A Generic Acoustics Prediction Model Engine

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Travis J. Fischer

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2010

\

A Generic Acoustics Prediction Model Engine

By Travis J. Fischer

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Kenny Hunt Examination Committee Chairperson	Date
Dr. Mark Headington Examination Committee Member	Date
Dr. Mao Zheng Examination Committee Member	Date

ABSTRACT

FISCHER, TRAVIS, J., "A Generic Acoustics Prediction Model Engine", Master of Software Engineering, 05 2010, Dr. Kenny Hunt.

Trane Company in La Crosse, WI has a continual need for software systems that can calculate predictions of the acoustic performance for their air conditioning products. A suite of acoustic prediction software tools has been developed and maintained at Trane over the last 20 years. Over time these legacy software tools have become cumbersome and expensive to maintain. This manuscript describes the development of a software system designed to make these acoustics prediction calculations for various Trane air conditioning products. The software system includes a software library with a public API that will be used by various Trane software systems in order to make acoustic calculations. The system also includes a graphical user interface application that will be deployed to Trane engineers and sales people in order to allow them to more easily make acoustic calculations.

ACKNOWLEDGEMENTS

I would like to thank my faculty advisor, Dr. Kenny Hunt, for his valuable guidance and insight on this Capstone project. I would like to thank the UW-L MSE faculty whom I have studied under, including Dr. Kasi Periyasamy and Dr. Mao Zheng, for skillfully educating me in the practices of quality software engineering. I would like to sincerely thank all of the tremendously talented people at Trane Company who have directly supported my work on this project and allowed me to dedicate an entire year of my internship to the project. I would like to thank Vikas Patnaik, Engineering Manager for Global Modeling & Analysis, for his sponsorship and continual support of the project. I would like to thank Steve Lind, Acoustic Engineer for Climate Solutions, for supporting and guiding this project on a daily basis. I would also like to thank both the Acoustics and Vibration Engineering group and the Global Modeling and Analysis group at Trane for their time and support of this project. I would like to thank my parents for contributing to, supporting and encouraging my education over the last 17 years. I would like thank my wife, Melissa, for her unwavering support of my academic journey and her patience with the amount of time and energy I had to give in order to complete this project.

TABLE OF CONTENTS

LIST OF FIGURES/TABLES

GLOSSARY

Acoustic Silencer

A device installed in heating and air conditioning systems used to attenuate the sound power exiting the unit by absorbing some of the sound.

API

An Application Program Interface (API) is an interface that provides access to an underlying software system of services.

CLCHLw

A software program created and owned by Trane that allows users to calculate predicted sound power levels for a group of air handling products.

CSAA

The name of a new Trane product line of air handling units.

Insertion Loss

The sound level reduction at a given location due to the insertion of a noise control device, expressed in decibels.

Intellipak II

A product line of Trane rooftop air conditioning units.

Microsoft® .Net Framework

A software framework made by Microsoft that can be installed on computers running the Windows® operating system. The framework includes a library of code that can be used by developers to solve common programming problems as well as a virtual machine that manages the execution of programs written specifically for the framework.

Microsoft® Access®

A relational database management system product from Microsoft that combines the relational Jet Database Engine with a GUI and a collection of development tools

NDepend

A code analysis and metrics software tool designed to be used with code written for the .Net Framework.

NUnit

An open source unit testing framework from the xUnit family of unit testing tools. NUnit is designed to be used with Microsoft .NET and works similar to the popular Java testing tool JUnit.

Regenerated Noise

Noise that is generated on the “quiet” side of sound attenuators which is the result of concentrated air flow turbulence.

Sound Power Level

A logarithmic measure of sound power in comparison to a specified reference level. A sound power level is abbreviated as PWL with units dB re 1 pW, which is in contrast to a sound pressure level which is abbreviated as SPL with units dB re 20 μPa.

SQL

Server Query Language is a standard language used for accessing and modifying databases.

SQL Server CE®

Microsoft’s SQL Server Compact Edition is a compact relational database that is designed to be used with applications that run on mobile devices and desktops.

TOPSS

The Trane Official Product Selection System (TOPSS) is a Windows program that contains information to select and predict performance of Trane products operating under various conditions.

UML

A standardized general-purpose modeling language created by the Object Management Group and used extensively in the field of software engineering.

Windows® Forms Framework

A graphical application programming interface that is included as a part of Microsoft's .NET Framework. The framework provides access to the native Microsoft Windows interface elements by encapsulating the existing Windows API in managed code.

1. Introduction

1.1 Background Information

Trane, a business of Ingersoll Rand, has a continuous business need for the ability to model the acoustic properties of their air conditioning products. These products come in a multitude of applications, sizes and configurations. When they are installed and operated in a building they produce certain levels of sound and vibration which can be either desirable or undesirable depending on the situation. Each of Trane's products has certain physical properties which can be modeled in software in order to determine the sound power levels (Lw) that will be generated while that equipment is operating.

