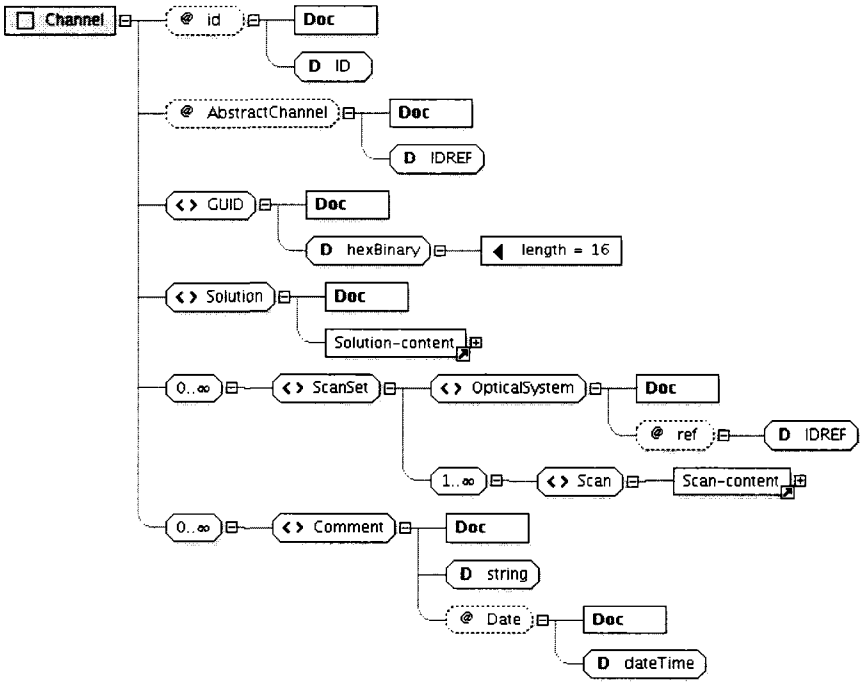


Figure 2-13: Channel Sub-tree

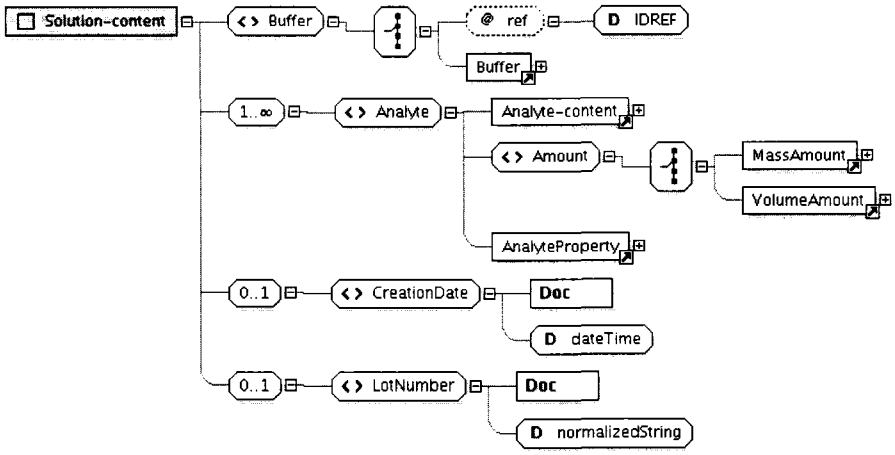


Figure 2-14: Solution Sub-tree

2.4.1 Solution

Solutions are defined as a combination of a set of analytes (Figure 2-15) with a buffer (Figure 2-19). Unlike many other sub-trees, the Solution sub-tree does not contain an abstract branch. It is assumed that solutions for analysis will be short-lived and not often reused, so an abstract branch would add complexity without reducing duplication of information. The PANDaS database does store abstract solutions, so it will be easy to add an abstract solution to this schema in a future revision if necessary.

The Analyte sub-tree (Figure 2-15) captures the properties of the set of molecules being studied in the experiment. Analytes are likely to be found in much lower concentration than other solution components, but their relative quantity should should still be recorded in mass or weight/weight concentration units to allow for the most precise calculations. Most measured analyte properties are stored in the Solution object (Figure 2-16) since they depend on the analyte's environment. Molecular weight is stored directly in the Analyte object since this is not expected to depend on the solution (Figure 2-17). The Analyte sub-tree contains an AbstractAnalyte branch (Figure 2-18) which contains information about this type of molecule.

For example, assume the analyte for an experiment is a batch of recombinant Bovine Serum Albumin that was produced in *E. coli* on 12/1/2004. The abstract analyte would contain the canonical

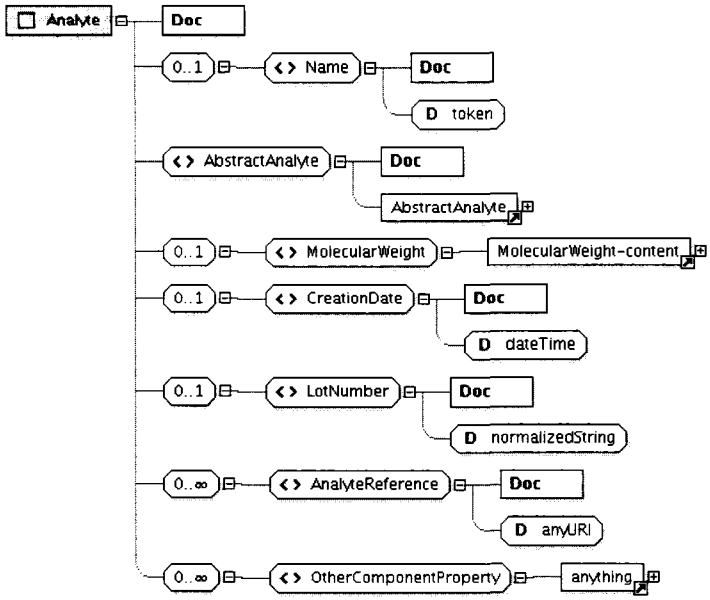


Figure 2-15: Analyte Sub-tree

molecular weight, the name of the analyte, and possible references to its sequence and structure. The main Analyte object might contain the measured molecular weight for this batch (including any modifications), a lot number, and production date.

The following analyte properties are explicitly specified in the schema (Figure 2-16):

- Partial specific volume (figure B-1). This item is required since it is a necessary input for many analysis programs.
- Quantum yield (figure B-2)
- Refractive increment (figure B-3)
- Extinction coefficient (figure B-4)
- Extinction function (figure B-5)
- Intrinsic viscosity (figure B-6)
- Valence (figure B-7)

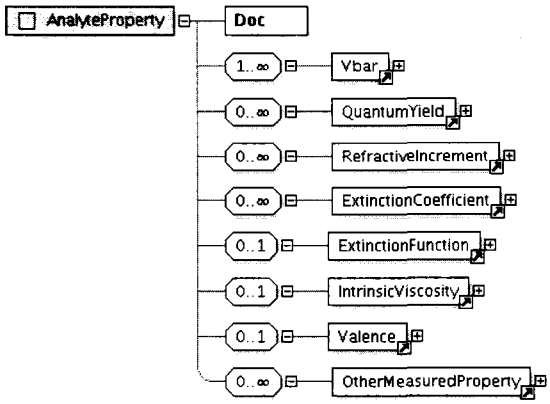


Figure 2-16: Analyte Properties

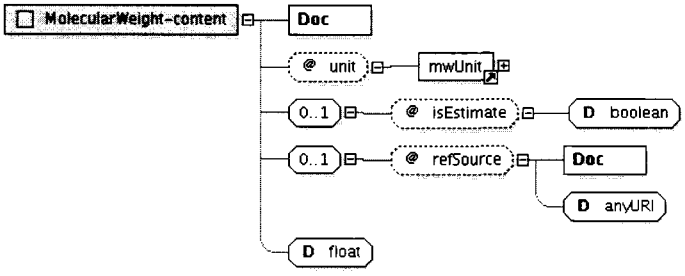


Figure 2-17: Molecular Weight

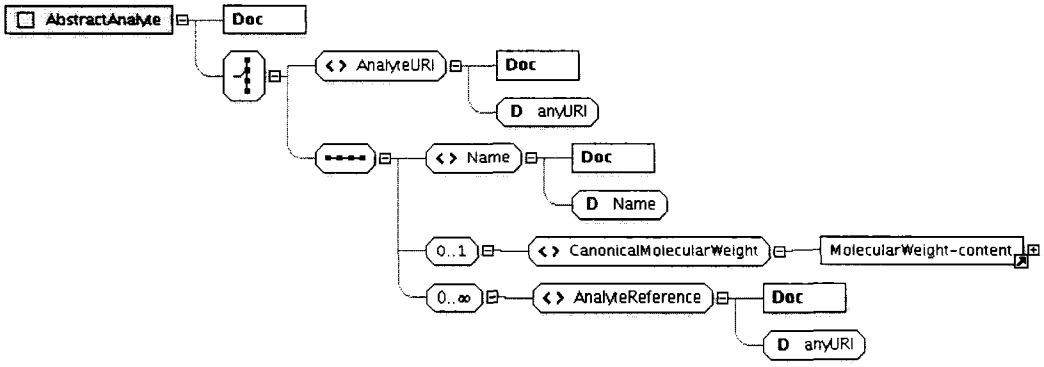


Figure 2-18: Abstract Snalyte

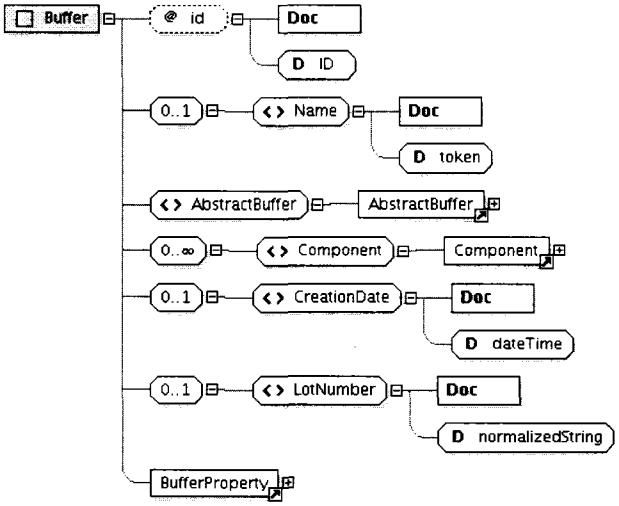


Figure 2-19: Buffer Sub-tree

A reference to an "other measured property" (figure B-15) node is included to allow unanticipated properties to be recorded.

The Buffer sub-tree (Figure 2-19) is similar to the Analyte sub-tree in that it can store lot number and creation date information. Buffers must contain a reference to an abstract buffer (Figure 2-22). Information about the actual components used in the buffer (Figure 2-20), and a series of measured properties of the buffer (Figure 2-21) may also be stored.

Component objects are intended provide any information that is dependent on the physical material used in an experiment. They contain references to the same abstract components found in the

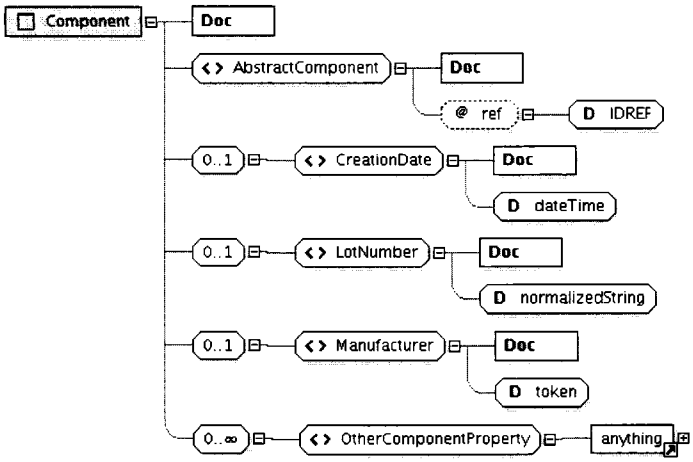


Figure 2-20: Buffer Component

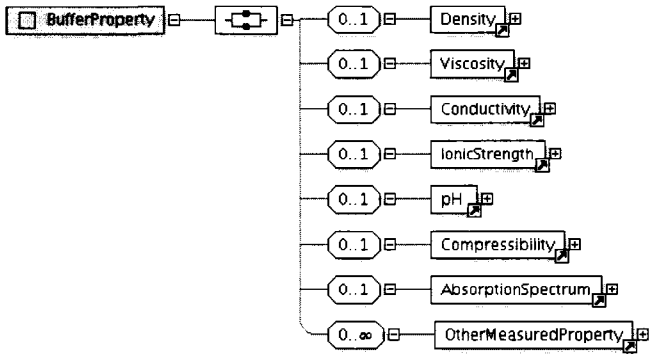


Figure 2-21: Buffer Properties

abstract buffer and include batch-specific information.

Commonly measured buffer properties are enumerated in the list of buffer properties (Figure 2-21). It is also possible to specify unanticipated properties through the use of the OtherMeasuredProperty elements (Figure B-15).

An AbstractBuffer sub-tree records the amounts of the various buffer components to define a type of buffer. Bulk solvent and minor components are both recorded as abstract components (Figure 2-23) and are most accurately reported in mass units. It is possible to record relative amounts of components in other concentration units if necessary. Each abstract component must be accompanied by a mass amount (Figure 2-24) or a volume amount (Figure 2-25).

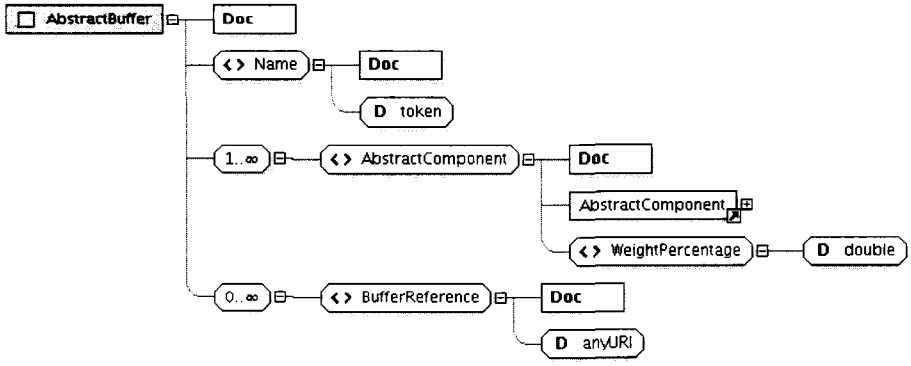


Figure 2-22: Abstract Buffer Sub-tree

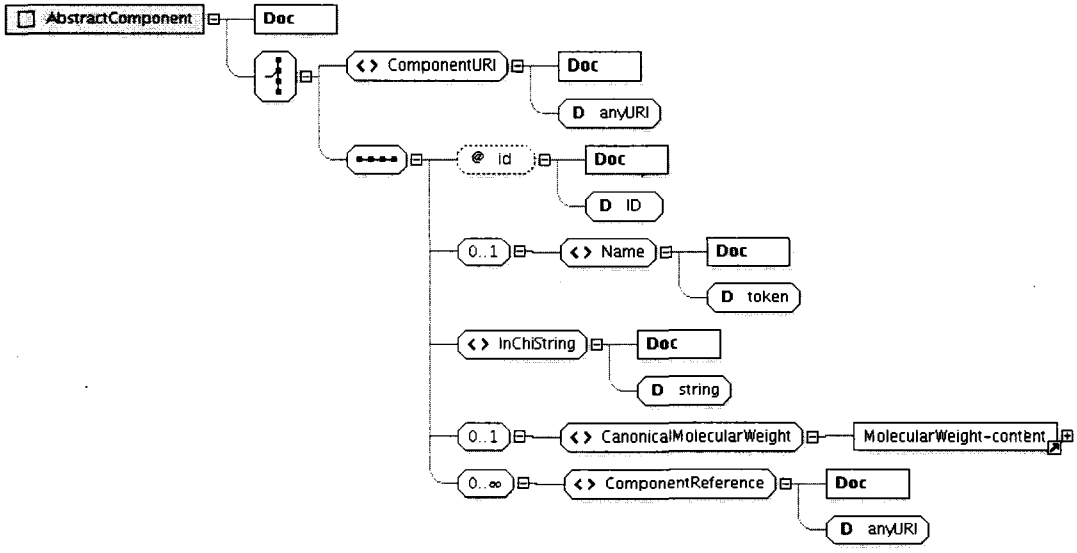


Figure 2-23: Abstract Buffer Component

An abstract component is made up of a name and an InChI [40] string. InChI is a notation developed by the International Union of Pure and Applied Chemistry (IUPAC) to precisely identify a molecule. An AbstractComponent object may also contain a molecular weight value. While it is similar to an abstract analyte, components do not require a human readable name. An InChI string identifier is strongly recommended, but not explicitly required because component molecules may be too large to represent as an InChI.

Amounts may be reported in either mass units or in concentration units. Concentration units are inferior to mass units since they depend on volume, which varies with temperature and pressure

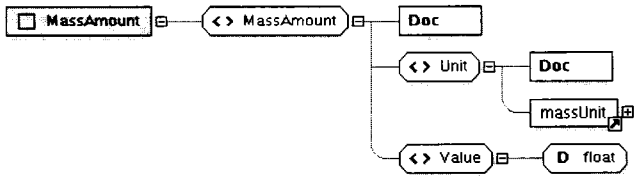


Figure 2-24: Mass Amount

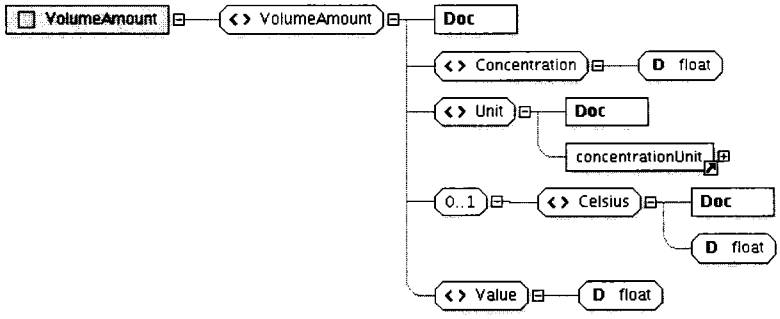


Figure 2-25: Volume Amount

changes. Temperature should be reported if concentration units are used. If mass units have been specified, the value in this element is the mass. If concentration units and a concentration have been specified, the value is the volume in the units specified in the denominator. This element is not intended to be easy for a person to specify directly, instead it is designed to maintain accuracy but allow storage of both concentrations and masses. A user interface program should ask the user for the amount of their analyte in units that are more commonly employed in the lab and calculate a mass. The user interface should warn the user about inaccuracy if high concentrations are specified in terms of concentration units.

2.4.2 Scan

Scans are stored in combination with a reference to the optical system (see Appendix A for more detail) used to acquire them. The actual data acquired by an optical system is stored in the Scan-Data element in a format defined by the developer of the optical system. Each scan includes the relevant information about instrument state at the time of the scan. Scans may be included in other

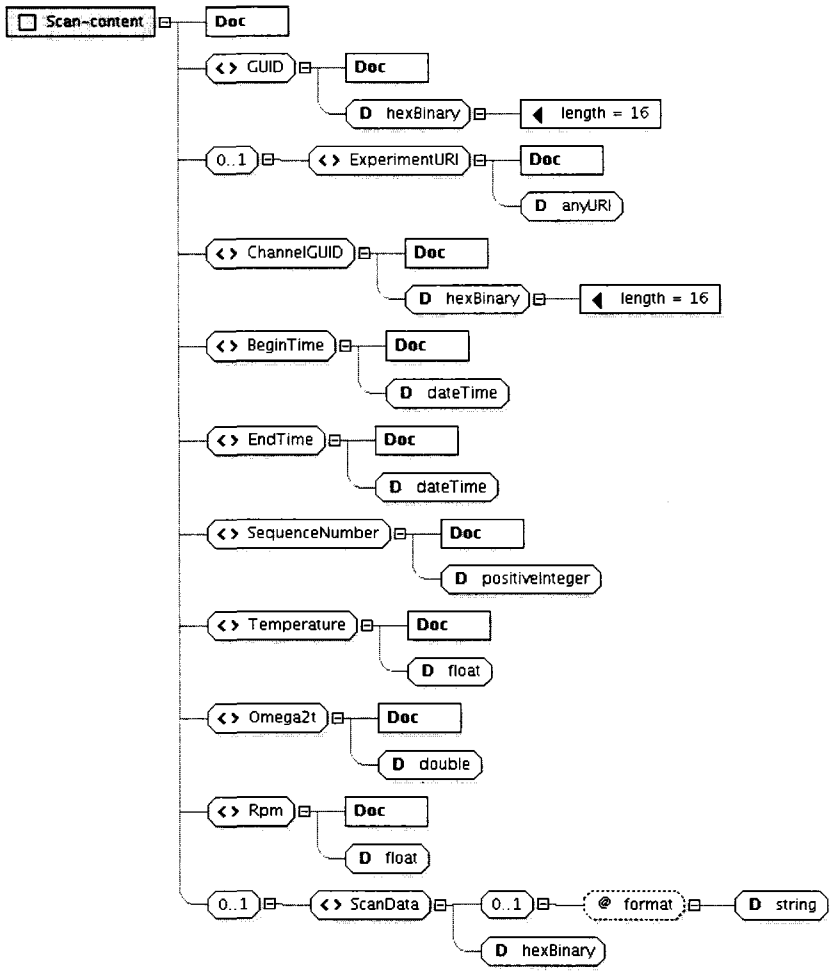


Figure 2-26: Scan

documents or exist as independent documents since the required ChannelGUID element will allow proper re-connection. If a scan is not part of an experiment document, it should contain a reference to the experiment that acquired the scan so the scan's original context can be located. Each scan has a GUID element to allow each scan to be uniquely identified even when separated from the experiment that acquired it.

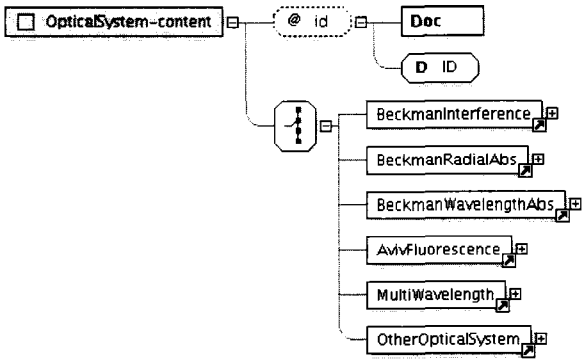


Figure 2-27: Optical System Sub-tree

2.5 Analysis Interchange Schema

A format for the communication and documentation of results of analysis has been developed to allow submission and aggregation of analysis results (Figure 2-28). By reading from this format, authors of diverse analysis programs such as DCDT+ [35], Sedfit [36], Ultrascan [15], Nonlin [25], HeteroAnalysis [14], Sedanal [39], etc. can take advantage of previously specified data landmarks (e.g. meniscus position), editing, and grouping of experimental results, without having to rebuild user interface tools to perform these steps in each distinct analysis program. Users benefit from not having to specify this information multiple times and from employing their preferred set of tools to perform editing, grouping and specification of positions. Having a common data format can even allow for combinations of components from the various analysis programs into an analysis tool-chain for further efficiency improvements. By writing to this format it becomes possible to store analyses from diverse programs in a database for later meta-analysis (section 4.3).

Each analysis is identified by a globally unique identifier so it may be retrieved in isolation. It must refer to the technique used in the analysis and the set of results generated by that technique (subsection 2.5.1). The identity of the person performing the analysis is stored in the User sub-tree. The person structure is defined in figure 2-3 and described in subsection 2.3.3. Finally, the set of data files included in the analysis is detailed in the EditGroup sub-tree (subsection 2.5.2).

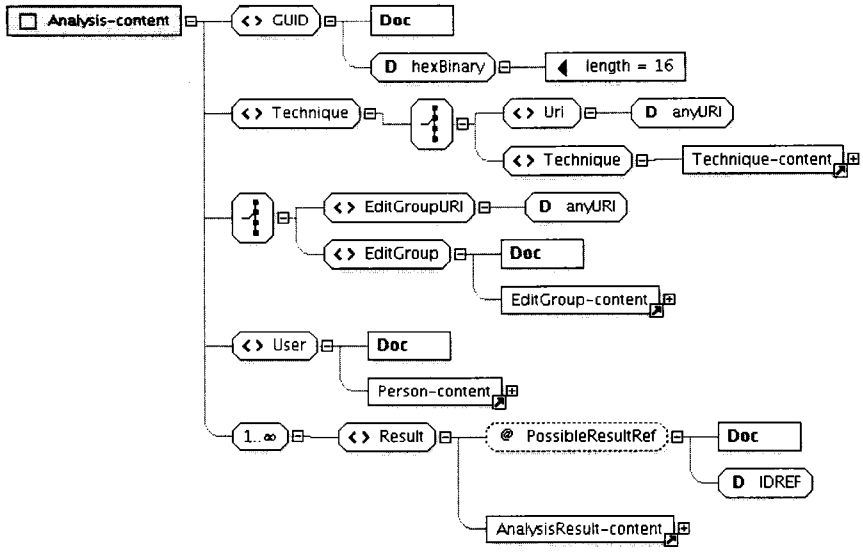


Figure 2-28: Analysis

2.5.1 Analysis technique

An analysis technique is identified by a human readable name with an optional version number and a GUID. Since each technique is tagged with a GUID, it is possible for the technique to be specified by a short URI of the form `http:\\server\nds\Techniques\?GUID`. This allows techniques from multiple databases to be mixed together into a single Analysis tree. It is also possible to embed the definition of the technique in the document (Figure 2-29). A technique's definition includes a list of the possible results (Figure 2-30) that the technique might produce.

Each possible result is identified by a name. When analyzing mixtures, more than one result of each type is may be produced, but only one possible result should be defined. Possible results might include "s value", "s-95confidenceinterval", "D", "molecular weight", etc. Possible results should also contain upper and lower bounds for reasonable values (e.g. 0 to 100 for s values), and a name of the unit in which the result will be reported (e.g. Svedberg).

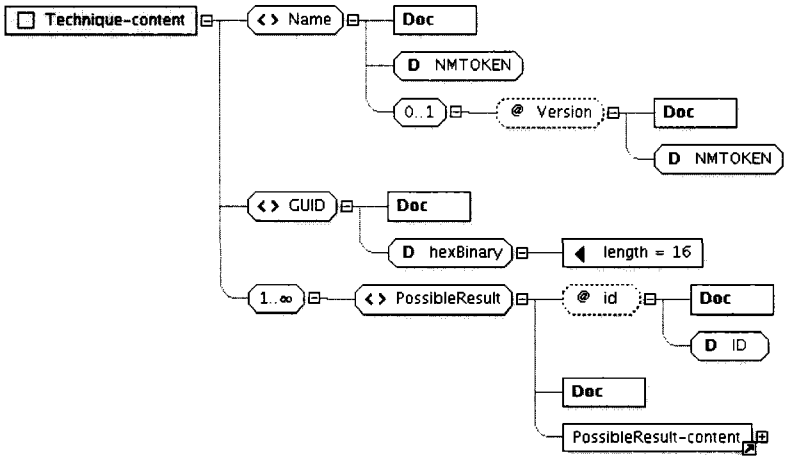


Figure 2-29: Analysis Technique

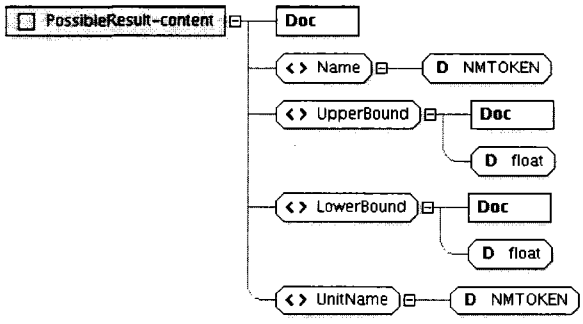


Figure 2-30: Possible Result of Analysis Technique

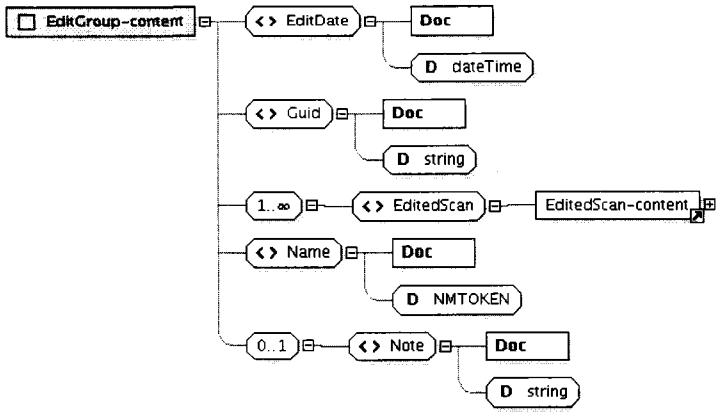


Figure 2-31: Set of Files Analyzed

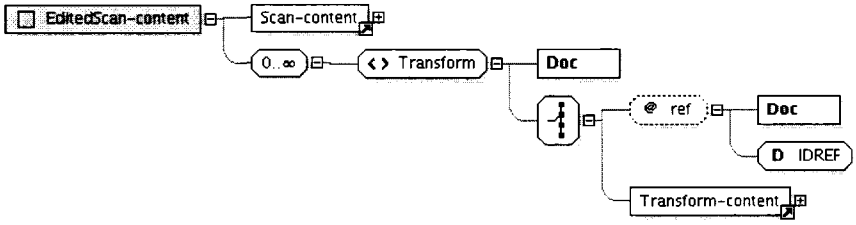


Figure 2-32: Edited Scan

2.5.2 Analyzed files

The set of analyzed files is recorded in the EditGroup sub-tree (Figure 2-31. Each edit group has a name which may be user-specified or generated from the scans included in the group. The EditDate element indicates when this set of files was grouped together for analysis. An edit group must also contain a set set of of edited scans (Figure 2-32. Each EditedScan object is made up of a reference to the base scan (Figure 2-26) and a set of transforms (Figure 2-33) applied to those scans that define the changes to them. The EditGroup may also contain user-specified notes about the set of files.

A transform is classified as either a landmark (meniscus position, beginning of plateau, etc) or a bad point. Simple equations to shift data along the position axis or the signal axis are possible through use of the posTransformEquation and/or valTransformEquation elements. Only simple scaling or shifting is supported. Both types may be defined as either a beginning and ending radial position or a point number. If the transform includes more than one point, an EndPointNumber element

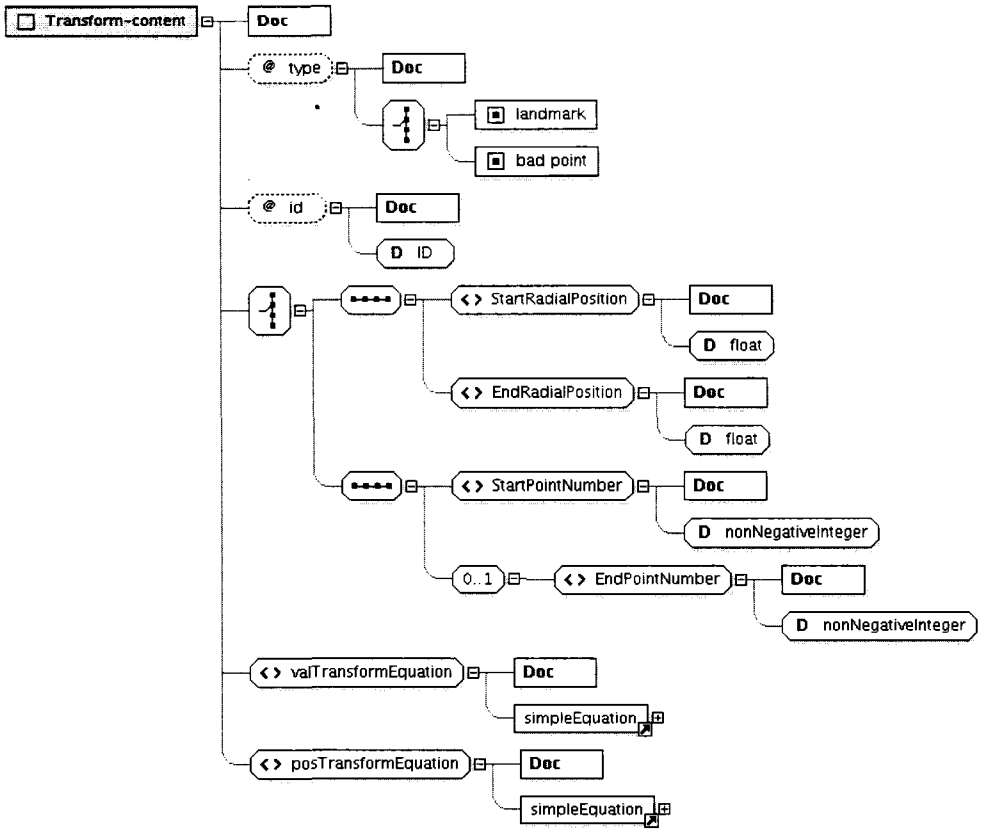


Figure 2-33: File Transform

may be specified. If the transform is specified using radial positions, both beginning and ending positions must always be specified. Point numbers are both more accurate and more precise than radial positions, but radial positions can be applied to sets of scans that may have different numbers of points.

Possible transforms include:

- Inclusion of only a portion of a scan in case a single scan covers multiple channels of data
- Excluding ranges of bad data (due to optical system quirks)
- Marking the position of a meniscus
- Marking the position of a plateau
- Data shifting to correct integer fringe shifts

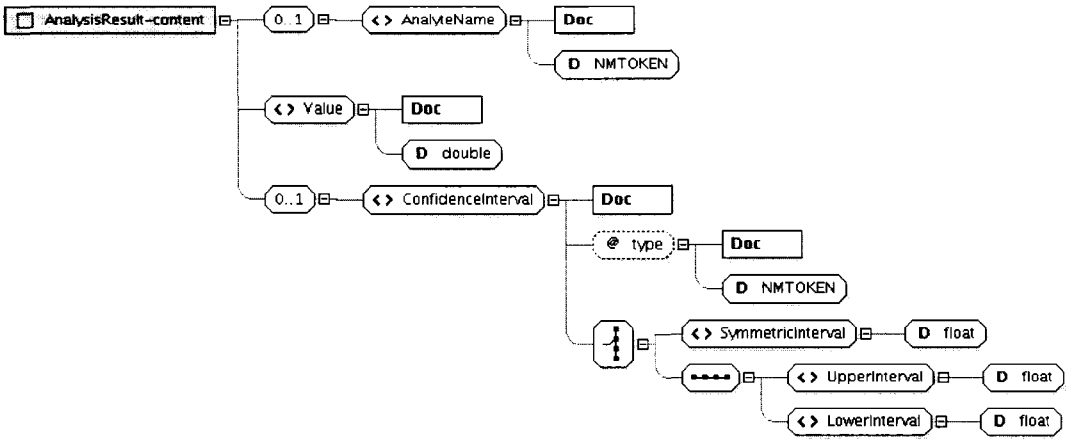


Figure 2-34: Analysis Result

2.5.3 Results

The actual results of analysis are stored in one or more Result sub-trees (Figure 2-34). Each result refers to a possible result from the Technique sub-tree (Figure 2-29). In case there are multiple analytes, an Analyte element is available to allow specification of which analyte is referred to by this result. The value of the result is stored in the Value element as a double precision floating point value. Finally, a confidence interval may be specified either in terms of a single "plus or minus" value or both upper and lower values defining the range for which the specified type of confidence (e.g. 95%) is asserted. These values are expected to be reported as numbers added to or subtracted from the result, rather than upper and lower bounds of the confidence interval.

Conclusion

These schema components define an interface that diverse software may use to interoperate and evolve over time without having to be concerned about implementation details of other software. While it is a necessary prerequisite for useful collaboration, speaking a language is not valuable without a communication partner. A network connected server has been created to be the "listener" in this new communication protocol and is described in Chapter 3.

CHAPTER 3

PROTEIN ASSOCIATION NETWORK DATA SERVER (PANDaS)

3.1 Purpose

PANDaS exists to serve as an interface between the database described in Chapter 4 and client tools. In addition to analysis programs consuming data from PANDaS, the Advanced Operating System (AOS) [29] running on the centrifuge instrument controller is expected to communicate with PANDaS. It must store setup and layout information at the start of a run, and incrementally store scans as they are acquired. PANDaS may run on the instrument controller in a small site setup or could run on a different computer. PANDaS allows client software to remain unaware of the details of database implementation, and even allows for changes to the database layout without requiring modification of all client software. Having PANDaS between clients and the database allows for a flexible authentication system instead of relying on the authentication system built into the underlying database. Should transport layer security [17] be required, Apache [4] may be used in a proxy configuration [1] to perform this function.

3.2 Architecture

PANDaS is written in portable C# [22] using the MindTouch Dream REST library [7]. It has been tested using both Microsoft's .Net framework [33] version 2.0.50727 and Mono version 1.2.6 [34]. All requests are composed of the HTTP1.1 [18] methods, GET, POST, PUT and DELETE. GET requests acquire data from the server. POST requests send new information to the server for processing. PUT requests add or update data on the server. DELETE requests remove data from the server. Use of this RESTful Application Programming Interface (API) allows GET requests to

be cached instead of calculated for each request [19]. Nearly every modern programming language has access to a robust library for HTTP interactions so this API should be accessible to any author of an analysis program.

The following resources are provided by PANDaS:

- Experiments
  - `GET\Experiments\?` retrieves a listing of all the experiments in the database matching the query string (?)
  - `GET\Experiments\{GUID}` retrieves the XML representation of the specified experiment
  - `PUT\Experiments\` creates a new experiment from the passed XML document
  - `PUT\Experiments\{GUID}\Scans` adds the passed scan to the specified experiment
- EditGroups
  - `GET\EditGroups\?` retrieves a listing of all the EditGroups in the database matching the query string
  - `GET\EditGroups\{GUID}` retrieves the XML representation of the specified EditGroup
  - `PUT\EditGroups\` creates a new EditGroup from the passed XML document
  - `GET\EditGroups\{GUID}\Scans\{GUID}` gets the specified EditedScan from this EditGroup
  - `PUT\EditGroups\{GUID}\Scans\` adds the passed EditedScan to the specified EditGroup
  - `GET\EditGroups\{GUID}\Analyses\{GUID}` gets the specified Analysis from this EditGroup
  - `PUT\EditGroups\{GUID}\Analyses\` adds the passed Analysis to the specified EditGroup

- Solutions
  - `GET\Solutions\?` retrieves a listing of all the Solutions in the database matching the query string
  - `GET\Solutions\{GUID}` retrieves the XML representation of the specified Solution
  - `PUT\Solutions\` creates a new Solution from the passed XML document
- Analyses
  - `GET\Analyses\?` retrieves a listing of all the Analyses in the database matching the query string
  - `GET\Analyses\{GUID}` retrieves the XML representation of the specified Analysis
  - `PUT\Analyses\` creates a new Analysis from the passed XML document
- Instruments
  - `GET\Instruments\?` retrieves a listing of all the instruments in the database matching the query string
  - `GET\Instruments\{GUID}` retrieves the XML representation of the specified instrument
  - `PUT\Instruments\` creates a new instrument from the passed XML document

If finer control is needed, more resources could be exposed via this REST API in the future.

3.3 Authentication

After brief experimentation with X.509 [23] public key infrastructure, it was abandoned in favor of a simpler shared secret key system. While this shared secret key system does not allow for cross-site authentication, X.509's cross site abilities are nearly useless because of the unmanageable complexity involved with multiply signed certificates. In future work, support for OpenID authentication will enable cross-site authentication and key retrieval.

When operating in authenticated mode, requests to PANDaS must include an HTTP authorization header [18] with the public access key and a message signature. The message signature is a SHA1 hash [28] of the data being sent, the date it was sent, the URI used in the HTTP request, and the HTTP method used.

The PANDaS access key, along with the private key used by the sending software, is provided to a user when they register with the database and are given permissions. By signing each message sent to the server, the user is proving that they possess the private key matching the PANDaS access key being sent and that the message was not delayed or modified during transmission. To validate the message, PANDaS calculates a new instance of the signature using its copy of the shared secret key and compares it to the client generated signature.

The date is required in the message signature to reduce the opportunity for an attacker to intercept the message, break the hash, change the content of the message, re-hash with the broken key, and resend. Allowing 30 minutes for message transmission should account for any differences between clocks of the sender and PANDaS but not allow enough time for an attacker to break the hash. The date should be in ISO8601 format [24] and converted to UTC time [27] (e.g. YYYY-MM-DDTHH:MM:SSZ). Times are expected to be in 24-hour format.

The URI is required to avoid the possibility for an attacker to redirect a request to an inappropriate location with possibly destructive results. The method name is required to avoid an attacker replacing a GET request with a DELETE request.

Software tools for generating SHA1 hashes are ubiquitous and freely available, but some may find authentication an unnecessary burden. Use of access keys and authorization are only required for access to the non-public data in a database, so participating software developers may skip this step and still provide a useful mechanism for interaction with PANDaS.

The message signature must be calculated using the pseudocode algorithm in listing 3.1. Any differences in implementation between PANDaS and the client will cause PANDaS to be unable to validate the message. Authorization occurs after a user has been authenticated and takes place in the PANDaS Database Library described in section 3.4.

```
1        secretkey  = ' dajdkfa17987ualdfjay79a847iuj ';
2        messagehash = sha1.sign(messagepayload, secretkey );
3        date  = '2008-01-01T13:01:00Z';
4        URI = ' http :// server .domain.tld/pandas/' ;
5        method = 'GET';
6        signature  = sha1.sign(date  .  method .  URI .  messagehash,  secretkey );
7        pandasAccessKey = 'abd123456789';
8        AuthHeader = 'PANDaS' . '␣' .  pandasAccessKey .  ':'  .  signature ;
```

Listing 3.1: Perlish Pseudocode for Authentication Header Generation

3.4 Database Interaction Library

A high quality library for interacting with the database described in Chapter 4 has been created. This library exists to read and write information from the database into in-memory structures for access by PANDaS (section 3.1) or other local programs. It also handles reading from and writing to the XML formats defined in Chapter 2.

3.4.1 Architecture

The database library consists of a single base class, AUFDB, providing overridable functions for the derived class. As development continues, the AUFDB class may be converted from a base class to an interface providing class to avoid binding these objects too tightly to their database functionality. The AUFDB class implements the details of constructing database connections and executing stored procedures. Each class must override three methods to provide the class-specific logic. The `setVariables` method (Listing 3.2) takes an `IDataReader` object containing the unparsed results of a stored procedure call as its only argument and extracts the values from that data set into class variables.

```
1  /// <summary>
2  /// Sets the variables for the cell class as they are read from the database.
3  /// </summary>
4  /// <param name="rdr">The data reader.</param>
5  protected override void setVariables (IDataReader rdr)
6  {
7          _expID = (int)rdr["experimentid"];
8
9          if (_exp != null && _exp.key != _expID)
10         {
11         throw new Exception("DB experiment id does not match cell's exp id. " +
12                                 _exp.key.ToString() + "!=" + _expID.ToString());
13         }
14
15         _serialNumber = (string)rdr["centerpieceserialNumber"];
16         _holeNumber = (int)rdr["holenumber"];
17         _name = (string)rdr["cellname"];
18         _windowType = (WindowType)Enum.Parse(typeof(WindowType),
19                         (string)rdr["windowtype"]);
20         _absCenterpiece = new AbstractCenterpiece((int)rdr["abscenterpieceid"],
21                                                   ConnectionString);
22
23         do
24         {
25         AbstractChannel absc =
26                 this.AbstractCenterpiece.ChannelWithKey((int)rdr["abstractChannelid"]);
27
28         Channel c = new Channel(
29                 (int)rdr["channelid"],
30                 absc, new Solution((int)rdr["solutionid"], ConnectionString),
31                 this, new Guid((string)rdr["channelguid"]),
32                 (string)rdr["channelcomment"],
33                 (DateTime)rdr["channelDateupdated"], ConnectionString);
34         _channels.Add(c);
35         } while (rdr.Read());
36 }
```

Listing 3.2: setVariables Method for Cell Object

The `addPkParams` method (Listing 3.3) adds whatever parameters are necessary to define a unique record in the database for a given class.

```
1  /// <summary>
2  /// Adds the primary key parameters required to identify a unique cell object.
3  /// </summary>
4  /// <param name="cmd">The command.</param>
5  protected override void addPKParams(IDbCommand cmd)
6  {
7          addParam(cmd, "v_cellid", DbType.Int32, _key);
8  }
```

Listing 3.3: addPkParams Method for Cell Object

`addInsertParams` (Listing 3.4) adds the stored procedure parameters and sets their values during an insert or update query.

```
1  /// <summary>
2  /// Adds the insert parameters required to insert a new cell object to the database.
3  /// </summary>
4  /// <param name="cmd">The command.</param>
5  protected override void addInsertParams(IDbCommand cmd)
6  {
7          addParam(cmd, "v_abscenterpieceid", DbType.Int32, _absCenterpiece.key);
8          addParam(cmd, "v_experimentid", DbType.Int32, _expID);
9          addParam(cmd, "v_cellName", DbType.String,
10                 _name.Substring(0, _name.Length > 100 ? 100 : _name.Length));
11         addParam(cmd, "v_windowtypename", DbType.String, _windowType.ToString());
12         addParam(cmd, "v_holenumber", DbType.Int32, _holeNumber);
13         addParam(cmd, "v_centerpieceserialnumber", DbType.String, _serialNumber);
14 }
```

Listing 3.4: addInsertParams Method for Cell Object

Classes may override the dbWrite(conn) method from the superclass if they need to cause referenced classes to be written before writing themselves. For example, the solution class must call dbWrite on abstract solution and component objects before executing the stored procedure to write a solution record. This architecture has the advantage of allowing all database interaction to flow through a single method and avoids duplicate code in each class for writing to the database. The AUFDB class also provides equality (==) and inequality (!=) operators for comparing database objects.

3.4.2 Testing

The library has been tested using both unit tests and integration tests [20] with an emphasis on integration testing. Unit tests exercise individual methods, while integration tests validate a sequence of interacting methods. At the time of this writing 357 tests are required to exercise 97% of the code in the library. The remaining 3% of untested lines is mostly error checking logic that is difficult to test. Each class has a dedicated test class containing methods to test all aspects of the targeted class. A test harness instantiates a test method, which in turn sets up the object to be tested. Each test method targets a specific code path to exercise. As the test method makes calls into the target, method assertions about the correct state of the target object are made. A test fails upon the first assertion failure, and a message is logged before the next test method is run.

Code coverage is measured using the NCover program [46]. It acts as a test harness keeping track of access to every line of source code. NCover calculates code coverage on a class by class basis and highlights those lines that were not executed during the test run.

The test method in listing 3.5 is a typical simple unit test. It tests the creation of solutions by adding 1 gram of a water and 1 milligram of BSA to a new Solution object. It then writes the Solution object to the database at line 18. At line 20, a second Solution object is constructed using the key generated for the first object during the write to the database. Finally, the two objects are passed to the`fieldsMatch` method at line 21 for comparison. The target object is not instantiated manually in every test method, instead many test methods call a setup method to construct a boilerplate target object.

```
1        [Test]
2        public void addComponentsByWeight()
3        {
4
5            Solution s = new Solution(connstr);
6
7            s.Name = "bsa in water deleteme";
8            Solvent c = new Solvent(connstr);
9            c.Name = "water deleteme";
10           c.IsSolvent = true;
11           c.InChi = new InChILib.InChI("1/H2O/h1H2");
12           c.CanonicalMw = 18;
13           s.AddSolvent(c,1,Unit.MassUnit.gram);
14           Solute c2 = new Solute(connstr);
15           c2.Name = "BSA deleteme";
16           c2.CanonicalMw = 65000;
17           s.AddSolute(c2,1,Unit.MassUnit.milligram);
18           s.dbWrite();
19
20           Solution s2 = new Solution(s.key, connstr);
21           fieldsMatch(s, s2);
22       }
```

Listing 3.5: Simple Test Method

In the `fieldsMatch` method (Listing 3.6) all the properties of the two solution objects are compared to be certain that reading and writing from the database results in no loss of detail. Lines 17-19 call a generic list comparison method that matches up items and calls the ComponentTest fieldsMatch method to compare each component. If any of these assertions fail, the third parameter is printed so it is easy to understand what caused the failure.

```
1  internal static void fieldsMatch(Solution s, Solution s2)
2  {
3      Assert.AreEqual(s.key, s2.key, "key match");
4      Assert.AreEqual(s. LotIdentifier , s2. LotIdentifier , "LotID Match");
5      Assert.Less(s.DateCreated.Ticks - s2.DateCreated.Ticks, 100, "DateCreated match");
6      Assert.Less(s.DateExpired.Ticks - s2.DateExpired.Ticks, 100, "DateExpired match");
7      Assert.AreEqual(s.Name, s2.Name, "Name match");
8      Assert.AreEqual(s.Guid.ToString(), s2.Guid.ToString(), "guid");
9      RefSourceTest.fieldsMatch(s.RefSource, s2.RefSource);
10     Assert.AreEqual(s.SolventComponents.Count,
11                         s2.SolventComponents.Count, "solvent component number");
12     Assert.AreEqual(s.SoluteComponents.Count,
13                         s2.SoluteComponents.Count, "solute component number");
14     Assert.AreEqual(s.AnalyteComponents.Count,
15                         s2.AnalyteComponents.Count, "analyte component number");
16
17     ListCompare<Solute>(s.SoluteComponents, s2.SoluteComponents);
18     ListCompare<Analyte>(s.AnalyteComponents, s2.AnalyteComponents);
19     ListCompare<Solvent>(s.SolventComponents, s2.SolventComponents);
20 }
```

Listing 3.6: Solution Fields Comparison

Conclusion

PANDaS is a fairly thin layer of software that makes extensive use of a large data access library to store and retrieve information from the database described in Chapter 4.

CHAPTER 4

DATABASE DESIGN

A database to store experiment results, experiment setup parameters, and results of analysis has been created. It was implemented using the PostgreSQL database engine [41] but has been designed so that it can easily be moved to any modern database management system. All database interaction takes place via an exhaustive set of "stored procedures". For example, to read a solution and relevant components from the database instead of executing SQL directly, the`Solution_FindSingle(solutionid)` function is called. This function performs the necessary joins to return a denormalized dataset that can be parsed to generate the entire solution with one database "roundtrip".

These procedures follow a naming convention of`logicalobject_function` where`logicalobject` might be a solution, an experiment, an instrument, etc., and`function` might may be Insert, FindAll, FindSingle, FindFor*SharedProperty*, Delete, or Update. FindAll suffixes indicate that every logical object should be returned. FindFor... suffixes signify that the set of logical objects sharing the property indicated in the FindFor... suffix should be returned. Delete and Update suffixes remove or modify existing database objects. Insert suffixes add data to the database and return the generated primary key value. In some cases they will catch an attempt to add a duplicate record, returning the primary key value of an existing record. Diagrams of the tables in this database have been divided into functional areas for easier readability. Some tables are represented in multiple diagrams for clarity.

Use of stored procedures is a hallmark of good database design. The database engine is able to re-use an optimal query plan for stored procedures instead of having to calculated one every time an ad-hoc sql statement is executed. In addition, through the use of stored procedures it's possible to

avoid most "sql injection" attacks in which a hostile user attempts to inject destructive commands to user. Stored procedures also allow for easier migration between versions and vendors of databases. During a transition, instead of finding and updating embedded sql in all programs that access the database, only the known, discrite set of stored procedures must be updated.

This database is "normalized" to to avoid duplication of information [13]. Duplication can lead to unresolvable conflicts between tables, so it should be avoided even if tables with only a few records must be created. In addition to a normalizing data to the maximum extent possible, a well designed database will remain usable even as data changes over time. For example, it is possible to store the polynomial $2x + 5.2x^2 + 6x^3$ in a single "poly" table with columns "polyid", "term1", "term2", and "term3". A two-table design might contain a "poly" table with only an "polyid" field and a second "term" table with "polyid","value","degree" fields. Clearly the two table design provides superior flexibility in the event that more terms are needed. The additional complexity is only worthwhile if the polynomial's terms must be examined in isolation. If the polynomial is always referred to in its entirety it may be better to store its string representation instead.