Historically these sound power prediction calculations have been done manually by acoustical engineers using tables of data and well defined formulas. In order to reduce the amount of time spent on these prediction calculations and the number of errors made during the manual process, Trane has developed software models which can be used to compute these predictions automatically. Over the last 20-25 years, engineers and interns at Trane have developed and maintained eight individual acoustic software models. Each of these software programs models a specific Trane air conditioning product.

Each of these software models takes some number of input variables which describe the physical characteristics of the product, retrieves acoustics test data from a data store and then executes an algorithm which calculates a set of acoustics related outputs. The key output in each of these models is a collection of data arrays which contain sound power levels across a twenty-four frequency band sound spectrum.

Many of the core equations and algorithms used in these calculations are common among the various acoustic models. For this reason, many of the programs share common software components. The software development process that was used while developing these projects was unorganized, haphazard and did not use good software design principles. Software reuse was not included as a goal in the development of these acoustic software projects. Because of the short-sighted design that was used in these systems, efforts towards software re-use and component sharing have resulted in a very poor collection of highly coupled and un-cohesive software modules. A lot of the code in these models is replicated multiple times across the various software programs. Clearly defined layers of abstraction were not included in the design of these legacy systems. Business logic, GUI logic and database logic were inter-mingled throughout all layers of the software in a haphazard fashion. This has created unnecessarily complicated dependencies among the various components. All of these issues have resulted in a moderately large base of source code that has become extremely difficult and expensive to maintain.

Another issue with the legacy software is that, while the engineering practices of the acoustical engineers at Trane has been greatly improved over the last 20 years; the design of the software models has not kept up. Many pieces of legacy software contain algorithms that are no longer relevant in Trane's sound modeling practices.

The ideal acoustics modeling software would have a well designed architecture that would make efficient use of common logic of acoustic sound models. This system would contain a robust core library of that would provide the functionalities needed for each of the acoustic prediction models. This system would make the process of adding models for new products a simple and low risk task.

1.2 Project Goals

The goal of this project was to design a single core software library for Trane acoustics modeling while making use of good software engineering practices. This core library was designed to include all the functionality included in previous Trane acoustic models and also to make improvements in the design and implementation of the software. This was done in order to improve the efficiency of program maintenance as well as the overall robustness of the prediction software. This project also included implementing a product specific extension of the core acoustics library for one of Trane's new air handling products.

In order to create an optimal design for the Trane acoustics software library, the project required an in-depth analysis of both the legacy acoustic models as well as the long term vision for Trane acoustics software. One of the project goals was to provide documentation on findings of this analysis for use in future acoustics software engineering efforts.

Software quality assurance standards and processes have only been recently implemented in Trane's software groups. Another goal of this project was to make use of basic testing tools and frameworks while working with one of Trane's software quality assurance employees to set up a test plan for unit, integration, system and regression testing.

The legacy acoustic models relied on product test data that was stored in a Microsoft® Access® database. This database was built up in an ad-hoc manner as these models were developed which resulted in an inefficient and poorly designed database structure. Another goal of this project was to design a well documented database that would improve the maintenance of the database and simplify maintaining the code that interfaces with this database.

The legacy acoustic models were written in the Fortran, Basic and Visual Basic programming languages. The most heavily used piece of the legacy models was written in Visual Basic 6 which proved problematic since Microsoft stopped supporting this technology in 2008. Another goal of this project was to write the acoustics library exploiting the object oriented aspects of the domain using the C# programming language on the Microsoft® .Net framework.

The final goal of this project was to develop and document a future vision for Trane acoustics software. This vision would be documented in the form of a set of requirements for future enhancements to the acoustics library and the supporting software tools.

1.3 Software Lifecycle Model

There are many software lifecycle models that are recorded in software engineering literature [1] and are currently used in the development of software projects around the world. Trane does not have a company-wide standard lifecycle model that is used by developers across all the various software groups. Each individual group uses a unique lifecycle model that they have adopted through years of development experience. The group that owns the acoustics software does not have a defined lifecycle model although a hybrid lifecycle model similar to an iterative model is what is most commonly used.

In order to select a development model that would be used for this project, several lifecycle models, including waterfall, spiral, iterative and prototyping, where considered and measured against project objectives and environment. Ultimately, an iterative lifecycle model was selected for this project as it was found to be the best fit for a variety of reasons.