4.1 Experiment Results

The experimental results set of tables includes information relevant to storing or retrieving acquired experimental data (Figure 4-1). This set of tables also stores groupings of scans for later analysis. EditGroups may be formed from any set of edited scans, even if they are not from the same experiment.

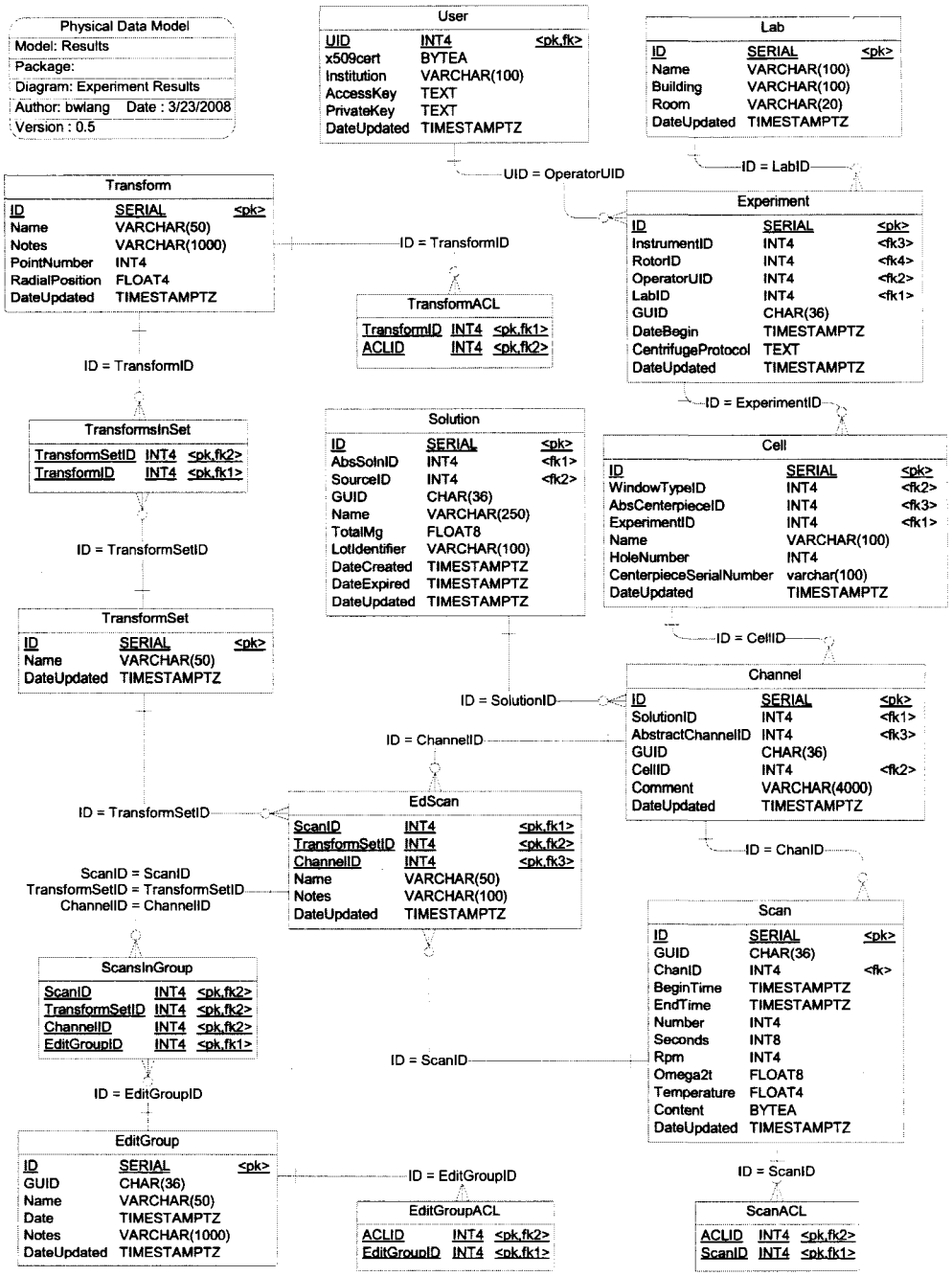


Figure 4-1: Experiment Results Tables

4.2 Hardware

The experimental hardware tables (Figure 4-2) contain tables referring the equipment used in a centrifuge experiment and the "Abstract" companion tables to some physical objects. Abstract objects refer to the concept or type of the physical analog. An abstract rotor contains all the information about a rotor that is common to all rotors of the same type (e.g. # of holes, maximum speed rating, etc). The physical rotor holds information specific to a particular rotor (e.g. serial number, name).

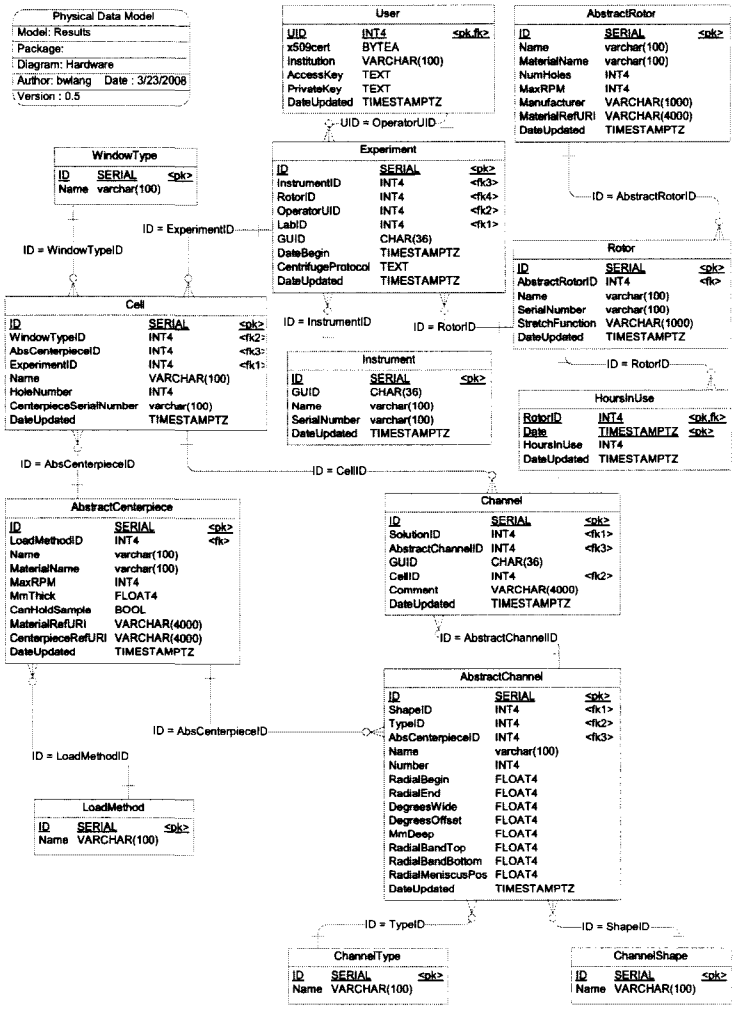


Figure 4-2: Experiment Hardware Tables

4.3 Analysis Results

Information about analysis techniques and results of actual analyses are stored in a set of four dependent tables (Figure 4-3). The Technique table contains a list of techniques used for analysis. It should contain one record for each version of each analysis program used. The list of possible results produced by an analysis technique are stored in the PossibleResult table. The Analysis table stores one record for each technique applied to a group of files. Access is to analyses is controlled at the level of individual analyses through the AnalysisACL table.

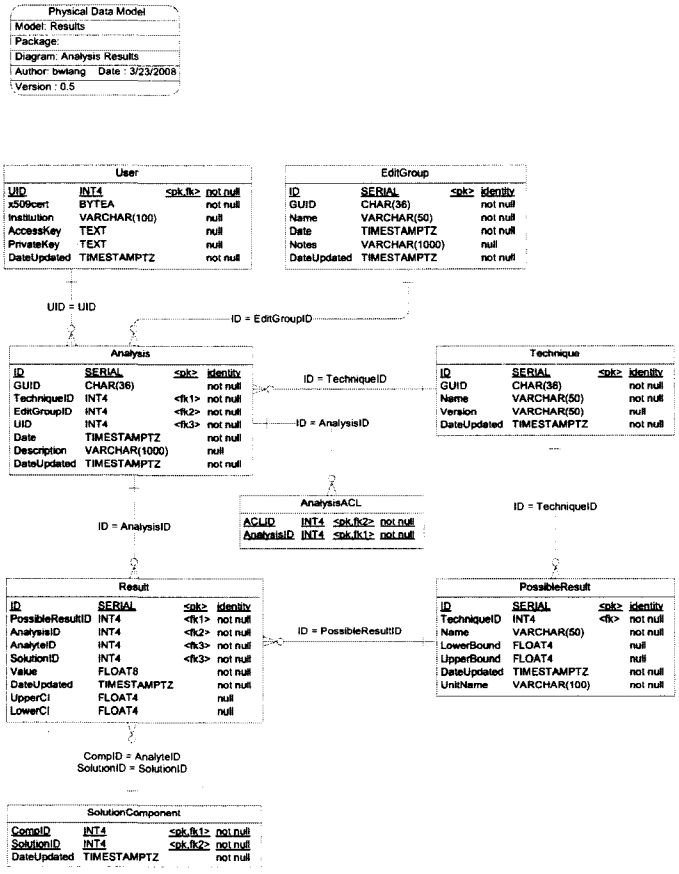


Figure 4-3: Analysis Results Tables

4.4 Optical System Details

Known optical system details are stored in tables specific to each optical system (Figure 4-4). Each table is related to the OpticalSystem table which stores the duration that optical system was active. These settings are stored once for each optical system record. In addition to the fixed information, it is possible to store channel-specific settings like PGA gains in the OpSysSetting table. Unanticipated settings could also be stored in the OpSysSetting table at the cost of some data duplication if those settings are not specific to a single channel.

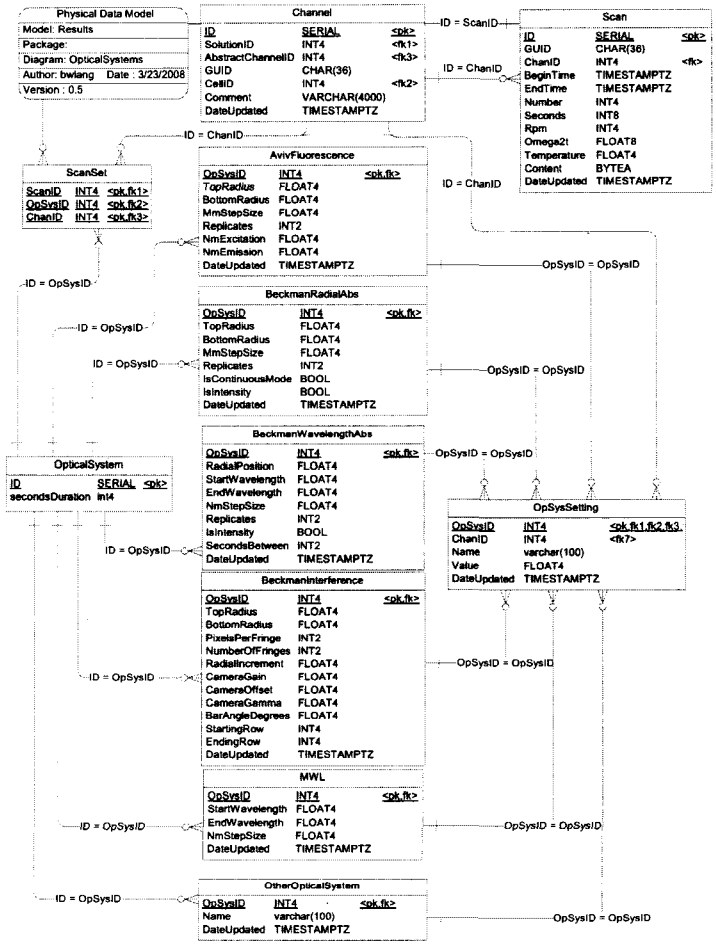


Figure 4-4: Optical System Tables

4.5 Solution Components

Details of the construction of solutions are stored in the tables displayed in Figure 4-5. Abstract components represent all the components that might be mixed into a solution. Solvent components are distinguished from solute components through the use of the boolean IsSolvent flag. Each abstract component is identified by a GUID so they may be shared across PANDaS databases. Abstract components that are small molecules are also represented by InChI strings [40], which uniquely identify molecules so they may be linked to other types of databases. Abstract solutions represent a generic solution as a set of components in defined quantities (e.g. 200mM KCl) using the AbsSolutionAbsComponent table. Relative amounts of the various components are stored as a fraction of the total mass in a solution. This construct enables accurate representation of the relative amounts of components across a broad range of concentrations (pico-Molar to deci-Molar) and avoids ambiguity caused by use of weight/volume concentrations. The status of a molecule as an analyte is stored in the AbsSolutionAbsComponent table since a given molecule may be an analyte in one experiment, but only a solvent component in another. People are not expected to construct solutions using weight fractions, instead the database library (section 3.4) includes conversions from weight/volume, molar, and total mass added systems to weight fractions.

Solutions are represented in the database as sets of components. Components may store batch specific properties in the Component table, while the solution batch as a whole is stored in the Solution table. The Solution table may store several batches of an abstract solution. Solutions must specify a total mass so calculations of actual amounts of each component can be calculated from their weight fractions. Components and solutions may be set to expire on a specified date so they can be hidden from the user interface when no longer relevant.

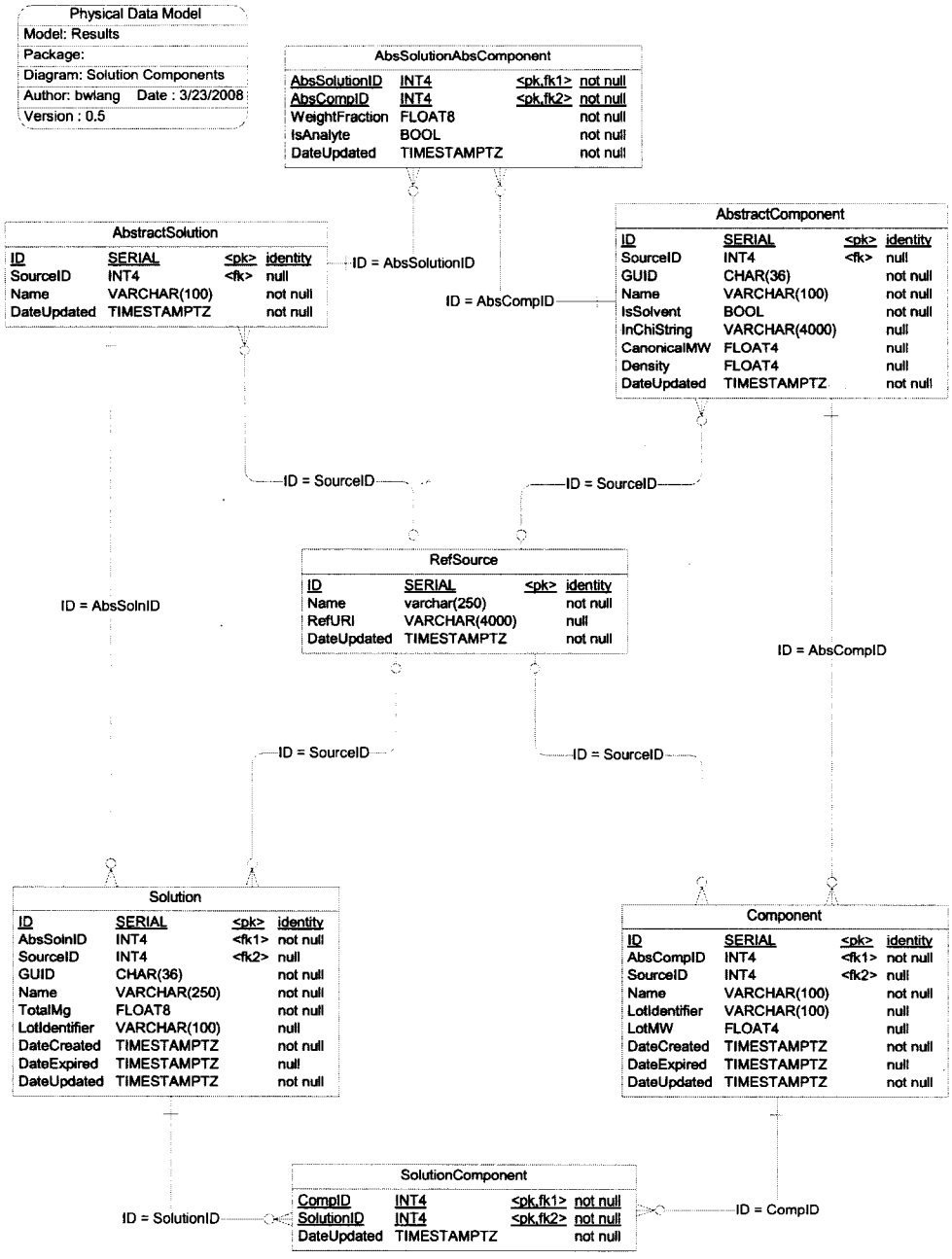


Figure 4-5: Solution Tables

CHAPTER 5

SOFTWARE TOOLS

5.1 Sucker

The Sucker was created to suck data from from the filesystem into the database. When the Sucker starts, it prompts the user to locate the base directory structure containing the data files. It is able to import data produced by the AOS instrument control software and from Beckman's directory layout. The FileTree class library (Figure D-1) is used to interpret the number of cells and which optical systems were used. It also reads the header information from representative scans to identify the optical system parameters used to acquire them.

While the file tree is being scanned, the user may specify the lab, instrument, operator, and rotor using the dropdown lists on the main screen (Figure 5-1) Once the file tree has been completely scanned, the user is asked to select which group of scans to import.

After specification of the scan group, an entry area for each cell in the experiment is created in which the user is expected to skip or specify the type of centerpiece and contents of the channels. The channel dialog (Figure 5-2) contains a "Set Common Solution" button to specify a common solution for all channels. The common solution may then be modified by clicking on each individual channel's "Choose Solution" button. If a solution has been set, a string representing its components is displayed. Colored boxes are drawn to indicate which solutions are the same and which are different.

When a user chooses a solution button, a dialog appears (Figure 5-3) to allow selection of an existing solution or definition of a new one. Solutions that are in use in other experiments may not be modified, but may be reused by selecting the "Choose Existing Batch..." button. The "Fill from

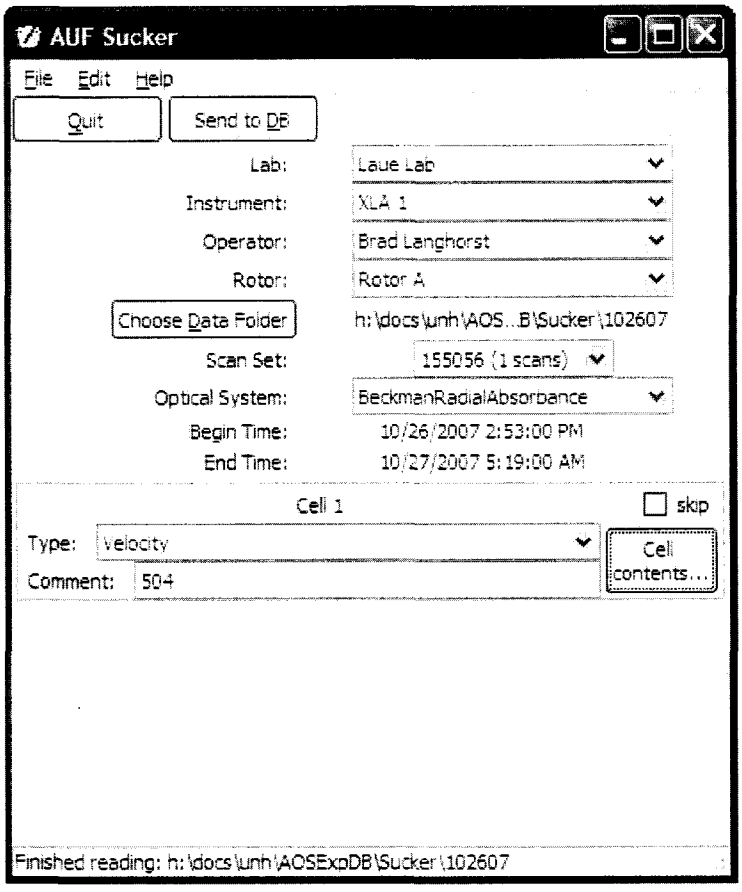


Figure 5-1: Sucker Main Screen

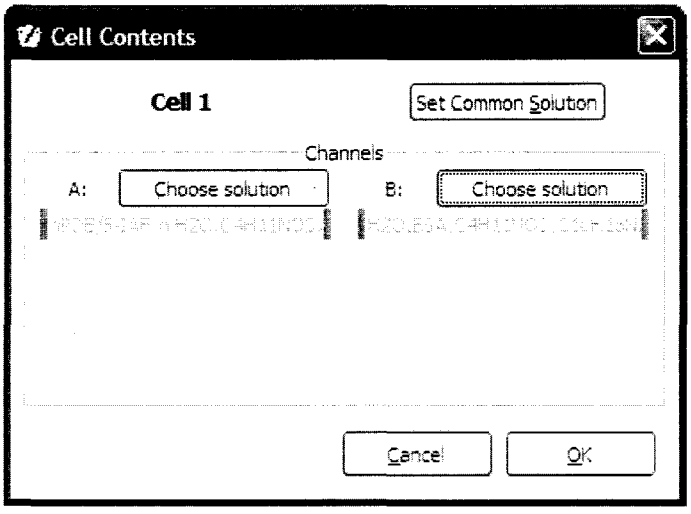


Figure 5-2: Channel Dialog

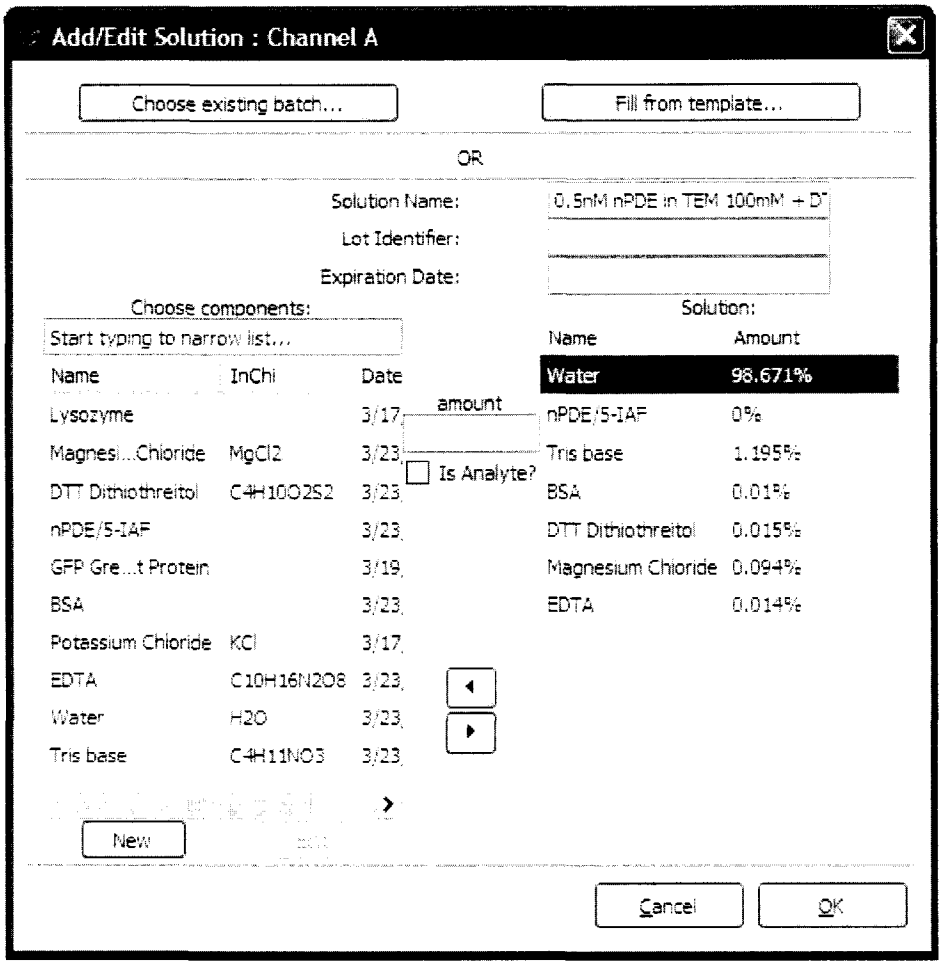


Figure 5-3: Solution Construction Dialog

template..." button allows the user to construct a new solution based on an existing one. The user may select a component from the list of existing components on the left side of the screen. It is possible to type a partial chemical formula or English name of a component above the list of components to narrow the list for easier selection. If the component does not exist, the user may choose the Add button at the bottom of the screen. The Edit button is available if a component has not yet been used in a solution. Before adding a selected component to the solution, the user must specify how much of that component to add (in mass or concentration units) and specify whether the component should be considered the analyte in this solution.

Once the user has specified the solution contents for every channel in the experiment, it is ready

for export. Choosing the "Add to database" button causes the system to read in and compress every scan using bzip2 [8] as it adds the experiment and all scans to the database. The same experiment object is then used to serialize its structure to an the XML format described in section 2.3, which is suitable for archiving or sharing with a colleague.

The Sucker was written in C# [22] using the GTK# windowing toolkit [26] [32] in a platform independent manner, so it can be run in Microsoft Windows, Apple's MacOS, and X11.

5.2 Database Administration Tool

A database administration tool was created to allow creation of users and groups and assignment of permissions on various database objects (Figure 5-4). At the time of this writing, it has only a minimal feature set. It will be extended in a future revision to allow management of solutions in addition to users and permissions.

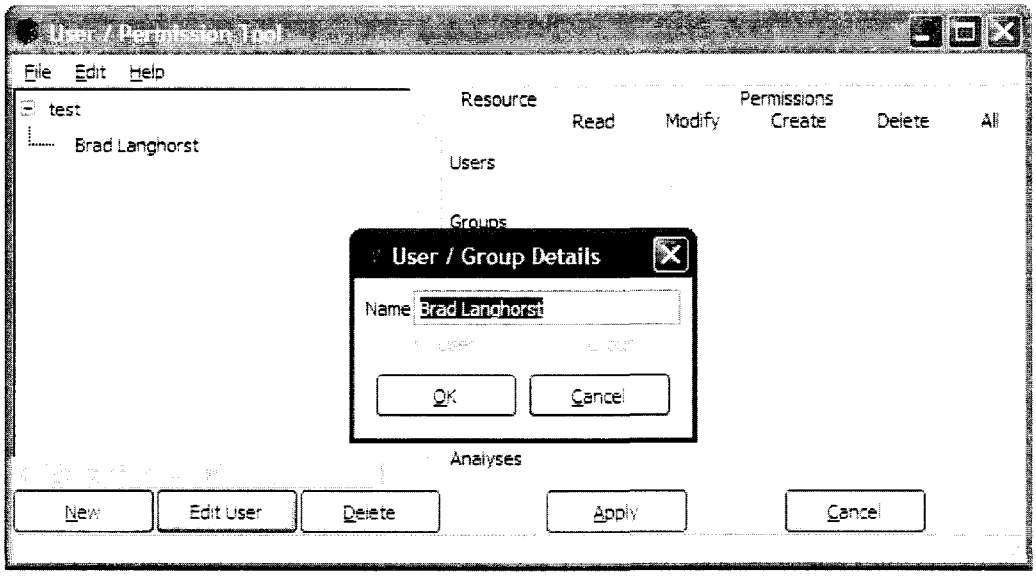
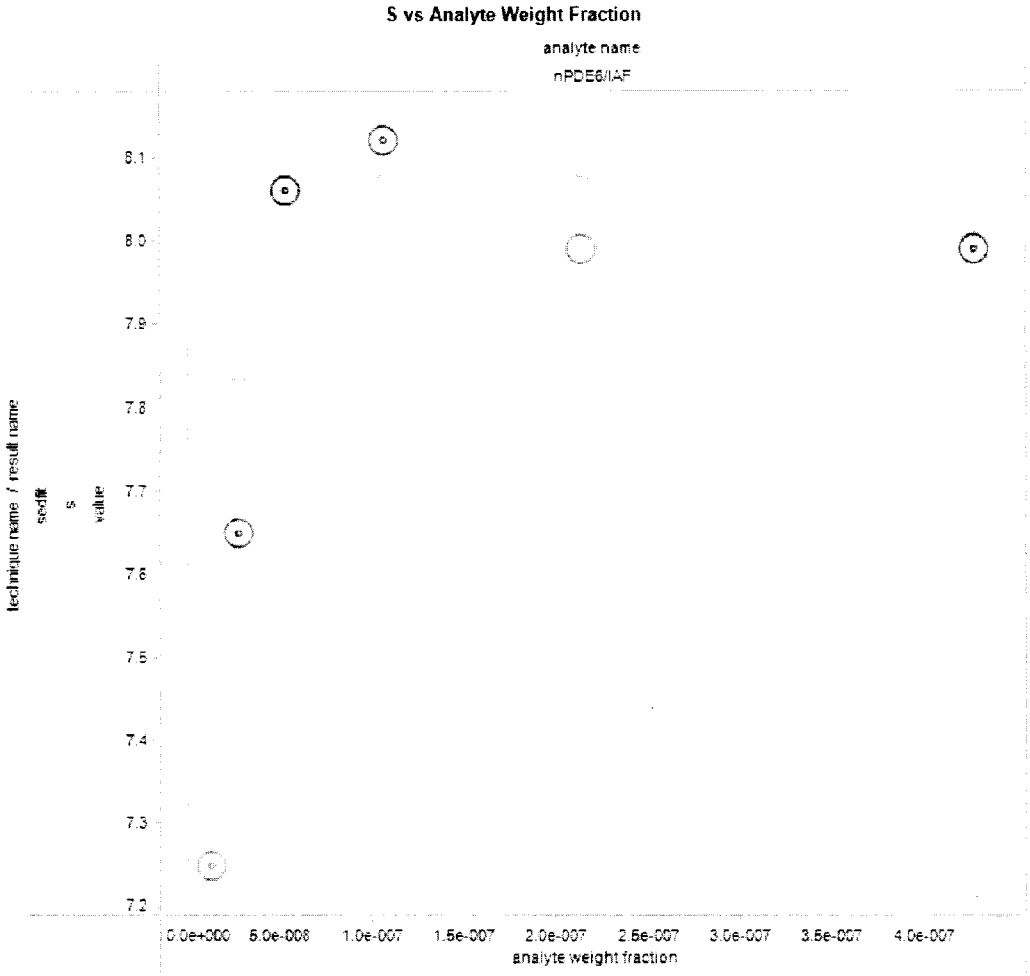


Figure 5-4: Database Administration Utility

5.3 Meta-analysis Tool

A commercial meta-analysis tool, Tableau [31], has been employed as a visualization client for this data warehouse. Using Tableau, it is possible to quickly search for patterns in the results of analysis of multiple experiments and techniques. As an example of this approach, a set of experiments on a photoreceptor cyclic nucleotide phophodiesterase (PDE), was examined. PDE/6 is present as a heterodimer of α and β [3] associated with 2 inhibitory γ subunits [2] to form a tetrameric, inactive enzyme. During visual transduction, the enzyme is activated when the γ subunits are displaced. A representative graph from Tableau shows the results of an experiment on PDE/6 over a range of analyte concentrations. The analytical ultracentrifuge allows a detailed examiniation of subunit association as manifested by changes in molecular size under various solution conditions. The graph (Figure 5-5) conveys a change in PDE's subunit association as its concentration in the solution varies. It is also clear from this graph that presence or absence of GFP does not meaningfully affect the s value. As more experiments and analyses are performed on an analyte, this method of meta-analysis provides a quick way to identify which variables are most important to analyte behavior. As with any tool that allows for rapid testing of multiple hypotheses, a user must be mindful of the possibility of false discoveries [5].

Full source code for these tools is available at http://lauelab.unh.edu/projects/PANDaS



Analyte weight fraction vs. value broken down by analyte name vs. technique name and result name for nPDE6/IAF. Color shows details about name. Size shows details about component_name. Details are shown for name and editgroup_name. The data is filtered on issolvent, which keeps False. The view is filtered on component_name and analyte name. The component_name filter keeps GFP Green Fluorescent Protein, nPDE/6-IAF and nPDE6/IAF. The analyte name filter keeps nPDE6/IAF.

component_name

- GFP Green Fluorescent Prote..
- nPDE6/IAF

name

- 0.06nM nPDE in TEM 100mM + DTT + BSA + GFP
- 0.06nM nPDE in TEM 100mM + DTT +BSA
- 0.13nM nPDE in TEM 100mM + DTT + BSA + GFP
- 0.13nM nPDE in TEM 100mM + DTT +BSA
- 0.25nM nPDE in TEM 100mM + DTT + BSA + GFP
- 0.25nM nPDE in TEM 100mM + DTT +BSA
- 0.5nM nPDE in TEM 100mM + DTT + BSA + GFP
- 0.5nM nPDE in TEM 100mM + DTT +BSA
- 1nM nPDE in TEM 100mM + DTT + BSA + GFP
- 1nM nPDE in TEM 100mM + DTT +BSA
- 2nM NPDE in TEM 100mM + DTT + BSA + GFP
- 2nM nPDE in TEM 100mM + DTT + BSA

Figure 5-5: Tableau Meta-data Analysis Software

Conclusion

Detailed studies of molecular interactions by Analytical Ultracentrifugation are inherently time and effort intensive experiments. To increase our rate of knowledge acquisition, we should automate as many manual tasks as possible. Our ability to implement automation is hampered by the lack of a predictable, computer-readable record of what is being studied and the methods used in the study. All relevant details about what is being studied should be specified at the time of experiment setup and recorded with the chosen instrument parameters. Having this information allows for the construction of a loosely coupled network of software and human collaborators. Further, storing experimental detail will decrease the occurrence of human errors and minimize duplicated effort. Wide adoption of this XML-based data interchange format is an important step toward increasing our rate of understanding of macromolecular interactions.

BIBLIOGRAPHY

[1] Apache Developers. Apache module mod_proxy. [Online; accessed 22-May-2008].

[2] Nikolai O. Artemyev, Rajendran Surendran, James C. Lee, and Heidi E. Hamm. Subunit Structure of Rod cGMP-Phosphodiesterase. *J. Biol. Chem.*, 271(41):25382–25388, 1996.

[3] W Baehr, EA Morita, RJ Swanson, and ML Applebury. Characterization of bovine rod outer segment G-protein. *J. Biol. Chem.*, 257(11):6452–6460, 1982.

[4] Brian Behlendorf, Roy T. Fielding, Rob Hartill, David Robinson, Cliff Skolnick, Randy Terbush, Robert S. Thau, and Andrew Wilson. The apache http server project. [Online; accessed 28-May-2008].

[5] Y Benjamini and Y Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B*, 57:125–133, 1995.

[6] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738, Internet Engineering Task Force, December 1994. http://ds.internic.net/rfc/rfc1738.txt.

[7] Steve Bjorg. Dream reference documentation, 03 2008. [Online; accessed 22-May-2008].

[8] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, DIGITAL Systems Research Center, 1994.

[9] James Clark. Multi-format schema converter based on RELAX NG. http://www.thaiopensource.com/relaxng/trang.html.

[10] James Clark. XSL transformations (XSLT) version 1.0. W3C recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[11] James Clark, Ken Holman, and Martin Bryan. iso 19757-3 Information technology – Document Schema Definition Language (DSDL) – Part 3: Rule-based validation – Schematron. Technical Report 19757-3, International Organization for Standardization, June 2006.

[12] James Clark and MURATA Makoto. RELAX NG Specification. Technical report, 2001. http://www.oasis-open.org/committees/relax-ng/spec.html.

[13] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Comm. ACM*, 13(6):377 – 387, 1970.

[14] Jim Cole. Analysis of heterogeneous interactions. *Methods Enzymol.*, 384:212–232, 2004.

[15] B. Demeler, H. Saber, and J. C. Hansen. Identification and interpretation of complexity in sedimentation velocity boundaries. *Biophys. J .*, 72(1):397 – 407, 1997.

[16] Borries Demeler. *Analytical Ultracentrifugation: Techniques And Methods*, chapter Ultrascan - A comprehensive Data Analysis Software Package for Analytical Ultracentrifugation Experiments, pages 210–230. Royal Society of Chemistry, 2005.

[17] T. Dierks and C. Allen. The TLS Protocol Version 1.0. Technical Report 2246, Internet Engineering Task Force, January 1999. Obsoleted by RFC 4346, updated by RFC 3546.

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical report, United States, 1999.

[19] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[20] Erich Gamma and Kent Beck. Test infected: Programmers love writing tests. *Java Report*, 3(7):37–50, July 1998.

[21] David B Hayes, Thomas M. Laue, and John Philo. Sednterp, 1995.

[22] Anders Hejlsberg, Scott Wiltamuth, and Peter Golde. *C# Language Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[23] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280, Internet Engineering Task Force, April 2002. http://www.ietf.org/rfc/rfc3280.txt.

[24] International Organization for Standardization. ISO 8601:2000. Data elements and interchange formats — Information interchange — Representation of dates and times. Technical report, 2000.

[25] M.L. Johnson, J. J. Correia, D. A. Yphantis, and H. R. Halvorson. Analysis of Data from the Analytical Ultracentrifuge by Nonlinear Least Squares Techniques. *Biophys. J .*, 36:575 – 588, 1981.

[26] Mike Kestner, Duncan Mak, Miguel de Icaza, John Luke, Marques Johansson, Iain McCoy, Eric Butler, Jamin Philip Gray, Chris Turchin, Jasper van Putten, and Néstor Salceda. Gtk#, a gui toolkit, is a set of .net bindings for the gtk+ toolkit., September 2001. [Online; accessed 26-May-2008].

[27] G. Klyne and C. Newman. Date and Time on the Internet: Timestamps. Technical Report 3339, Internet Engineering Task Force, July 2002.

[28] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Technical report, United States, 1997.

[29] Tom Laue and Brett Austin. AOS Advanced instrument Operating System. [Online; accessed 22-May-2008].

[30] P. Leach, M. Mealling, and R. Salz. A universally unique identifier (uuid) urn namespace. RFC 4122 (Proposed Standard), July 2005. http://www.ietf.org/rfc/rfc4122.txt.

[31] D. Macinlay, Jock, Christopher Stolte, and Patrick Hanrahan. Computer systems and methods for automatically viewing multidimensional databases, 2007. Patent WO/2007/030817.

[32] Peter Mattis, Spencer Kimball, Josh MacDonald, Owen Taylor, Matthias Clasen, Federico Mena Quintero, Soeren Sandmann, Padraig O'Briain, Manish Singh, Kristian Rietveld,

and Tor Lillqvist. Gtk+-gnu toolkit for x windows development. [Online; accessed 26-May-2008].

[33] Microsoft Corporation. .net framework, 2006. [Online; accessed 22-May-2008].

[34] Mono Developers. Mono, 2007. [Online; accessed 22-May-2008].

[35] J. S. Philo. A method for directly fitting the time derivative of sedimentation velocity data and an alternative algorithm for calculating sedimentation coefficient distribution functions. *Anal. Biochem.*, 279(2):151 – 163, 2000.

[36] P. Schuck. Size-Distribution Analysis of Macromolecules by Sedimentation Velocity Ultracentrifugation and Lamm Equation Modeling. *Biophys. J .*, 78(3):1606 – 1619, 2000.

[37] Jérôme Siméon, Scott Boag, Jonathan Robie, Don Chamberlin, Daniela Florescu, and Mary F. Fernández. XQuery 1.0: An XML query language. W3C recommendation, W3C, January 2007. http://www.w3.org/TR/2007/REC-xquery-20070123/.

[38] C. M. Sperberg-McQueen, Eve Maler, Jean Paoli, François Yergeau, and Tim Bray. Extensible markup language (XML) 1.0 (fourth edition). W3C recommendation, W3C, August 2006. http://www.w3.org/TR/2006/REC-xml-20060816.

[39] W. F. Stafford and P. J. Sherwood. Analysis of Heterologous Interacting Systems by Sedimentation Velocity: Curve Fitting Algorithms for Estimation of Sedimentation Coefficients, Equilibrium and Kinetic constants. *Biophys. Chem.*, 108(1-3):231 – 243, 2004.

[40] S. E. Stein, S. R. Heller, and Tchekhovskoi, editors. *The IUPAC Chemical Identifier*, Columbus, Ohio, USA, 1 July 2002. CAS/IUPAC Conference on Chemical Identifiers and XML for Chemistry.

[41] M. Stonebraker and L. A. Rowe. The Design of Postgres. In C. Zaniolo, editor, *SIGMOD*, pages 340 – 355. ACM, may 1986.

[42] The Svedberg. Molecular Weight Analysis in Centrifugal Fields. *Science*, 79(2050):327–332, 1934.

[43] The Svedberg and Robin Fahraeus. A New Method for the Determination of the Molecular Weight of the Proteins. *J. Am. Chem. Soc.*, 48:430–438, 1926. canonical equilibrium centrifugation reference.

[44] The Svedberg and J Nichols. Determination of Size and Distribution of Size of Particle by Centrifugal Methods. *J. Amer. Chem. Soc.*, 45(2):2910–2917, 1923.

[45] Henry S. Thompson, Murray Maloney, David Beech, and Noah Mendelsohn. XML schema part 1: Structures second edition. W3C recommendation, W3C, October 2004. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[46] Peter Waldschmidt. Ncover.

[47] David A Yphantis. Equilibrium Ultracentrifugation of Dilute Solutions. *Biochemistry*, 3(3):297–317, 1964.

APPENDICES

APPENDIX A

OPTICAL SYSTEMS

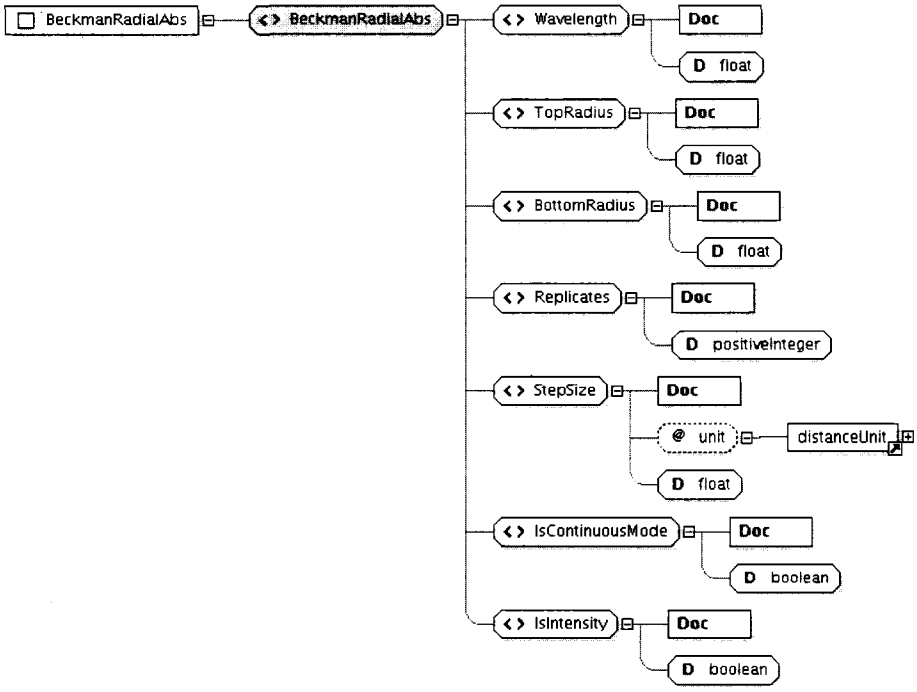


Figure A-1: Beckman Radial Absorbance

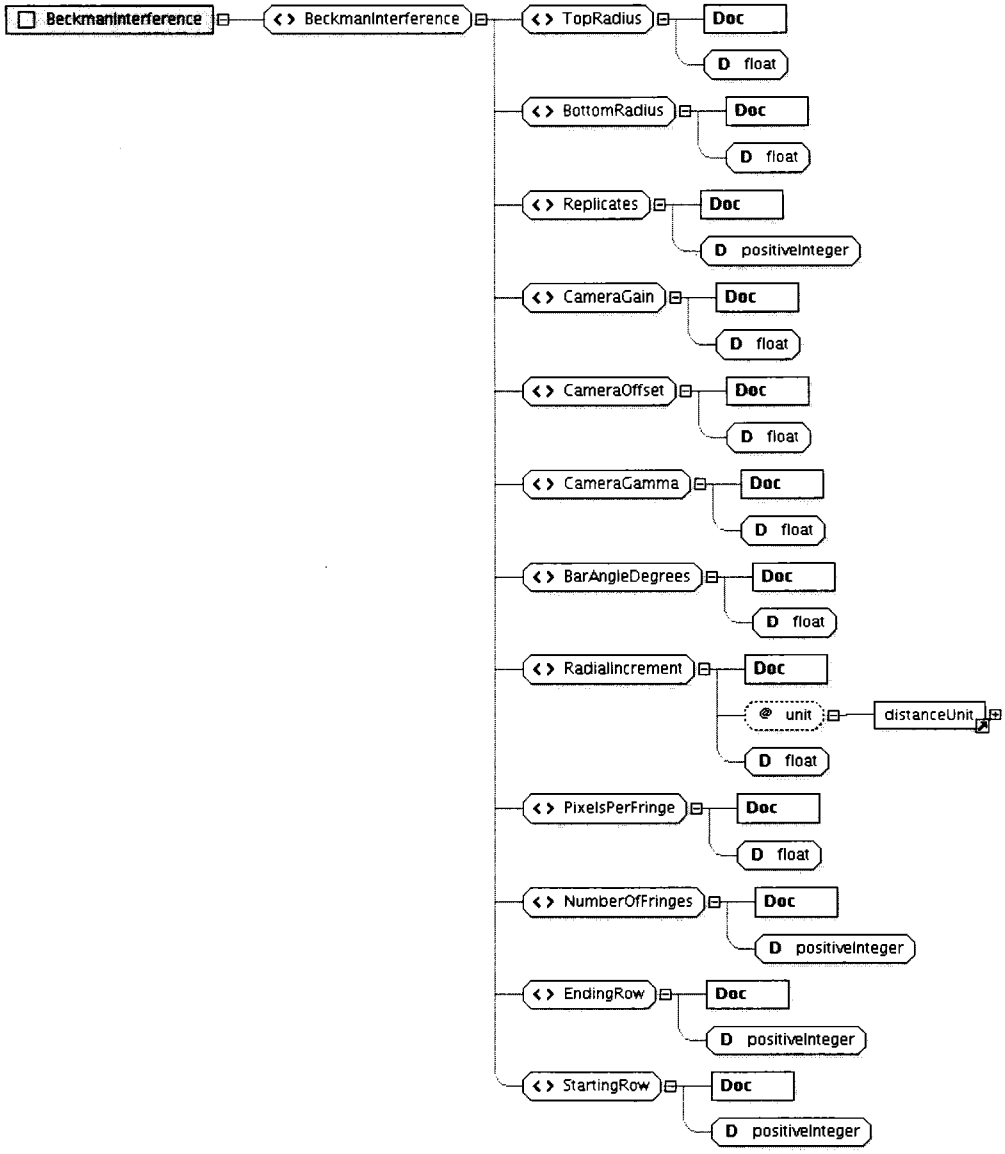


Figure A-2: Beckman Interference

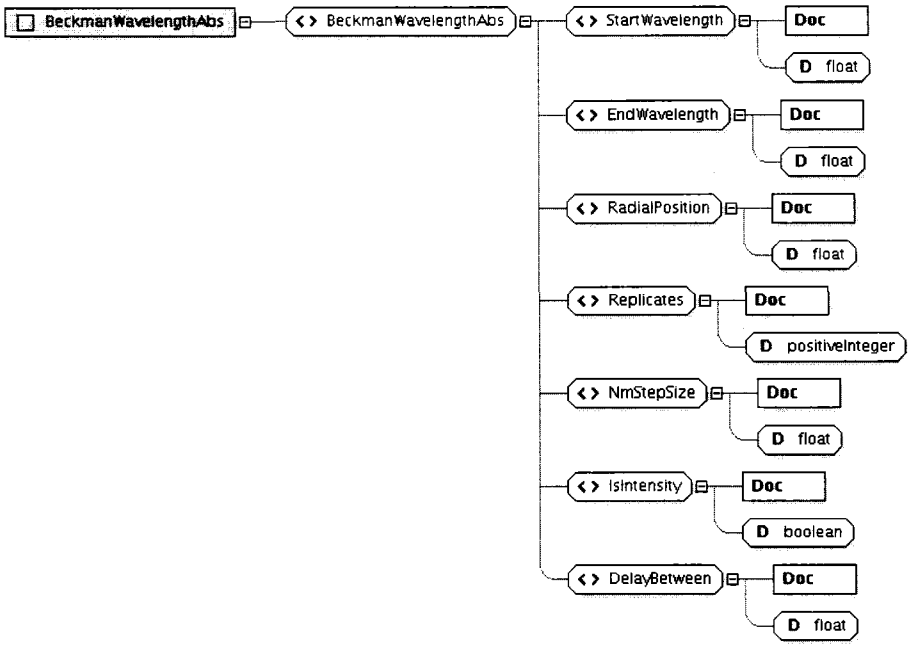


Figure A-3: Beckman Wavelength Absorbance

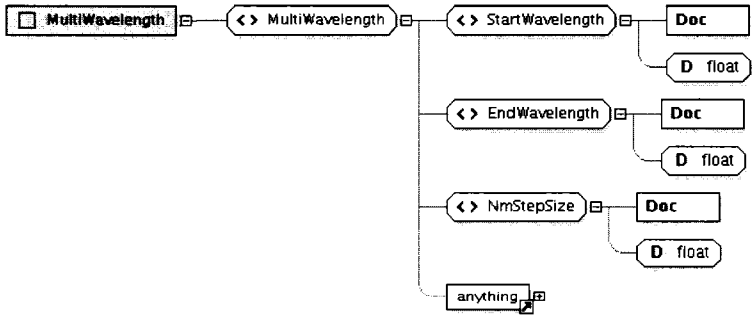


Figure A-4: Multi-wavelength

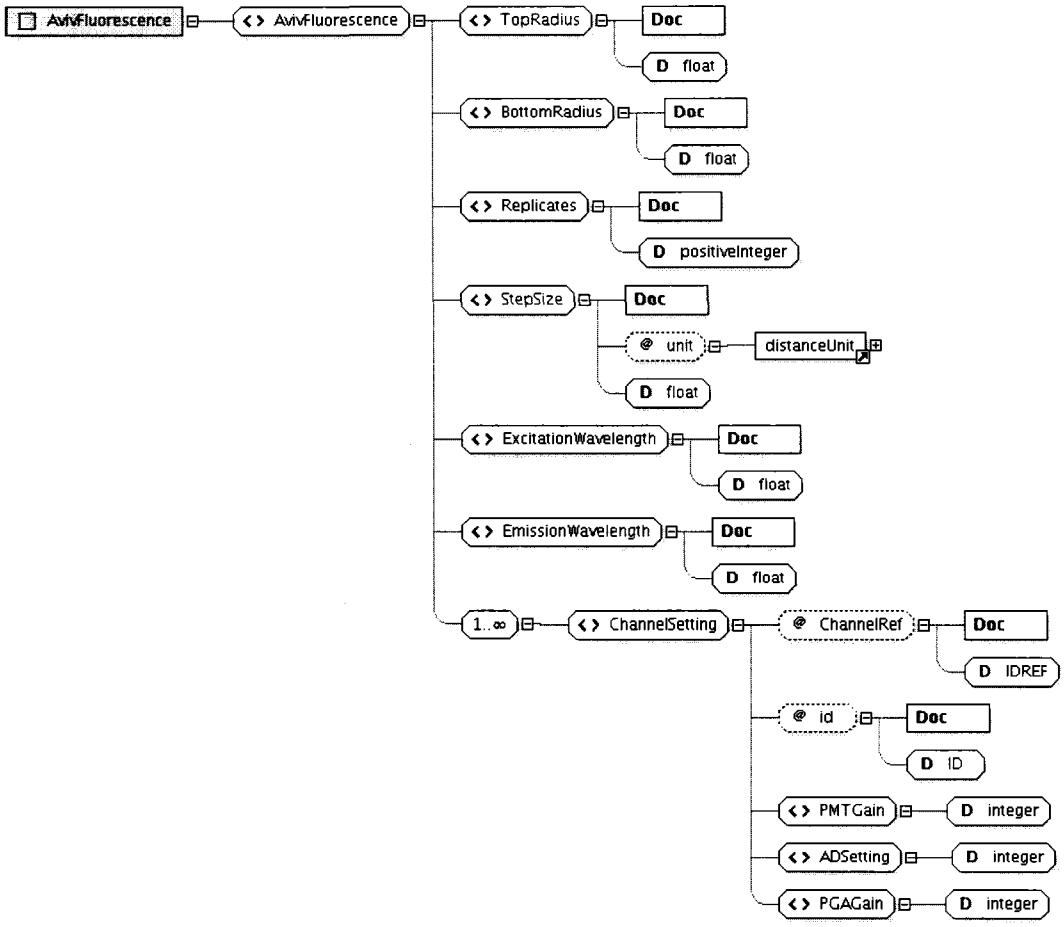


Figure A-5: Aviv Fluorescence

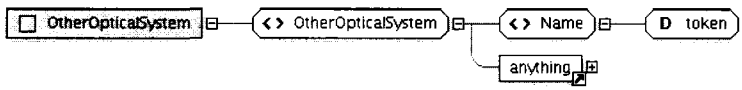


Figure A-6: Other Optical System

APPENDIX B

MEASURED PROPERTEIS OF SOLUTIONS AND ANALYTES

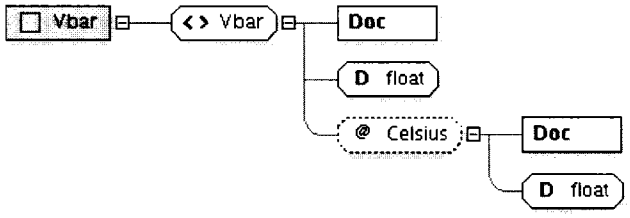


Figure B-1: Partial Specific Volume

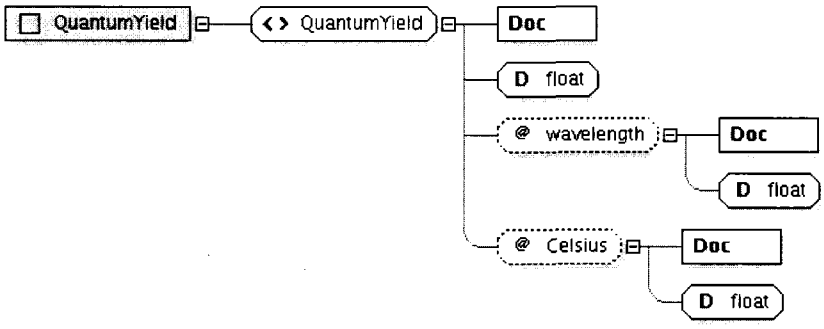


Figure B-2: Quantum Yield

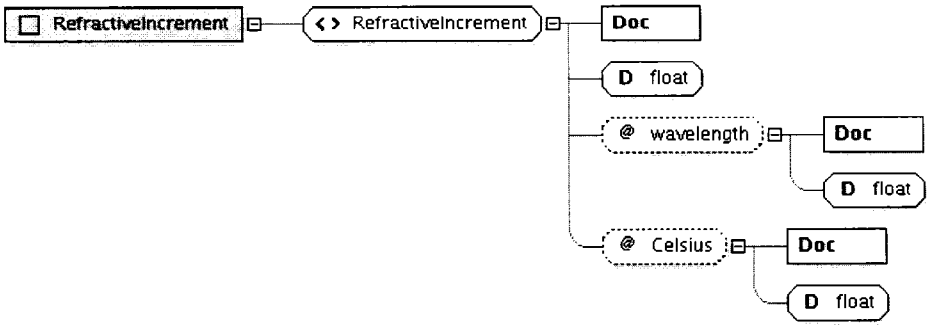


Figure B-3: Refractive Increment

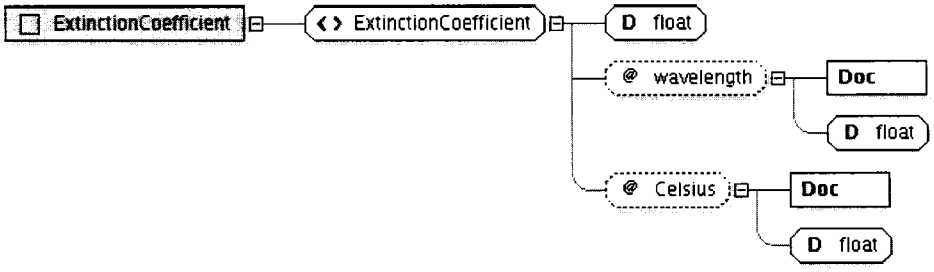


Figure B-4: Extinction Coefficient

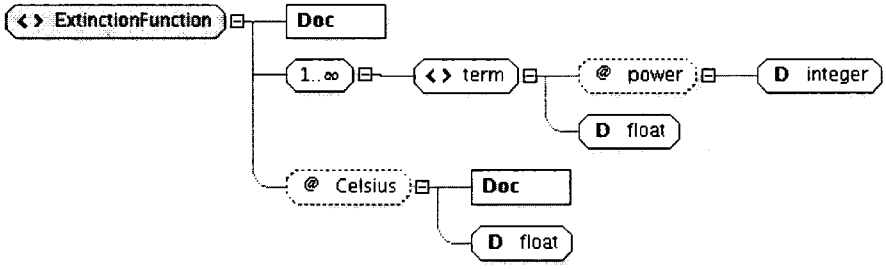


Figure B-5: Extinction Function

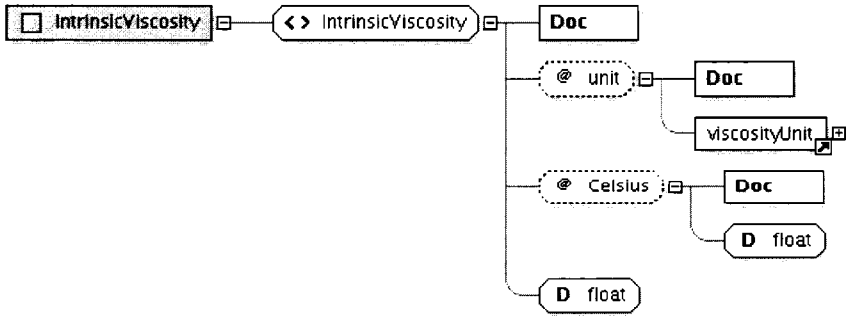


Figure B-6: Intrinsic Viscosity

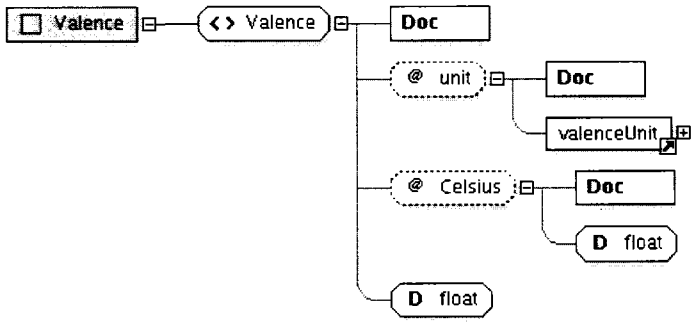


Figure B-7: Valence

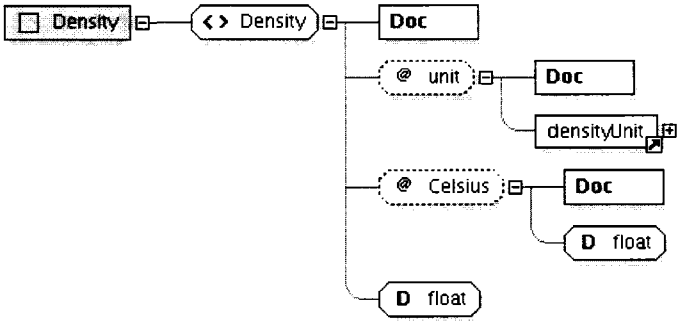


Figure B-8: Density

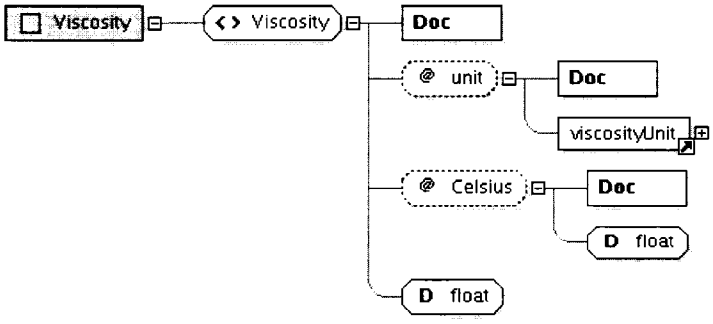


Figure B-9: Viscosity

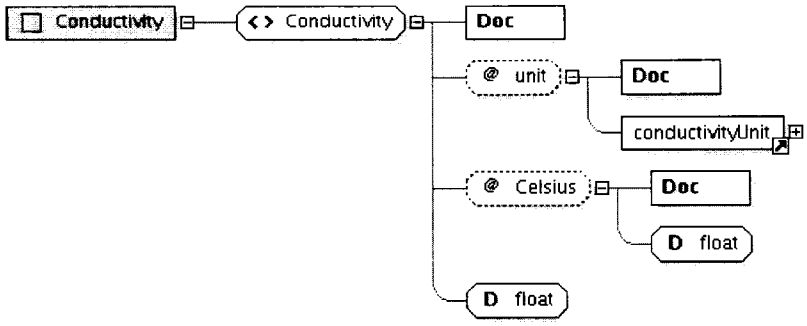


Figure B-10: Conductivity

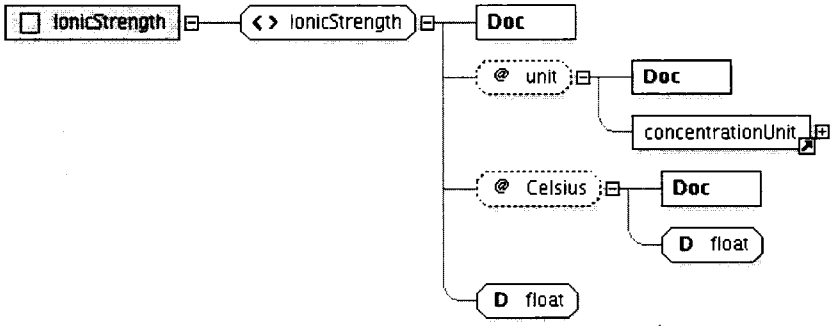


Figure B-11: Ionic Strength

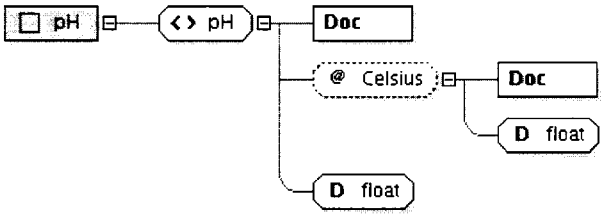


Figure B-12: pH

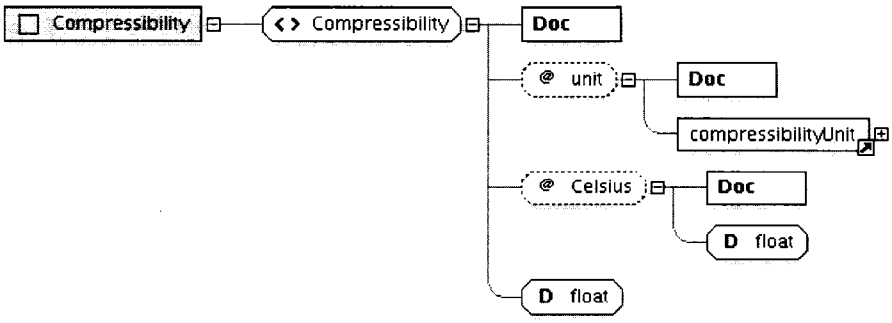


Figure B-13: Compressibility

Figure B-14: Absorption Spectrum

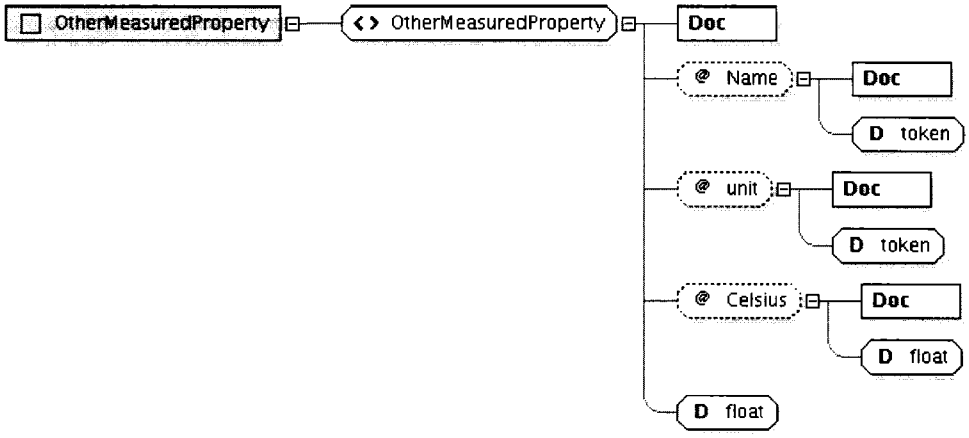


Figure B-15: Other Measured Property

APPENDIX C

REPORTING OF UNITS

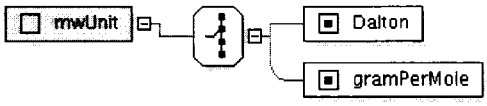


Figure C-1: Molecular Weight Unit

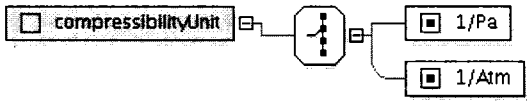


Figure C-2: Compressibility Unit

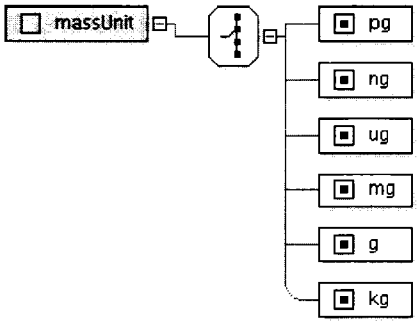


Figure C-3: Mass Unit

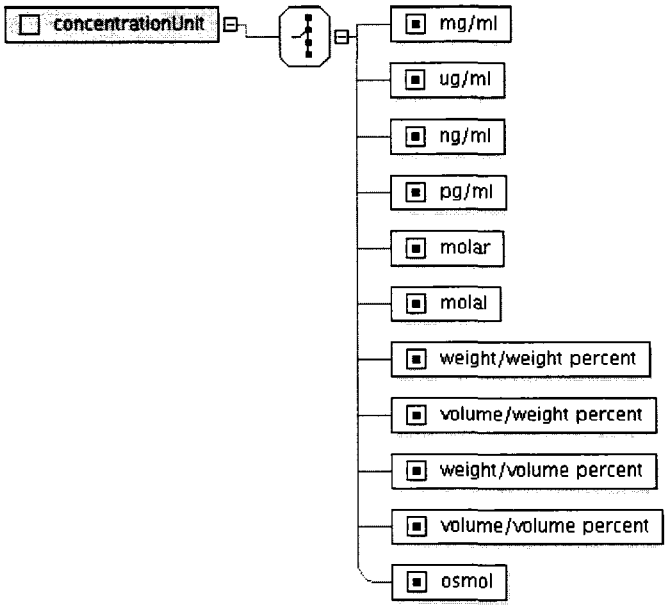


Figure C-4: Concentration Unit

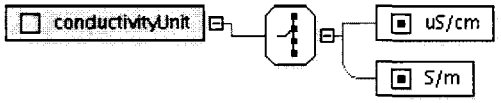


Figure C-5: Conductivity Unit

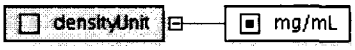


Figure C-6: Density Unit

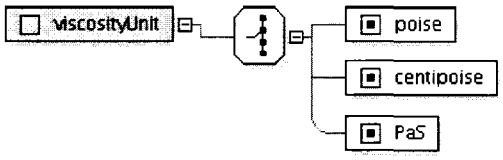


Figure C-7: Viscosity Unit

APPENDIX D

FILETREE LIBRARY

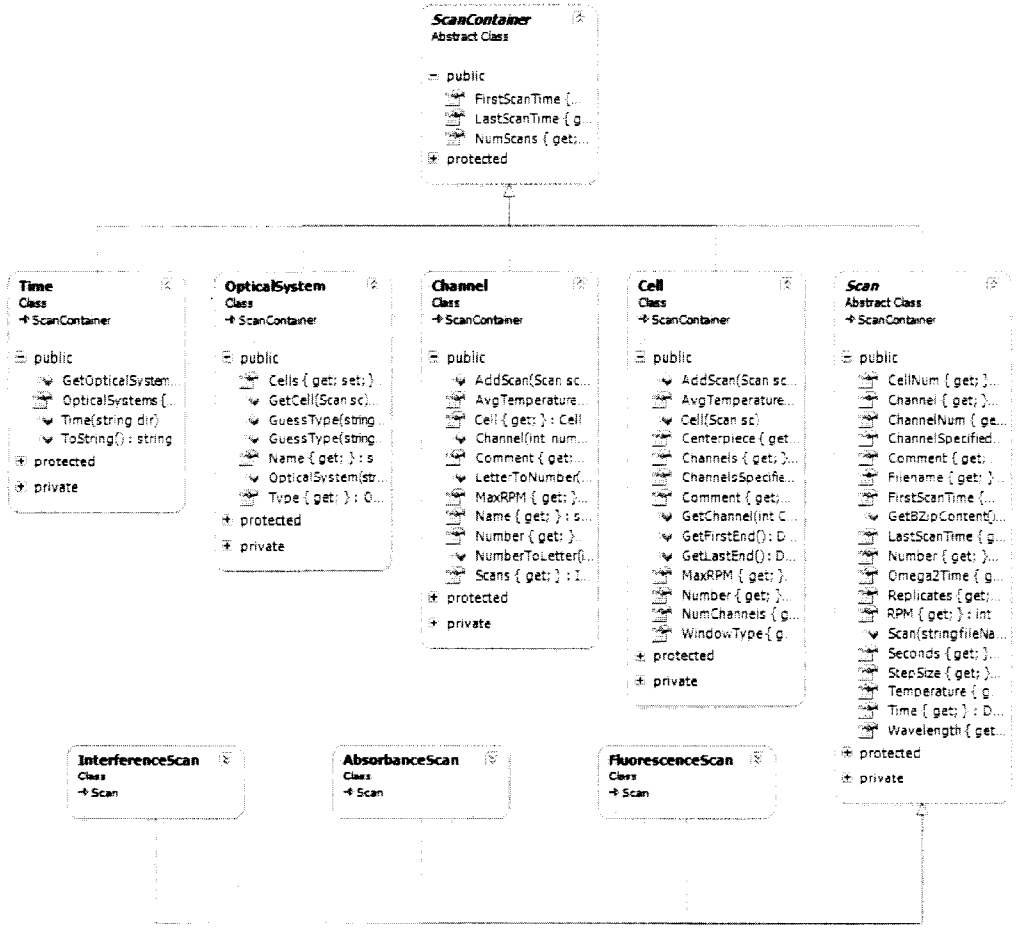


Figure D-1: FileTree Class Library

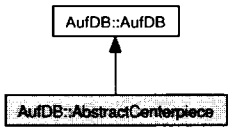
APPENDIX E

PANDaS DATABASE LIBRARY CLASSES

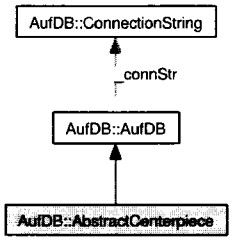
E.1 AufDB::AbstractCenterpiece Class Reference

An abstract centerpiece contains all the details of a centerpiece that are specific to its type (e.g. number of channels).

Inheritance diagram for AufDB::AbstractCenterpiece:



Collaboration diagram for AufDB::AbstractCenterpiece:



Public Member Functions

- AbstractCenterpiece ()
- AbstractCenterpiece (string connstr)
- AbstractCenterpiece (string name, string connstr)
- AbstractCenterpiece (long id, string connstr)
- AbstractCenterpiece (AbstractCenterpiece src)
- AbstractCenterpiece (long id, string name, LoadMethodType loadMethod, string materialName, int maxrpm, float mmThick, string materialRefUri, string centerpieceRefUri, List< AbstractChannel > channels, string connstr)

- AbstractCenterpiece (IDataReader rdr, string connstr)
- override string ToString ()
- AbstractChannel ChannelWithKey (long key)
- override void dbWrite (IDbConnection conn)
- void AddChannel (AbstractChannel ac)

Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

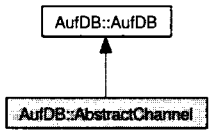
Properties

- string Name `[get, set]`
- string MaterialName `[get, set]`
- int MaxRPM `[get, set]`
- float MmThick `[get, set]`
- LoadMethodType LoadMethod `[get, set]`
- string MaterialRefUri `[get, set]`
- string CenterpieceRefUri `[get, set]`
- ReadOnlyAbsChannelCollection Channels `[get]`

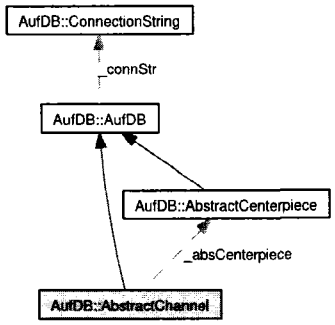
E.2 AufDB::AbstractChannel Class Reference

An abstract channel holds all the details of the layout of channel in a centerpiece.

Inheritance diagram for AufDB::AbstractChannel:



Collaboration diagram for AufDB::AbstractChannel:



Public Member Functions

- AbstractChannel (string connstr)
- AbstractChannel (AbstractCenterpiece ac, int key, ChannelShape shape, ChannelType type, string name, int number, float radialBegin, float radialEnd, float degreesWide, float degreesOffset, float mmDeep, float radialBandTop, float radialBandBottom, float radialMeniscusPos)
- AbstractChannel (AbstractChannel src)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

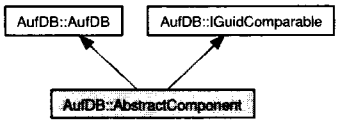
Properties

- string Name `[get, set]`
- int Number `[get, set]`
- float RadialBegin `[get, set]`
- float RadialEnd `[get, set]`
- float DegreesWide `[get, set]`
- float DegreesOffset `[get, set]`
- float MmDeep `[get, set]`
- ChannelShape Shape `[get, set]`
- ChannelType Type `[get, set]`
- float RadialBandTop `[get, set]`
- float RadialBandBottom `[get, set]`
- float RadialMeniscusPos `[get, set]`
- AbstractCenterpiece AbstractCenterpiece `[get, set]`

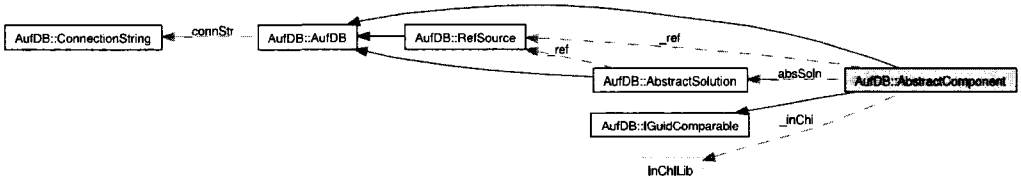
E.3 AufDB::AbstractComponent Class Reference

This class encapsulated the properties of a solute or solvent component of a solution.

Inheritance diagram for AufDB::AbstractComponent:



Collaboration diagram for AufDB::AbstractComponent:



Public Member Functions

- AbstractComponent (string connstr)
- AbstractComponent (string name, string connstr)
- AbstractComponent (long id, string connstr)
- AbstractComponent (AbstractComponent src)
- AbstractComponent (int id, Guid guid, string name, string inchi, float mw, float density, double weightFraction, bool isAnalyte, bool isSolvent, string connstr)
- bool EditsAllowed (Component c)
- override int CompareTo (object obj)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void handleOutParams (IDbCommand cmd)
- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

Properties

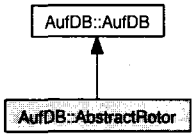
- RefSource RefSource `[get, set]`

- string Name [get, set]
- InChI InChi [get, set]
- float Mw [get, set]
- bool IsAnalyte [get, set]
- bool IsSolvent [get, set]
- double WeightFraction [get, set]
- AbstractSolution AbstractSolution [get, set]
- int NumSolutionsUsing [get]
- float BulkDensity [get, set]
- Guid Guid [get, set]

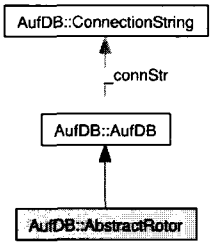
E.4 AufDB::AbstractRotor Class Reference

An abstract rotor encapsulates all the properties of a rotor that are specific to its type (# of holes, etc.).

Inheritance diagram for AufDB::AbstractRotor:



Collaboration diagram for AufDB::AbstractRotor:



Public Member Functions

- AbstractRotor ()
- AbstractRotor (AbstractRotor src)
- AbstractRotor (long key, string connectionstring)
- AbstractRotor (string name, string materialName, int numHoles, int maxRPM, string manufacturer, string connstr)
- AbstractRotor (IDataReader rdr, string connstr)
- override string ToString ()

Protected Member Functions

- override void setVariables (System.Data.IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

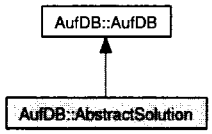
Properties

- string Name `[get, set]`
- string MaterialName `[get, set]`
- int NumHoles `[get, set]`
- int MaxRPM `[get, set]`
- string Manufacturer `[get, set]`
- string MaterialReferenceUri `[get, set]`

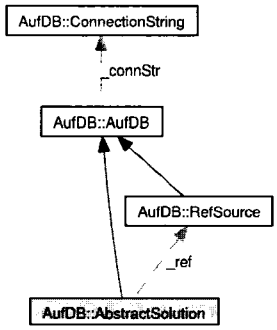
E.5 AufDB::AbstractSolution Class Reference

Represents the concept of a specific solution including the relative amouts of each component.

Inheritance diagram for AufDB::AbstractSolution:



Collaboration diagram for AufDB::AbstractSolution:



Public Member Functions

- AbstractSolution ()

- AbstractSolution (string connstr)
- AbstractSolution (string name, string connstr)
- AbstractSolution (AbstractSolution src)
- AbstractSolution (long id, string connstr)
- AbstractSolution (long id, string name, string connstr)
- AbstractSolution (IDataReader rdr, string _connstr)
- void AllowEdits ()
- void AddSolvent (AbstractComponent ac)
- void AddSolvent (AbstractComponent ac, double weightFraction)
- void AddSolute (AbstractComponent ac)
- void AddSolute (AbstractComponent ac, double weightFraction)
- override string ToString ()
- string ContentString ()
- bool ContainsComponent (AbstractComponent ac)
- void RegisterSolution (Solution solution)
- bool EditsAllowed (Solution caller)
- override void dbWrite (System.Data.IDbConnection conn)

Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

Properties

- RefSource RefSource `[get, set]`
- string Name `[get, set]`
- System.Collections.ObjectModel.ReadOnlyCollection< AbstractComponent > SoluteComponents `[get]`
- System.Collections.ObjectModel.ReadOnlyCollection< AbstractComponent > SolventComponents `[get]`
- int NumSolutionsUsing `[get]`

E.6 AufDB::Acl Class Reference

Wraps up a set of ACL entries so they're treated as a collection.

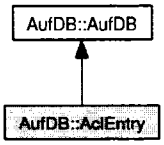
Public Member Functions

- void Add (AclEntry entry)
- AclEntry Item (int index)
- bool Contains (AclEntry entry)
- int IndexOf (AclEntry entry)

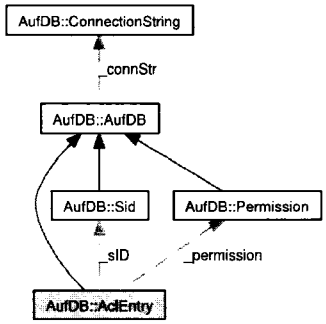
E.7 AufDB::AclEntry Class Reference

Represents an entry in an access control list.

Inheritance diagram for AufDB::AclEntry:



Collaboration diagram for AufDB::AclEntry:



Public Member Functions

- AclEntry ()
- AclEntry (AclEntry src)
- AclEntry (string connStr)
- AclEntry (Sid sid, Permission perm, AclResourceType resource, System.DateTime dateExpires, string connStr)
- AclEntry (Sid sid, Permission perm, AclResourceType resource, string connStr)
- AclEntry (long ACLKey, string connStr)
- override void dbWrite (System.Data.IDbConnection conn)

Protected Member Functions

- override void setVariables (IDataReader rdr)
- override void addPKParams (IDbCommand cmd)
- override void addInsertParams (IDbCommand cmd)

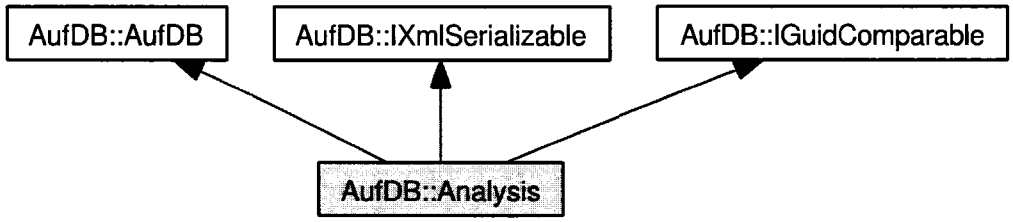
Properties

- Sid sID `[get, set]`
- Permission permission `[get, set]`
- AclResourceType resource `[get, set]`
- DateTime dateExpires `[get, set]`

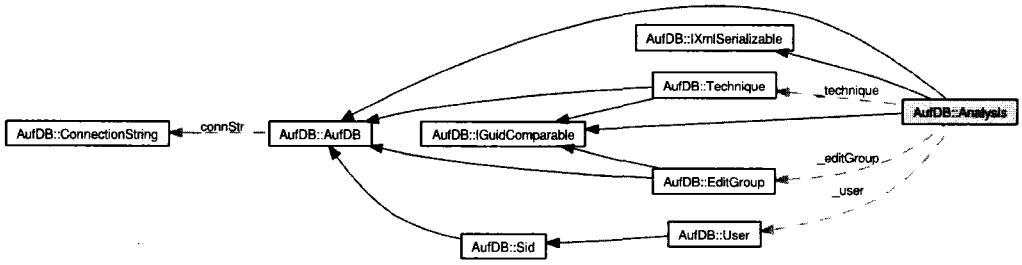
E.8 AufDB::Analysis Class Reference

This class represents the application of an analysis technique to a particular set of data.

Inheritance diagram for AufDB::Analysis:



Collaboration diagram for AufDB::Analysis:



Public Member Functions

- void addResult (Result newResult)
- Analysis ()
- Analysis (Analysis src)
- Analysis (string connStr)
- Analysis (long analysisKey, string connStr)
- Analysis (Technique technique, EditGroup editGroup, User user, string connStr)
- Analysis (Xml.Analysiscontent xAnalysis, string connstr)

- override void dbRead (IDbConnection conn)
- override void dbWrite (IDbConnection conn)
- override void dbDelete (IDbConnection conn)
- System.IO.MemoryStream ToXml ()

Protected Member Functions

- override void setVariables (IDataReader rdr)
- override void addPKParams (IDbCommand cmd)
- override void addInsertParams (IDbCommand cmd)

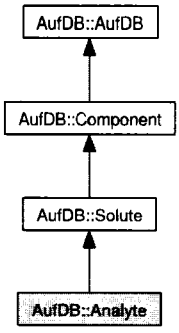
Properties

- Technique technique `[get, set]`
- string description `[get, set]`
- DateTime date `[get, set]`
- EditGroup editGroup `[get, set]`
- System.Collections.ObjectModel.ReadOnlyCollection< Result > Results `[get]`
- User User `[get, set]`
- Guid Guid `[get, set]`

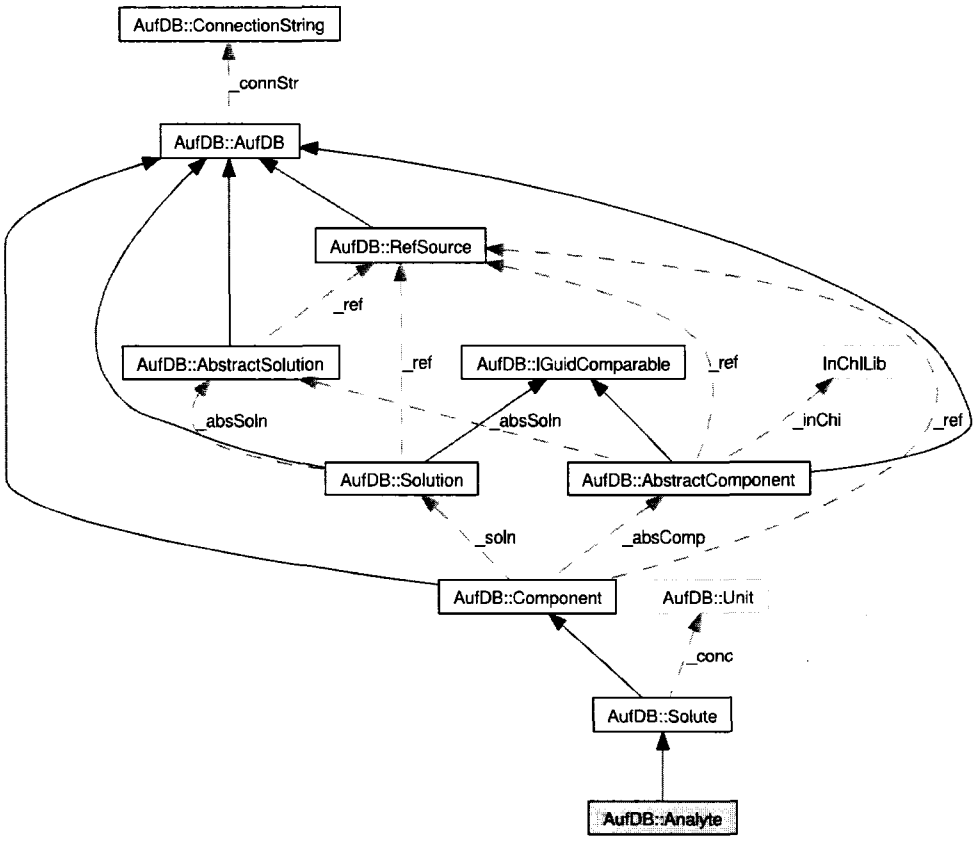
E.9 AufDB::Analyte Class Reference

This class represents the molecule being studied in a specific batch of a solution.

Inheritance diagram for AufDB::Analyte:



Collaboration diagram for AufDB::Analyte:



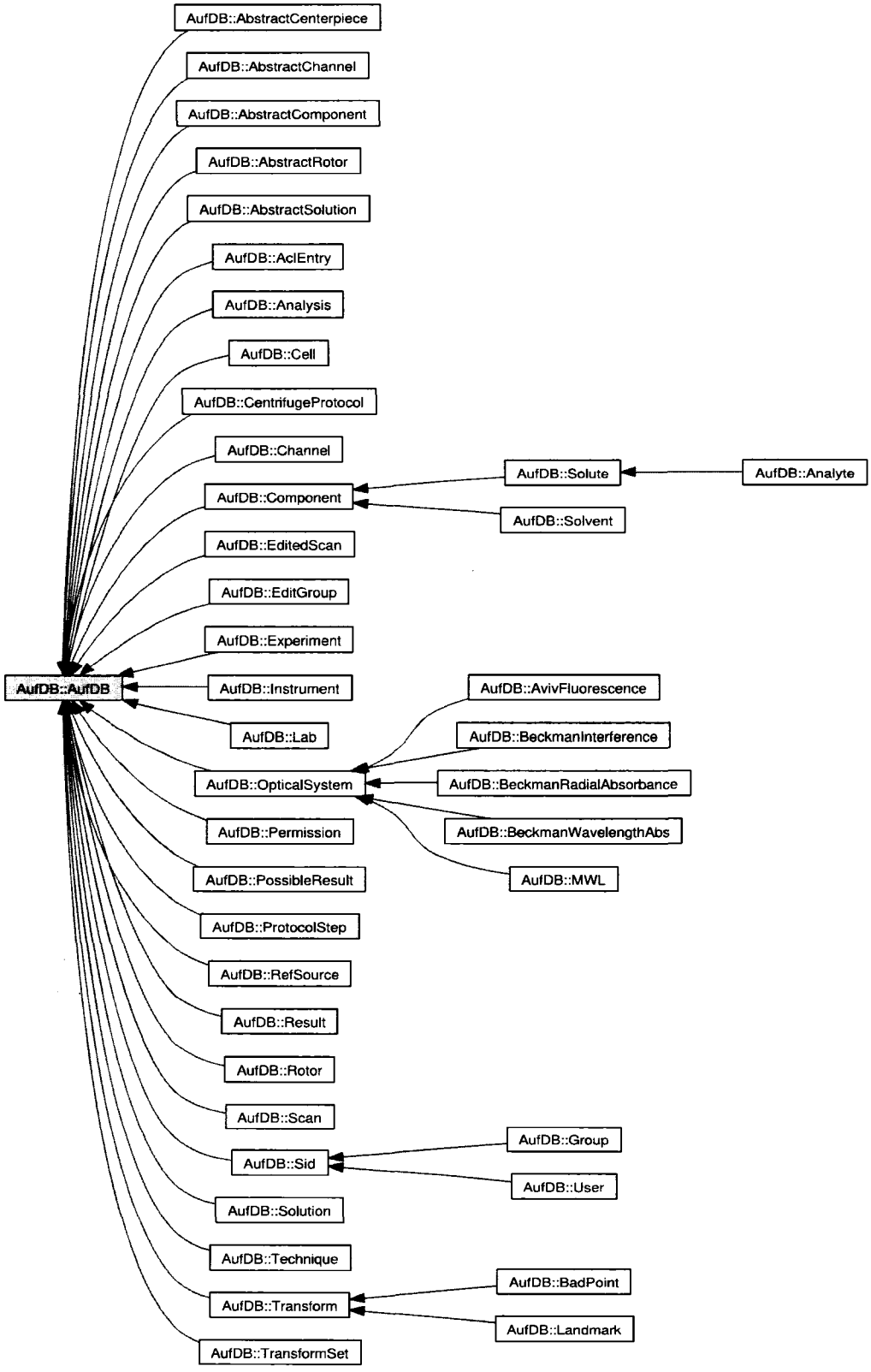
Public Member Functions

- Analyte (string connstr)
- Analyte (Analyte src)
- Analyte (AbstractComponent ac, double mg)
- Analyte (long key, string connstr)
- Analyte (long key, AbstractComponent ac, string name, string lotid, DateTime dateCreated, DateTime dateExpired, string connstr)
- Analyte (string name, float molecularWeight, string connstr)

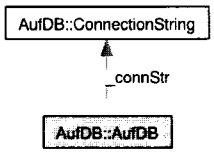
E.10 AufDB::AufDB Class Reference

This class provides database-specific details to derived classes.

Inheritance diagram for AufDB::AufDB:



Collaboration diagram for AufDB::AufDB:



Public Member Functions

- AufDB ()
- AufDB (string connstr)
- AufDB (AufDB copy)
- AufDB (long key, string connstr)
- override bool Equals (Object o)
- override int GetHashCode ()
- virtual int CompareTo (object obj)
- virtual string getFindAllSpName ()
- virtual void dbRead (IDbConnection conn)
- void dbWrite ()
- virtual void dbWrite (IDbConnection conn)
- void dbDelete ()
- virtual void dbDelete (IDbConnection conn)

Static Public Member Functions

- static bool operator== (AufDB a, AufDB b)
- static bool operator!= (AufDB a, AufDB b)

Public Attributes

- const int precision = 12
- string UriBase = "http://localhost/"

Protected Member Functions

- AufDB (long key, string connstr, bool deferRead)
- delegate void dataProcessorDelegate (IDataReader rdr)
- delegate void paramAdderDelegate (IDbCommand cmd)
- virtual string getReadSpName ()
- virtual string getInsertSpName ()
- virtual string getUpdateSpName ()
- virtual string getDeleteSpName ()
- abstract void addInsertParams (IDbCommand cmd)

- abstract void addPKParams (IDbCommand cmd)
- abstract void setVariables (IDataReader rdr)
- virtual void addUpdateParams (IDbCommand cmd)
- virtual void addReadParams (IDbCommand cmd)
- virtual void addDeleteParams (IDbCommand cmd)
- void dbRead ()
- virtual void dbRead (IDbConnection conn, string spName, paramAdderDelegate paramAdder, dataProcessorDelegate readHandler)
- virtual void dbRead (IDbCommand cmd, paramAdderDelegate paramAdder, dataProcessorDelegate readHandler)
- virtual void dbWrite (IDbConnection conn, string spName, paramAdderDelegate paramAdder)
- virtual void handleOutParams (IDbCommand cmd)
- virtual void checkAndSet< T > (ref T var, T val)

Static Protected Member Functions

- static void addParam (System.Data.IDbCommand cmd, string paramname, DbType type, Object value)

Protected Attributes

- bool _doWrite
- long _key
- DateTime _dateUpdated
- string _spBaseName

Static Protected Attributes

- static readonly ILog log = log4net.LogManager.GetLogger(typeof(AufDB))

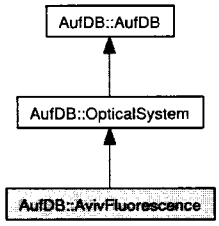
Properties

- String ConnectionString `[get, set]`
- long key `[get]`
- DateTime dateUpdated `[get]`
- bool IsChanged `[get]`
- string StoredProcedureBaseName `[get]`

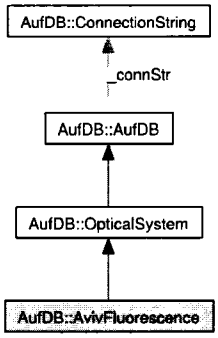
E.11 AufDB::AvivFluorescence Class Reference

This class represents the settings of an Aviv fluorescence optical system.

Inheritance diagram for AufDB::AvivFluorescence:



Collaboration diagram for AufDB::AvivFluorescence:



Public Member Functions

- AvivFluorescence ()

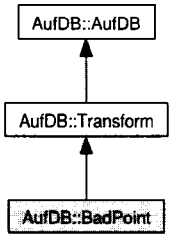
Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

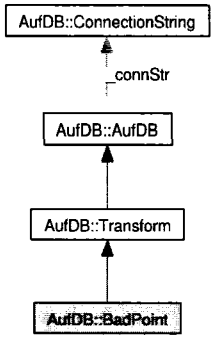
E.12 AufDB::BadPoint Class Reference

A Bad Point is a type of transform that marks a specific point in a scan.

Inheritance diagram for AufDB::BadPoint:



Collaboration diagram for AufDB::BadPoint:



Public Member Functions

- BadPoint ()
- BadPoint (BadPoint badpoint)
- BadPoint (string connStr)
- BadPoint (int pointNumber, string reason, string connStr)
- BadPoint (long transformKey, int pointNumber, string name, string reason, DateTime date-Updated, string connStr)
- BadPoint (long transformKey, string connStr)
- BadPoint (Xml.Transform xTrans, string connstr)

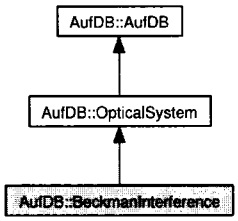
Properties

- int number `[get, set]`

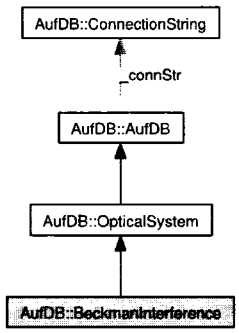
E.13 AufDB::BeckmanInterference Class Reference

This class represents the settings of an Beckman interference optical system.

Inheritance diagram for AufDB::BeckmanInterference:



Collaboration diagram for AufDB::BeckmanInterference:



Public Member Functions

- BeckmanInterference ()

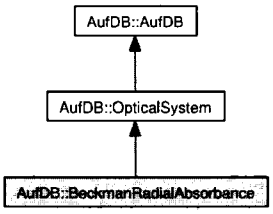
Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

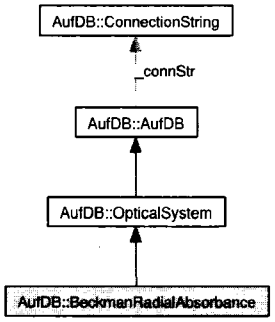
E.14 AufDB::BeckmanRadialAbsorbance Class Reference

This class represents the settings of an Beckman radial absorbance optical system.

Inheritance diagram for AufDB::BeckmanRadialAbsorbance:



Collaboration diagram for AufDB::BeckmanRadialAbsorbance:



Public Member Functions

- BeckmanRadialAbsorbance ()

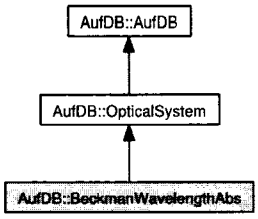
Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

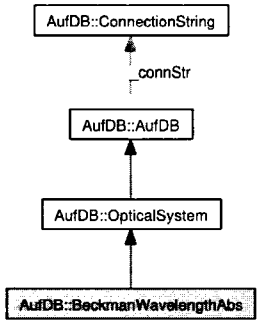
E.15 AufDB::BeckmanWavelengthAbs Class Reference

This class represents the settings of a Beckman Wavelength scanning absorbance optical system.

Inheritance diagram for AufDB::BeckmanWavelengthAbs:



Collaboration diagram for AufDB::BeckmanWavelengthAbs:



Public Member Functions

- BeckmanWavelengthAbs ()

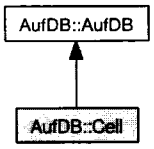
Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

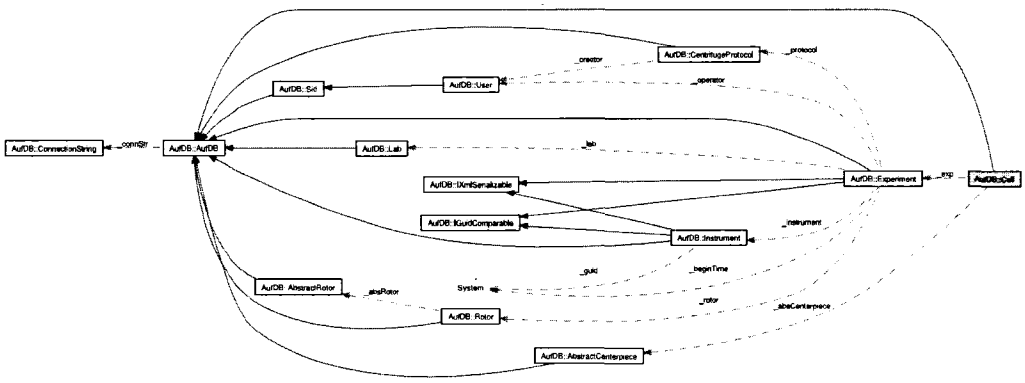
E.16 AufDB::Cell Class Reference

Represents a set of housing, and centerpiece for a specific centrifuge experiment.

Inheritance diagram for AufDB::Cell:



Collaboration diagram for AufDB::Cell:



Public Member Functions

- Cell (string connstr)
- Cell (long key, string connstr)
- Cell (Experiment exp, long key, string connstr)
- Cell (Cell src)
- void AddChannel (Channel c)
- void ReplaceChannel (Channel c)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)
- override void setVariables (IDataReader rdr)

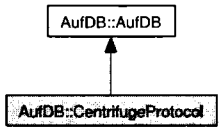
Properties

- string Name `[get, set]`
- ReadOnlyChannelCollection Channels `[get]`
- AbstractCenterpiece AbstractCenterpiece `[get, set]`
- string CenterpieceSerialNumber `[get, set]`
- int HoleNumber `[get, set]`
- WindowType WindowType `[get, set]`
- Experiment Experiment `[get, set]`

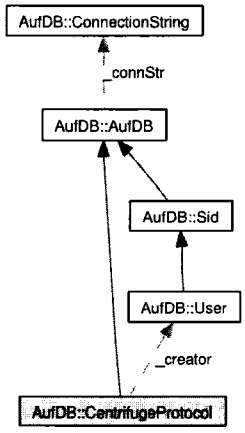
E.17 AufDB::CentrifugeProtocol Class Reference

Contains a list of the steps in a centrifuge experiment, including speed, temperature, etc.

Inheritance diagram for AufDB::CentrifugeProtocol:



Collaboration diagram for AufDB::CentrifugeProtocol:



Public Member Functions

- CentrifugeProtocol ()
- void AddStep (ProtocolStep step)

Protected Member Functions

- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)
- override void addInsertParams (System.Data.IDbCommand cmd)

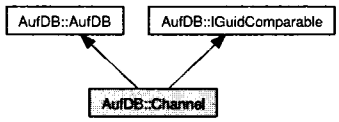
Properties

- User Creator `[get, set]`
- bool WaitForVacuum `[get, set]`
- bool ShutdownAfter `[get, set]`
- string Name `[get, set]`
- DateTime CreationDate `[get, set]`
- DateTime LastUsedDate `[get, set]`
- List< ProtocolStep > Steps `[get]`
- int NumScanSets `[get]`

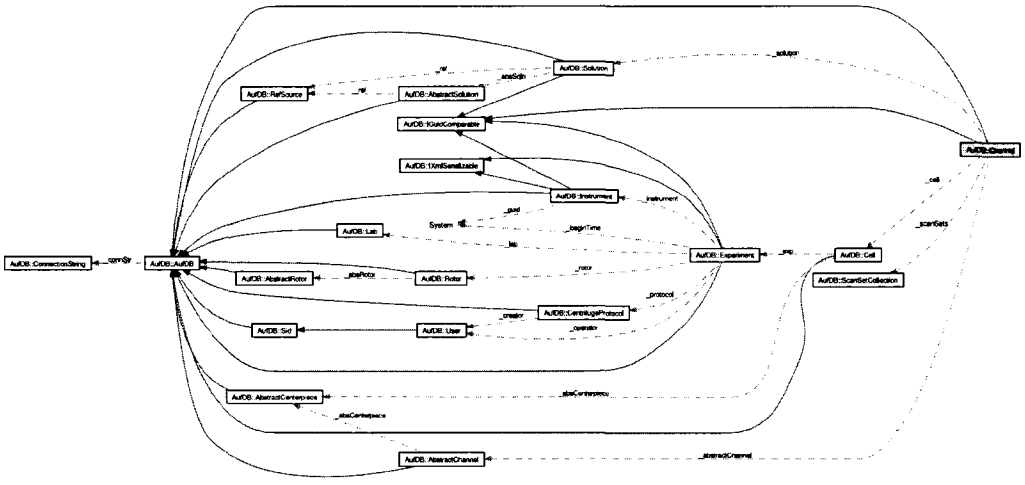
E.18 AufDB::Channel Class Reference

Represents a channel in a centerpiece. Contains scans as they are acquired from the instrument.

Inheritance diagram for AufDB::Channel:



Collaboration diagram for AufDB::Channel:



Public Member Functions

- Channel (string connstr)
- Channel (long key, string connstr)
- Channel (Cell c, long key, string connstr)
- Channel (long key, AbstractChannel absc, Solution soln, Cell cell, Guid guid, string comment, DateTime dateUpdated, string connstr)
- Channel (Channel src)
- OpticalSystem GetOpticalSystem (OpticalSystemType opSysType)
- bool AddOpticalSystem (OpticalSystemType opSysType)
- void AddScan (Scan s, OpticalSystemType opsysType)
- System.Collections.ObjectModel.ReadOnlyCollection< Scan > GetScans (OpticalSystemType opSysType)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void addPKParams (IDbCommand cmd)
- override void addInsertParams (IDbCommand cmd)
- override void setVariables (IDataReader rdr)

Properties

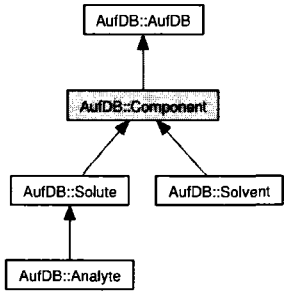
- ChannelType Type `[get]`
- ChannelShape Shape `[get]`
- string Name `[get]`

- int Number `[get]`
- float RadialBegin `[get]`
- float RadialEnd `[get]`
- string Comment `[get, set]`
- AbstractChannel AbstractChannel `[get, set]`
- Solution Solution `[get, set]`
- Cell Cell `[get, set]`
- Guid Guid `[get, set]`

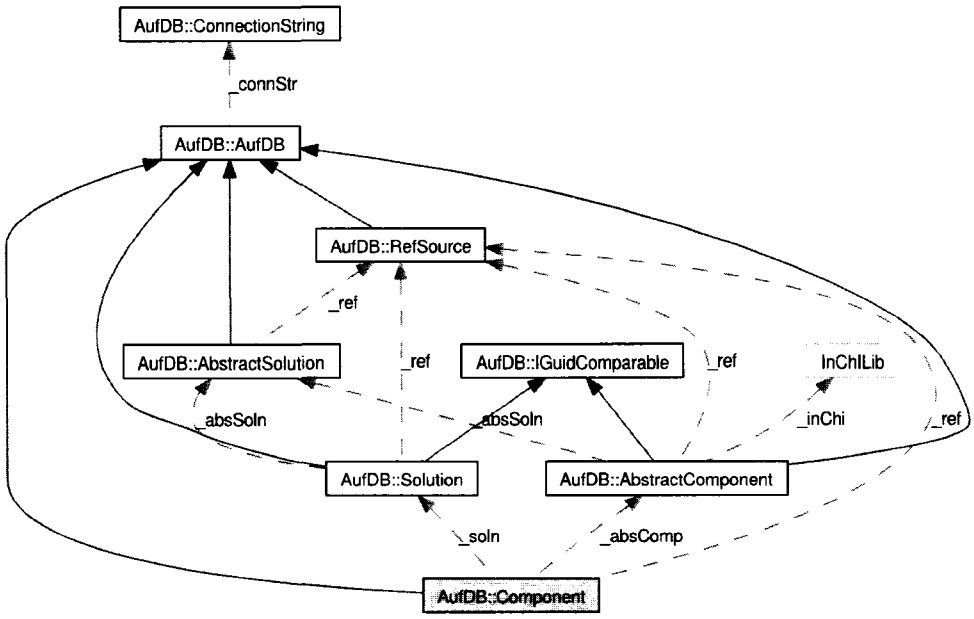
E.19 AufDB::Component Class Reference

This class represents a solution component, referring to a specific batch of this component.

Inheritance diagram for AufDB::Component:



Collaboration diagram for AufDB::Component:



Public Member Functions

- Component ()
- Component (string connstr)
- Component (Component src)
- Component (long key, string connstr)
- Component (AbstractComponent ac, double mg)
- Component (long key, AbstractComponent ac, string name, string lotid, DateTime dateCreated, DateTime dateExpired, string connstr)
- Component (IDataReader rdr, string connstr)
- override int CompareTo (object obj)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

Protected Attributes

- string _name
- double _mg

Properties

- double Mg [get, set]
- string LotIdentifier [get, set]
- RefSource RefSource [get, set]
- DateTime DateCreated [get, set]
- DateTime DateExpired [get, set]
- float lotMw [get, set]
- float Mw [get]
- string Name [get, set]
- InChILib.InChI InChi [get, set]
- float CanonicalMw [get, set]
- double WeightFraction [get, set]
- bool IsAnalyte [get, set]
- bool IsSolvent [get, set]
- Solution Solution [get, set]
- AbstractComponent AbsComponent [get]

E.20 AufDB::ConnectionString Class Reference

This class encapsulates all the details of a string containting information to connect to a specific database.

Public Member Functions

- ConnectionString ()
- ConnectionString (String connstr)
- ConnectionString (string server, string database, string user, string password, string provider)
- override Boolean Equals (Object a)
- override int GetHashCode ()
- override String ToString ()

Static Public Member Functions

- static Boolean operator== (String a, ConnectionString b)
- static Boolean operator== (ConnectionString a, ConnectionString b)
- static Boolean operator!= (String a, ConnectionString b)
- static Boolean operator!= (ConnectionString a, ConnectionString b)

Properties

- bool isValid [get]
- String server [get, set]
- String username [get, set]
- String password [get, set]
- String dbname [get, set]
- String strValue [get, set]

E.21 AufDB::DBUtility Class Reference

This class contains some database utility methods.

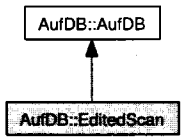
Static Public Member Functions

- static void SpToList (System.Type t, System.Collections.IList list, string spName, string connstr)

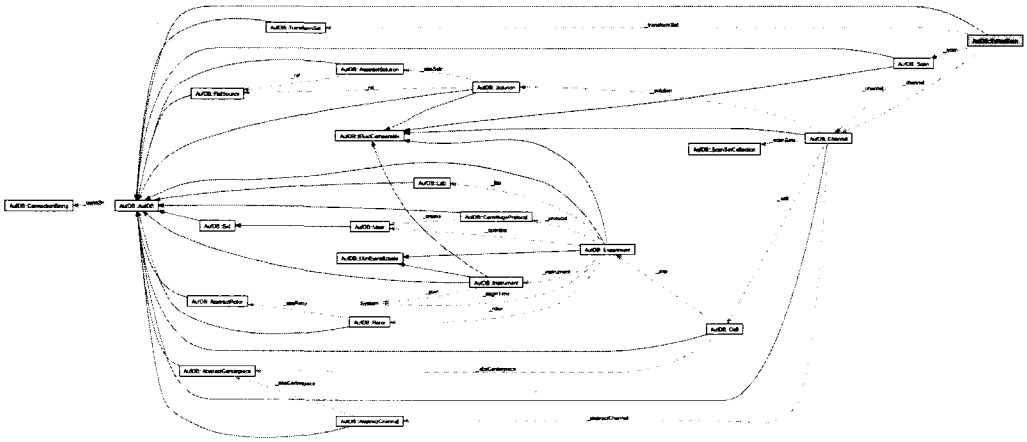
E.22 AufDB::EditedScan Class Reference

This class combines a scan with a set of transforms to document changes to a scan.

Inheritance diagram for AufDB::EditedScan:



Collaboration diagram for AufDB::EditedScan:



Public Member Functions

- EditedScan (string connStr)

- EditedScan (long scanKey, long transformSetKey, long channelKey, string connStr)
- EditedScan (Channel channel, long scanKey, long transformSetKey, string connStr)
- EditedScan (Scan scan, TransformSet transformSet, Channel channel)
- EditedScan (Xml.EditedScancontent xEdScan, string connstr)
- EditedScan (EditedScan EdScan)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void setVariables (IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

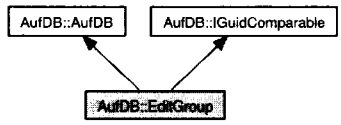
Properties

- Scan Scan `[get]`
- TransformSet TransformSet `[get, set]`
- Channel Channel `[get]`
- string Notes `[get, set]`
- string Name `[get, set]`

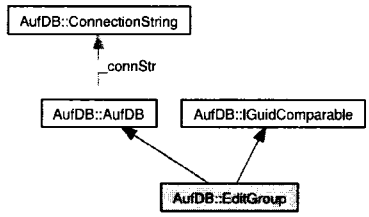
E.23 AufDB::EditGroup Class Reference

This class represents a group of edited scans to be analyzed together.

Inheritance diagram for AufDB::EditGroup:



Collaboration diagram for AufDB::EditGroup:



Public Member Functions

- EditGroup ()
- EditGroup (EditGroup src)
- EditGroup (string connStr)
- EditGroup (long groupKey, string connStr)
- EditGroup (EditedScan[ ] edScans, string connStr)
- EditGroup (Scan[ ] Scans, TransformSet ts, Channel c, string connStr)
- EditGroup (Xml.EditGroupcontent xEditGroup, string connstr)
- void addScan (EditedScan scan)
- void removeScan (EditedScan scan)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void setVariables (IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

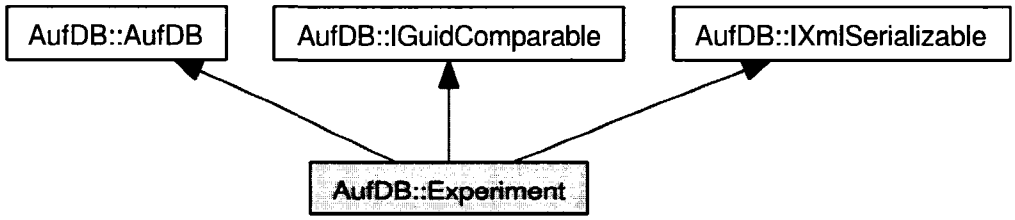
Properties

- string Name `[get, set]`
- Guid Guid `[get, set]`
- string notes `[get, set]`
- DateTime date `[get, set]`
- System.Collections.ObjectModel.ReadOnlyCollection< EditedScan > Scans `[get]`

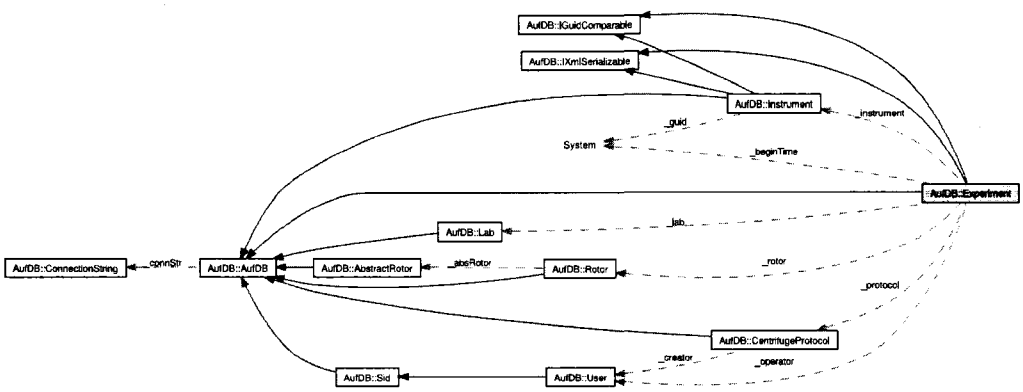
E.24 AufDB::Experiment Class Reference

This class represents the an entire centrifuge experiment, including centrifuge protocol, cell contents, data, and analyses.

Inheritance diagram for AufDB::Experiment:



Collaboration diagram for AufDB::Experiment:



Public Member Functions

- Experiment ()
- Experiment (string connstr)
- Experiment (long id, string connectionString)
- Experiment (Experiment src)
- System.IO.MemoryStream ToXml ()
- void AddCell (Cell c)
- void ReplaceCell (Cell c)
- void RemoveCell (int holenumber)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void setVariables (IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

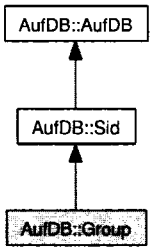
Properties

- Guid Guid `[get, set]`
- DateTime DateBegin `[get, set]`
- ReadOnlyCellCollection Cells `[get]`
- System.Collections.ObjectModel.ReadOnlyCollection< Cell > CellsByIndex `[get]`
- Instrument Instrument `[get, set]`
- Rotor Rotor `[get, set]`
- User Operator `[get, set]`
- Lab Lab `[get, set]`
- CentrifugeProtocol CentrifugeProtocol `[get, set]`

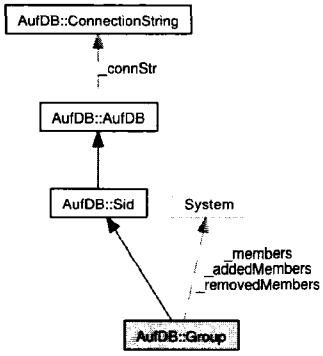
E.25 AufDB::Group Class Reference

Stores a list of SIDs as a group.

Inheritance diagram for AufDB::Group:



Collaboration diagram for AufDB::Group:



Public Member Functions

- Group ()
- Group (Group src)
- Group (string connstr)
- Group (long SIDKey, string connstr)
- Group (Sid[ ] members, string connstr)
- void addMember (Sid sid)
- void removeMember (Sid sid)
- Sid getMember (int index)
- Sid getMember (Sid sid)
- bool isMember (Sid sid)
- override void dbRead (IDbConnection conn)
- override void dbWrite (System.Data.IDbConnection conn)

Protected Member Functions

- void addMembers (IDataReader rdr)
- override void addPKParams (IDbCommand cmd)

Properties

- int Count `[get]`
- Sid[ ] Members `[get]`

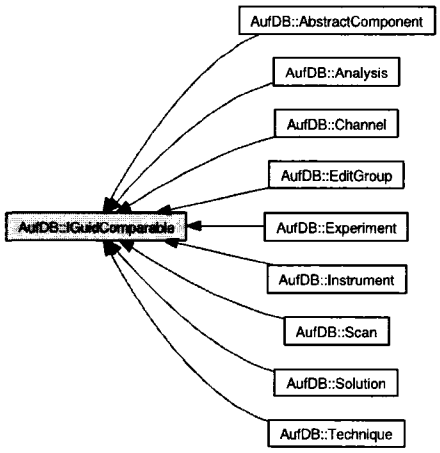
E.26 AufDB::IAccessControl Interface Reference

Public Member Functions

- Acl getACL ()
- void setACL (Acl coll)
- void addACLEntry (AclEntry newACL)

E.27 AufDB::IGuidComparable Interface Reference

Inheritance diagram for AufDB::IGuidComparable:



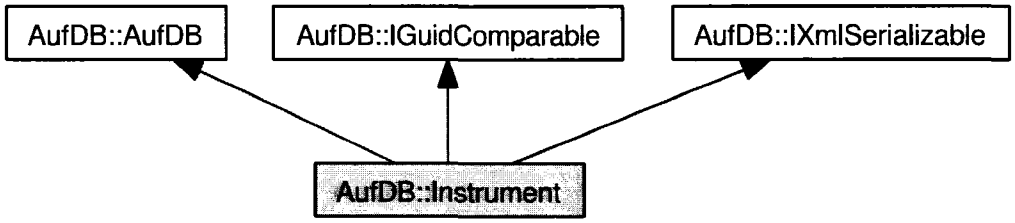
Properties

- Guid Guid `[get, set]`

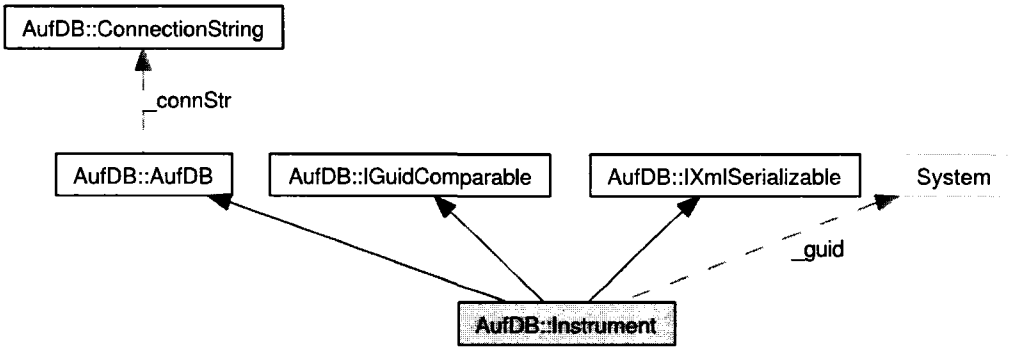
E.28 AufDB::Instrument Class Reference

Encapsulates the properties of a specific Beckman XL AUF instrument.

Inheritance diagram for AufDB::Instrument:



Collaboration diagram for AufDB::Instrument:



Public Member Functions

- Instrument ()
- Instrument (string name, string serialNumber, System.Guid guid, string connstr)
- Instrument (Instrument src)
- Instrument (long key, string connstr)
- Instrument (IDataReader rdr, string connstr)
- Instrument (Xml.Instrumentcontent xInst, string connStr)
- override string ToString ()
- System.IO.MemoryStream ToXml ()

Protected Member Functions

- override void setVariables (System.Data.IDataReader rdr)

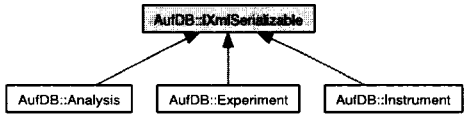
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

Properties

- string Name `[get, set]`
- string SerialNumber `[get, set]`
- Guid Guid `[get, set]`

E.29 AufDB::IXmlSerializable Interface Reference

Inheritance diagram for AufDB::IXmlSerializable:



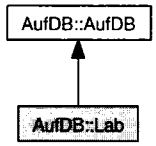
Public Member Functions

- System.IO.MemoryStream ToXml ()

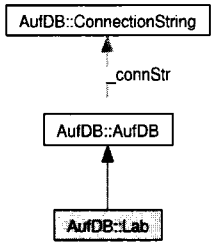
E.30 AufDB::Lab Class Reference

Encapsulates the properties of a physical AUF Lab.

Inheritance diagram for AufDB::Lab:



Collaboration diagram for AufDB::Lab:



Public Member Functions

- Lab ()
- Lab (string connStr)
- Lab (string name, string building, string room, string connstr)
- Lab (Lab src)
- Lab (long key, string connstr)
- Lab (IDataReader rdr, string connstr)
- override string ToString ()

Protected Member Functions

- override void setVariables (System.Data.IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

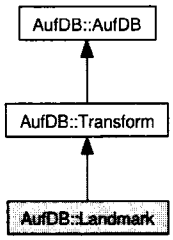
Properties

- string Name `[get, set]`
- string Room `[get, set]`
- string Building `[get, set]`

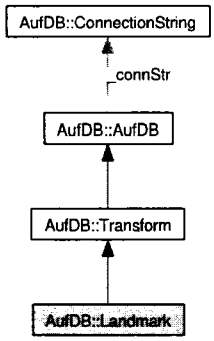
E.31 AufDB::Landmark Class Reference

A position in a scan that may be needed for analysis (e.g meniscus position).

Inheritance diagram for AufDB::Landmark:



Collaboration diagram for AufDB::Landmark:



Public Member Functions

- Landmark ()
- Landmark (Landmark landmark)
- Landmark (string connStr)
- Landmark (float radialPosition, string name, string connStr)
- Landmark (long transformKey, float radialPosition, string name, string notes, DateTime dateUpdated, string connStr)
- Landmark (long transformKey, string connStr)
- Landmark (Xml.Transform xTrans, string connstr)

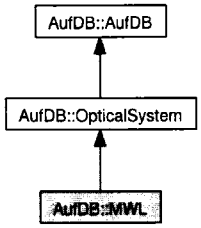
Properties

- float radialPosition `[get, set]`

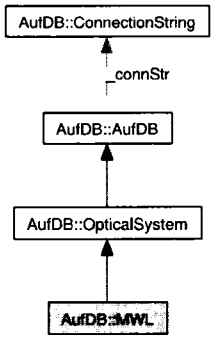
E.32 AufDB::MWL Class Reference

This class represents the settings of a Multi Wavelength absorbance optical system.

Inheritance diagram for AufDB::MWL:



Collaboration diagram for AufDB::MWL:



Public Member Functions

- MWL ()

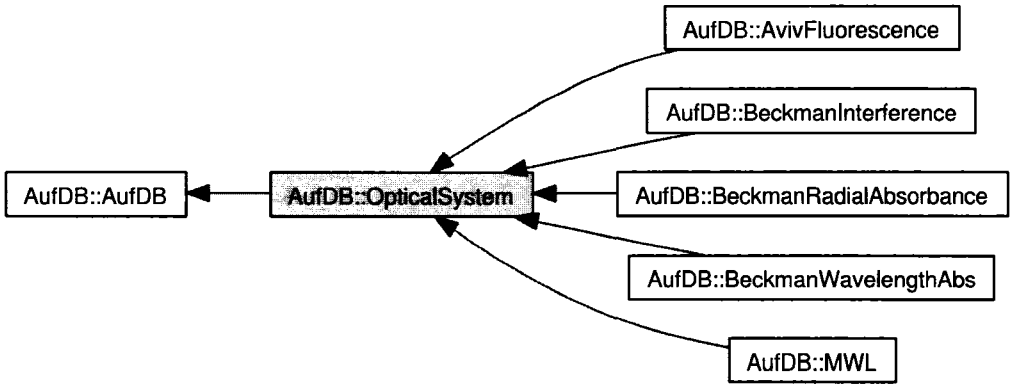
Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

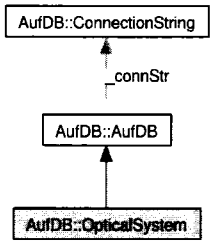
E.33 AufDB::OpticalSystem Class Reference

Optical system classes contain the setting for each type of optical system.

Inheritance diagram for AufDB::OpticalSystem:



Collaboration diagram for AufDB::OpticalSystem:



Static Public Member Functions

- static OpticalSystem GetInstanceOf (OpticalSystemType opSysType)

Protected Attributes

- OpticalSystemType _type
- int _secondsDuration

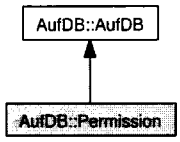
Properties

- OpticalSystemType Type `[get]`
- int SecondsDuration `[get, set]`
- string id `[get]`

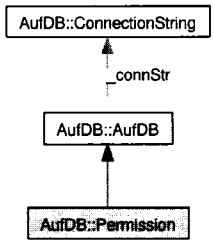
E.34 AufDB::Permission Class Reference

The class represents a possible permission (read, write...) on a db object.

Inheritance diagram for AufDB::Permission:



Collaboration diagram for AufDB::Permission:



Public Member Functions

- Permission ()
- Permission (Permission src)
- Permission (string connStr)
- Permission (long PermissionKey, string connStr)
- Permission (string name, string description, string connStr)

Protected Member Functions

- override void setVariables (IDataReader rdr)
- override void addPKParams (IDbCommand cmd)
- override void addInsertParams (IDbCommand cmd)

Properties

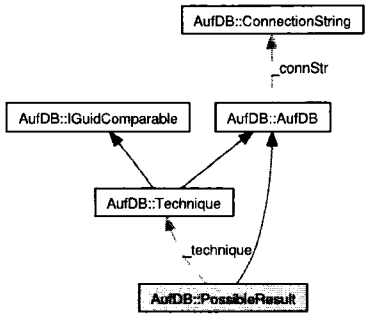
- string Name `[get, set]`
- string description `[get, set]`

E.35 AufDB::PossibleResult Class Reference

Represents a result that might be generated by application of a Technique.

Inheritance diagram for AufDB::PossibleResult:

Collaboration diagram for AufDB::PossibleResult:



Public Member Functions

- void SetTechnique (Technique technique)
- PossibleResult ()
- PossibleResult (string connstr)
- PossibleResult (long possibleResultID, string connstr)
- PossibleResult (long possibleResultID, Technique technique, string name, float upperBound, float lowerBound, System.DateTime dateUpdated, string unitName, string connstr)
- PossibleResult (PossibleResult pr)
- PossibleResult (Xml.PossibleResultcontent xPossibleResult, string connstr)
- override void dbWrite (IDbConnection conn)
- override int CompareTo (object obj)

Protected Member Functions

- override void checkAndSet< T > (ref T var, T val)
- override void setVariables (IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)
- void validateState ()

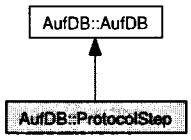
Properties

- string Name `[get, set]`
- float UpperBound `[get, set]`
- float LowerBound `[get, set]`
- string UnitName `[get, set]`
- Technique Technique `[get]`

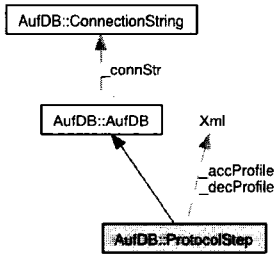
E.36 AufDB::ProtocolStep Class Reference

This class represents a specific step in a centrifuge protocol.

Inheritance diagram for AufDB::ProtocolStep:



Collaboration diagram for AufDB::ProtocolStep:



Public Member Functions

- ProtocolStep (int Number, int RPM, float degreesC)
- void AddOpticalSystem (OpticalSystem opsys)

Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

Properties

- int DelayAfterTempReached `[get, set]`
- bool WaitForTemperature `[get, set]`
- Xml.AccelerationProfile AccelProfile `[get, set]`
- Xml.DecelerationProfile DecelProfile `[get, set]`
- List< OpticalSystem > OpticalSystems `[get]`
- int RPM `[get]`
- int Number `[get]`
- float Temperature `[get]`

E.37 AufDB::ReadOnlyAbsChannelCollection Class Reference

This class creates a read-only collection of channels that can be indexed with a [channelname] accessor.

Public Member Functions

- ReadOnlyAbsChannelCollection (System.Collections.ObjectModel.ReadOnlyCollection< AbstractChannel > absChannels)

Properties

- AbstractChannel this [string channelName] `[get]`

E.38 AufDB::ReadOnlyCellCollection Class Reference

This is a read-only collection of Cell objects. It is indexed by hole number.

Public Member Functions

- ReadOnlyCellCollection (System.Collections.ObjectModel.ReadOnlyCollection< Cell > cells)

Properties

- new Cell this [int holeNumber] `[get]`

E.39 AufDB::ReadOnlyChannelCollection Class Reference

A read-only collection of channels, indexed by channel number (1 based), or channel name (A, B, C ...).

Public Member Functions

- ReadOnlyChannelCollection (System.Collections.ObjectModel.ReadOnlyCollection< Channel > channels)

Properties

- new Channel this [int channelNum] `[get]`
- Channel this [string channelName] `[get]`

E.40 AufDB::AufDB::ReadOnlyComponentCollection< T > Class Reference

This class is a read only collection of solution components.

Public Member Functions

- ReadOnlyComponentCollection (System.Collections.ObjectModel.ReadOnlyCollection< T > Components)

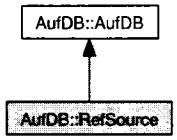
Properties

- T this [T c] `[get]`

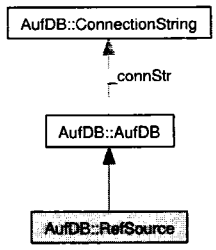
E.41 AufDB::RefSource Class Reference

This class represents an reference to arbitrary external information.

Inheritance diagram for AufDB::RefSource:



Collaboration diagram for AufDB::RefSource:



Public Member Functions

- RefSource ()
- RefSource (string connstr)
- RefSource (long id, string connstr)
- RefSource (long id, string name, string uri, string connstr)
- RefSource (RefSource src)

Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

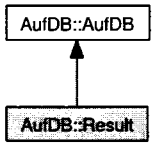
Properties

- string ReferenceUri `[get, set]`
- string ReferenceName `[get, set]`

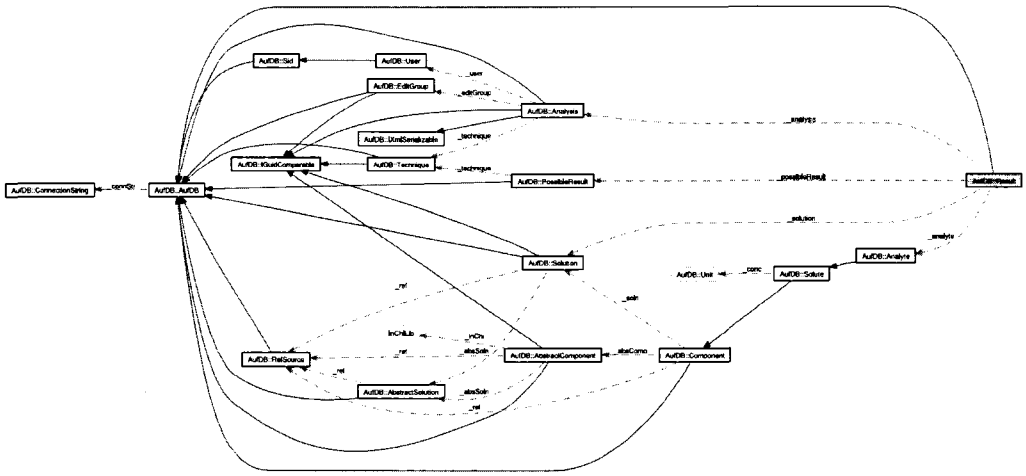
E.42 AufDB::Result Class Reference

This class represent a result produceds by application of an analysis technique to an edit group.

Inheritance diagram for AufDB::Result:



Collaboration diagram for AufDB::Result:



Public Member Functions

- Result ()
- Result (Result src)
- Result (Xml.AnalysisResultcontent xResult, string connstr)
- Result (string connStr)
- Result (long resultKey, string connstr)
- Result (int resultID, int possibleResultID, Analysis analysis, int analyteID, int solutionID, double value, float upperCI, float lowerCI, DateTime dateUpdated, string connStr)
- Result (long possibleResultID, Analysis analysis, Analyte analyte, Solution solution, double value, float upperCI, float lowerCI, string connStr)
- Result (PossibleResult possibleResult, Analysis analysis, Analyte analyte, Solution solution, double value, string connStr)
- void setAnalysis (Analysis analysis)
- override int CompareTo (object obj)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- override void setVariables (IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

Properties

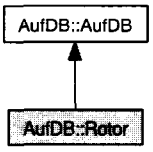
- PossibleResult possibleResult `[get, set]`
- Analysis analysis `[get, set]`

- Analyte Analyte `[get, set]`
- Solution Solution `[get, set]`
- double value `[get, set]`
- float UpperCI `[get, set]`
- float LowerCI `[get, set]`

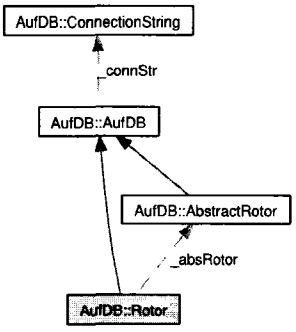
E.43 AufDB::Rotor Class Reference

This class encapsulates the properties of a physical AUF rotor.

Inheritance diagram for AufDB::Rotor:



Collaboration diagram for AufDB::Rotor:



Public Member Functions

- Rotor ()
- Rotor (string connStr)
- Rotor (string name, string serialNumber, string stretchFunction, AbstractRotor absRotor, string connstr)
- Rotor (Rotor src)
- Rotor (long key, string connstr)
- Rotor (IDataReader rdr, string connstr)
- override void dbWrite (IDbConnection conn)
- override string ToString ()

Protected Member Functions

- override void setVariables (System.Data.IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

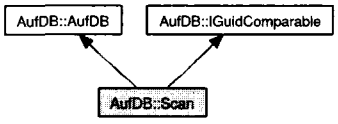
Properties

- string Name `[get, set]`
- AbstractRotor AbstractRotor `[get, set]`
- string SerialNumber `[get, set]`
- string StretchFunction `[get, set]`

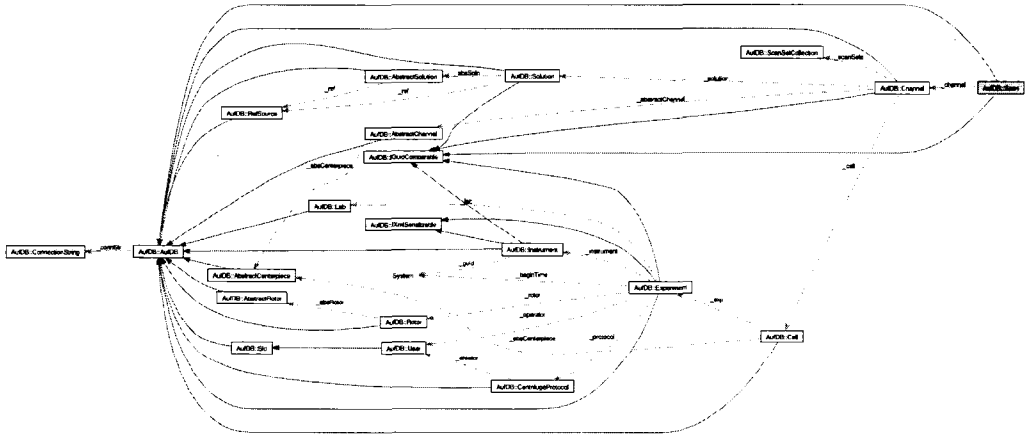
E.44 AufDB::Scan Class Reference

This class represents a raw scan acquired by an analytical ultracentrifuge.

Inheritance diagram for AufDB::Scan:



Collaboration diagram for AufDB::Scan:



Public Member Functions

- Scan (string connstr)
- Scan (Scan src)
- Scan (int Number, Channel channel, byte[ ] content, string connStr)
- Scan (int Number, Channel channel, System.IO.MemoryStream ms, string connStr)
- Scan (int Number, Channel channel, string ASCIIcontent, string connStr)
- Scan (long scanKey, string connStr)
- Scan (Channel c, long scanKey, string connStr)
- Scan (Xml.EditedScancontent xScan, string connstr)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

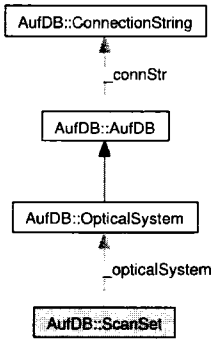
- void setScanFields (Scan src)
- override void setVariables (IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

Properties

- System.IO.MemoryStream Content `[get, set]`
- int Number `[get, set]`
- Channel Channel `[get, set]`
- long ChannelKey `[get]`
- DateTime BeginTime `[get, set]`
- DateTime EndTime `[get, set]`
- float Temperature `[get, set]`
- long Rpm `[get, set]`
- double Omega2t `[get, set]`
- Guid Guid `[get, set]`
- long Seconds `[get, set]`

E.45 AufDB::ScanSet Class Reference

Collaboration diagram for AufDB::ScanSet:



Properties

- OpticalSystemType OpticalSystemType `[get, set]`
- OpticalSystem OpticalSystem `[get, set]`
- List< Scan > Scans `[get, set]`

E.46 AufDB::ScanSetCollection Class Reference

Public Member Functions

- ScanSetCollection ()

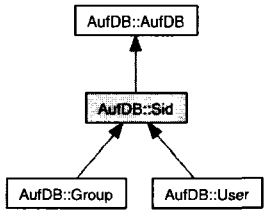
Properties

- ScanSet this [OpticalSystemType opSysType] `[get]`

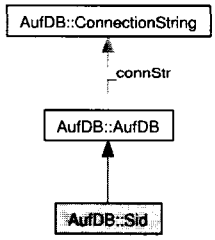
E.47 AufDB::Sid Class Reference

This class represents a user or a group. All access control entries are related to it.

Inheritance diagram for AufDB::Sid:



Collaboration diagram for AufDB::Sid:



Public Member Functions

- Sid ()
- Sid (string connStr)
- Sid (Sid src)
- Sid (long SIDKey, string connStr)

Static Public Member Functions

- static Sid SidBuilder (long SidKey, Type sidType, string connStr)

Protected Member Functions

- override void addPKParams (IDbCommand cmd)
- override void addInsertParams (IDbCommand cmd)
- override void setVariables (IDataReader rdr)

Protected Attributes

- string _name

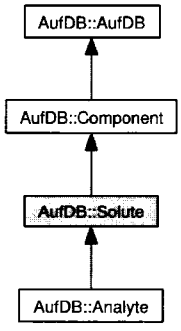
Properties

- virtual string Name `[get, set]`

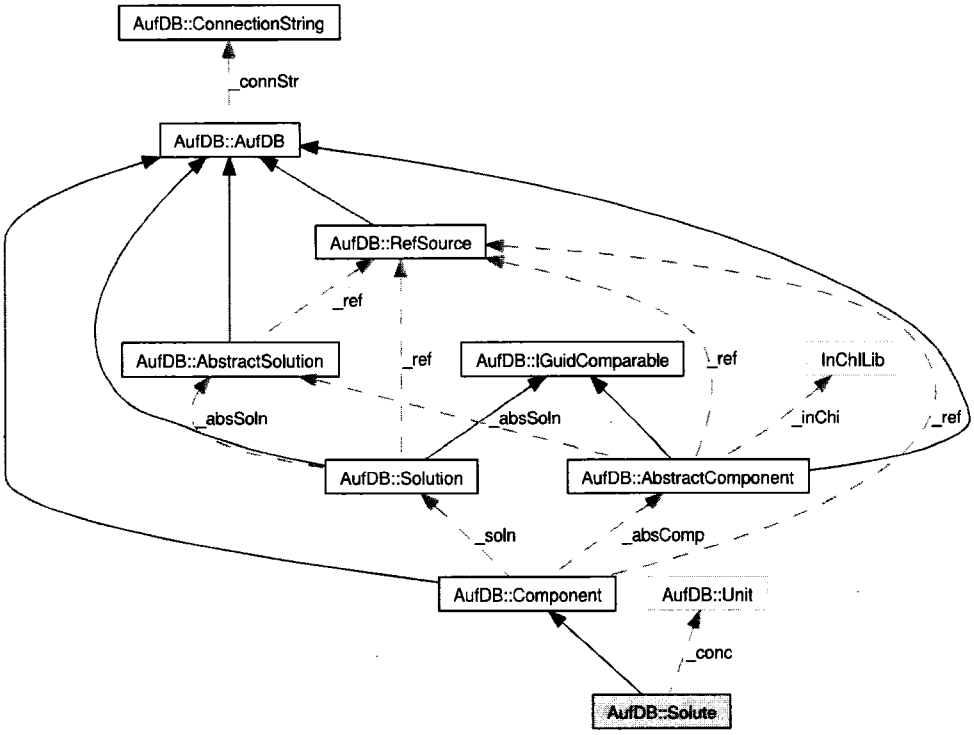
E.48 AufDB::Solute Class Reference

This class represents a solute component in a specific batch of a solution.

Inheritance diagram for AufDB::Solute:



Collaboration diagram for AufDB::Solute:



Public Member Functions

- Solute (string connstr)
- Solute (AbstractComponent ac, double mg)
- Solute (Solute src)
- Solute (long key, string connstr)
- Solute (long key, AbstractComponent ac, string name, string lotid, DateTime dateCreated, DateTime dateExpired, string connstr)
- Solute (string name, float molecularWeight, string connstr)
- Solute (IDataReader rdr, string connstr)

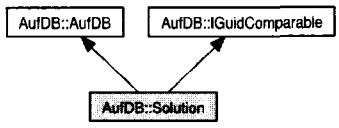
Properties

- Unit.Concentration Concentration `[get, set]`
- bool ConcentrationSet `[get]`

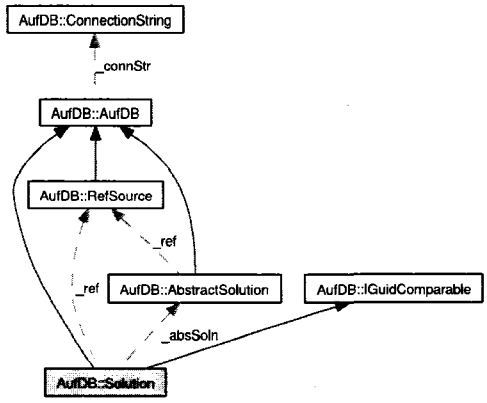
E.49 AufDB::Solution Class Reference

This class handles conversions of the various method of defining solutions (mass %, volume %, molality, molarity, weight/volume) to a single standard milligram/milligram measurement.

Inheritance diagram for AufDB::Solution:



Collaboration diagram for AufDB::Solution:



Public Member Functions

- Solution ()
- Solution (Solution src)
- Solution (long id, string connstr)
- Solution (string connstr)
- Solution (AbstractSolution absSoln, Unit.Mass totalSolutionMass)
- Solution (AbstractSolution absSoln, Unit.Volume totalSolutionVolume)
- Solution (IDataReader rdr, string _connstr)
- void RegisterChannel (Channel ch)
- override string ToString ()
- string ContentString ()
- void AddSolvent (Solvent sc)
- void AddSolvent (Solvent sc, float amount, Unit.MassUnit unit)
- void AddSolvent (Solvent sc, float amount, Unit.VolumeUnit unit)
- void AddSolute (Solute sc)
- void AddAnalyte (Analyte sc)

- void AddSolute (Solute sc, Unit.Concentration conc)
- void AddAnalyte (Analyte sc, Unit.Concentration conc)
- void AddSolute (Solute sc, float amount, Unit.MassUnit unit)
- void AddAnalyte (Analyte sc, float amount, Unit.MolarUnit unit)
- bool RemoveComponent (Component component)
- bool EditsAllowed ()
- bool EditsAllowed (Channel ch)
- void AllowEdits ()
- override void dbWrite (System.Data.IDbConnection conn)

Protected Member Functions

- override void addInsertParams (System.Data.IDbCommand cmd)
- override void addPKParams (System.Data.IDbCommand cmd)
- override void setVariables (System.Data.IDataReader rdr)

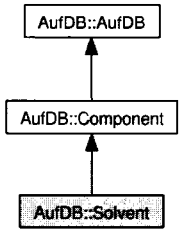
Properties

- RefSource RefSource `[get, set]`
- string Name `[get, set]`
- Guid Guid `[get, set]`
- string LotIdentifier `[get, set]`
- DateTime DateCreated `[get, set]`
- DateTime DateExpired `[get, set]`
- ReadOnlyComponentCollection< Solute > SoluteComponents `[get]`
- ReadOnlyComponentCollection< Analyte > AnalyteComponents `[get]`
- ReadOnlyComponentCollection< Solvent > SolventComponents `[get]`
- AbstractSolution AbstractSolution `[get]`

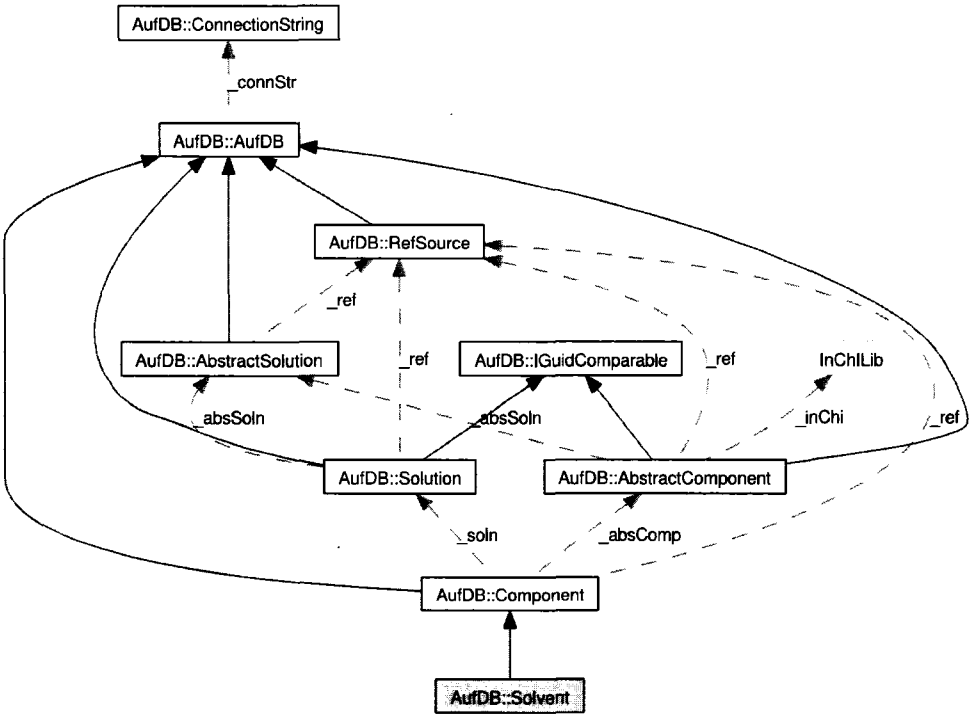
E.50 AufDB::Solvent Class Reference

This class represents a solvent component (e.g. water) in a specific batch of a solution.

Inheritance diagram for AufDB::Solvent:



Collaboration diagram for AufDB::Solvent:



Public Member Functions

- Solvent (string connstr)
- Solvent (Solvent src)
- Solvent (AbstractComponent ac, double mg)
- Solvent (long key, string connstr)
- Solvent (long key, AbstractComponent ac, string name, string lotid, DateTime dateCreated, DateTime dateExpired, string connstr)
- Solvent (string name, float molecularWeight, string connstr)
- Solvent (IDataReader rdr, string connstr)

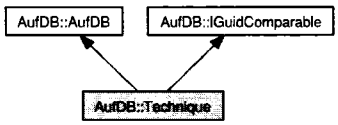
Properties

- float BulkDensity `[get, set]`

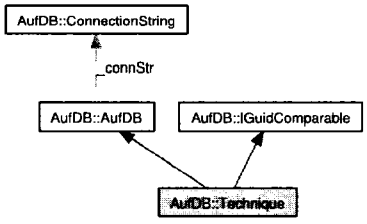
E.51 AufDB::Technique Class Reference

The technique class represents an technique used to analyze data files eg: dc/dt.

Inheritance diagram for AufDB::Technique:



Collaboration diagram for AufDB::Technique:



Public Member Functions

- Technique ()
- Technique (Technique src)
- Technique (string connstr)
- Technique (string name, string connstr)
- Technique (long key, string connstr)
- Technique (Xml.Techniquecontent xTechnique, string connstr)
- override void dbRead (IDbConnection conn)
- void AddPossibleResult (PossibleResult pr)
- void DeletePossibleResult (PossibleResult pr)
- override void dbWrite (IDbConnection conn)
- override void dbDelete (IDbConnection conn)
- void MarkChanged ()

Protected Member Functions

- override void setVariables (IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

Properties

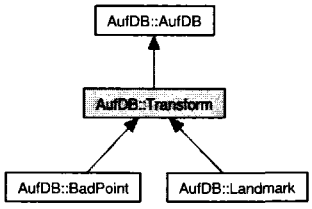
- string Name `[get, set]`

- Guid Guid `[get, set]`
- string Version `[get, set]`
- System.Collections.ObjectModel.ReadOnlyCollection< PossibleResult > PossibleResults `[get]`

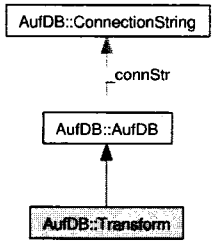
E.52 AufDB::Transform Class Reference

The transform class represents an operation that modifies raw scan data (e.g. eliminate bad points, mark a meniscus position).

Inheritance diagram for AufDB::Transform:



Collaboration diagram for AufDB::Transform:



Public Member Functions

- Transform (Transform transform)
- Transform (string connstr)
- Transform (long key, string connstr)

Protected Member Functions

- Transform ()
- override void setVariables (System.Data.IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

Protected Attributes

- string _name
- int _pointNumber
- float _radialPosition

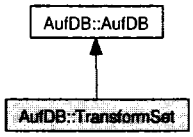
Properties

- string Notes `[get, set]`
- string Name `[get, set]`

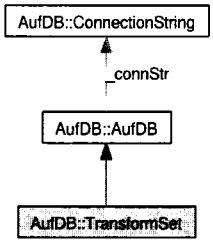
E.53 AufDB::TransformSet Class Reference

This class represents a set of transforms applied to a scan to derive an EditedScan.

Inheritance diagram for AufDB::TransformSet:



Collaboration diagram for AufDB::TransformSet:



Public Member Functions

- TransformSet ()
- TransformSet (String connstr)
- TransformSet (long key, String connstr)
- TransformSet (Transform[ ] transforms, String connstr)
- void addTransform (Transform newTransform)
- void removeTransform (Transform transformToRemove)

- override void dbRead (IDbConnection conn)
- override void dbWrite (IDbConnection conn)

Protected Member Functions

- void addTransforms (IDataReader rdr)
- override void setVariables (IDataReader rdr)
- override void addInsertParams (IDbCommand cmd)
- override void addPKParams (IDbCommand cmd)

Properties

- string Name `[get, set]`
- System.Collections.ObjectModel.ReadOnlyCollection< Transform > Transforms `[get]`

E.54 AufDB::Unit::Concentration Class Reference

Represents a concentration in molar or weight/volume units.

Public Member Functions

- Concentration (double value, MassUnit mu, VolumeUnit vu)
- Concentration (double value, MolarUnit mu)
- double GetMg (double solutionMl)
- double GetMg (double mw, double solutionMl)
- override string ToString ()

Static Public Member Functions

- static string ShortMassName (MassUnit u)
- static string ShortVolumeName (VolumeUnit u)
- static string ShortMoleName (MolarUnit u)

Properties

- bool IsMolar `[get]`
- double MgPerMl `[get]`
- double MolePerLiter `[get]`
- double Molarity `[get]`
- MassUnit MassUnit `[get, set]`
- VolumeUnit VolumeUnit `[get, set]`
- MolarUnit MolarUnit `[get, set]`

E.55 AufDB::Unit::Mass Class Reference

Allows conversions of various mass units.

Public Member Functions

- Mass (double value, MassUnit mu)

Properties

- double ToMg `[get]`
- double ToGram `[get]`

E.56 AufDB::Unit::Volume Class Reference

Allows conversion of various volume units.

Public Member Functions

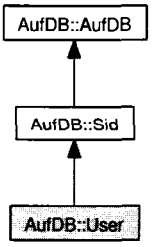
- Volume (double value, VolumeUnit mu)

Properties

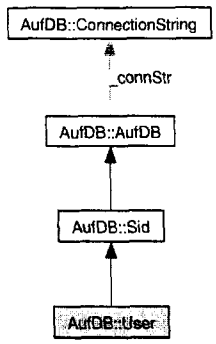
- double TomL `[get]`

E.57 AufDB::User Class Reference

User class represents a user of this system, and stores their authentication credentials.
Inheritance diagram for AufDB::User:



Collaboration diagram for AufDB::User:



Public Member Functions

- User ()
- User (User src)
- User (Xml.Personcontent xUser, string connstr)
- User (string connstr)
- User (long SIDKey, string connstr)
- User (long SIDKey, string name, System.Security.Cryptography.X509Certificates.X509Certificate x509cert, string connstr)
- User (long SIDKey, string name, byte[ ] rawCert, string connstr)
- User (X509Certificate x509cert, string connstr)
- User (IDataReader rdr, string connstr)
- override void dbWrite (System.Data.IDbConnection conn)
- void MakeCertFromValues (string issuerName, RSA signingKey, DateTime expirationDate)
- override string ToString ()

Protected Member Functions

- override void addInsertParams (IDbCommand cmd)
- override void setVariables (IDataReader rdr)
- void MakeCertFromBytes (byte[ ] rawcert)

Properties

- string FirstName `[get, set]`
- string LastName `[get, set]`
- override string Name `[get, set]`
- string Institution `[get, set]`
- X509Certificate x509cert `[get, set]`
- byte[ ] AccessKey `[get, set]`
- RSA PrivateKey `[get, set]`