It is well known at Trane that developing software within the organization requires an extremely fluid and flexible process. Trane is a very large organization and there are many players involved in the development of any software project. Every one of these people brings their own set of requests and requirements to the project. Every software project developed at Trane is subject to constant change and refinement as the business needs, priorities and personnel change around it. In the case of the acoustics software models, there are three distinct groups of internal customers (or users) who each have their own set of requirements. Each of these groups is also subject to its own set of customers (both internal and external) who impact the requirements. These factors make a waterfall model impractical as any comprehensive set of initial requirements would quickly become irrelevant at Trane.

Another factor that impacted the selection of a lifecycle model is the speed at which Trane products are designed, re-designed and released. Trane's business model dictates that acoustics modeling software must be available for the release of a new product. This means that the acoustics software is developed largely in-parallel with the testing and configuration of a new product. The result is that some of the requirements for a software model are not known until literally a few months before it must be released. This factor also makes a waterfall model unusable.

A spiral lifecycle model seems well suited for the acoustics library project due to is iterative nature and short feedback loop. However, it was determined that the support necessary to effectively employ a spiral model would not be available for this project at Trane. Continuous risk analysis is a key component of the spiral model. This risk analysis process requires project customers and development team leaders to give continual feedback on the risk of continuing the project. Trane was not able to provide the resources, in terms of internal customer time, to make a spiral model an effective choice.

In a prototyping lifecycle model, developers create a throw-away prototype of the system in order to get a better understanding of the system requirements and to help customers define what their requirements are. The prototyping model was not selected for this project because this project was based on an existing system. Therefore, it was more time effective to take the requirements for this project from the existing system rather than to build a throw-away prototype which would closely mimic the existing system.

An iterative lifecycle model was determined to be ideal for this project because it would allow continual progression on the project with incremental feedback and change requests from the various internal customer groups. The iterative lifecycle started by taking the initial system requirements from the existing system then allowed changes in the system requirements as the new Trane product was developed. The initial project plan included 4 iterations through the standard lifecycle phases of requirements, design, implementation and testing [1, 2]. The number of iterations that took place during the course of the project ended up being approximately seven. The reason for the increase in the number iterations was due to an increase in number of times that requirements changed from what was initially envisioned. This change forced the development loop to be tighter than was anticipated and resulted in a few additional iterations. Throughout the project there was a continuous flow of requirement changes, which were then folded into the design and implementation of the project and finally tested. During each cycle, requirement changes were collected and compiled into a list of requests. Sometimes these changes were added to the current cycle and sometimes they were added to a future cycle or to a "wish list" of future development efforts.

The iterative lifecycle proved to be very effective for this project primarily because it allowed the development of the system to keep up with the rapid change in requirements. It also provided reoccurring feedback in the testing phases which allowed for many helpful changes in the system design for future cycles.

2 Requirements

The initial phase of every cycle in the iterative lifecycle model is the requirements gathering and refining phase. This phase consists of meetings held between the software development team and customers of the software. During these meetings the required functionalities of the software are discussed and defined. In a traditional waterfall lifecycle model all of these meetings take place at the beginning of the software development process so that a well defined requirements document can be generated as the output of the initial stage. In the iterative model used for this project, these meetings take place throughout the duration of the development process.

For this project, the requirements gathering meetings were held beginning in the spring of 2009 and then throughout the development process. Requirements meetings were held with three distinct groups of internal acoustics software customers as well as with Trane acoustic engineers and Trane software modeling groups. An initial set of requirements was gathered from an analysis of the legacy acoustics software systems. These requirements were discussed with the various customers and refined to include only those functionalities that were determined necessary in future acoustics systems. The requirements were further discussed and refined with the TOPSS software development group. This group writes software which interfaces with and depends on the acoustics software systems. Further requirements were gathered from the teams involved with the development, testing and release of a new Trane product that was modeled in the new acoustics calculation library and GUI program.

This initial set of requirements was documented in a formal project requirements document [3] which was reviewed with project customers and the UW-L faculty advisor for this project. The approximate scope of the project was taken from these requirements and was determined to be appropriate for the Capstone project.

During the lifecycle iterations that followed, these requirements were both refined and expanded on. Due to the scope restrictions of a Capstone project many of these requirement expansions where added to a list of future requirement requests. Some new requirements were deemed necessary and were added to the scope of the project. This sometimes required restriction or cancelation of previously documented requirements.

2.1 Functional Requirements

The functional requirement of the acoustics software library consists of 3 basic use cases for each product that is modeled in the acoustics library.
These use cases are:

1. Calculate an acoustics prediction.
2. Save an acoustics prediction
3. Load a saved acoustics prediction.

Some products may have a few variations of the first use case if the unit configuration options for that product are complex and include multiple unit types which need to be modeled. The use case diagram in Figure 1 below shows the basic use cases and the various "actors" involved in the initial implementation of the acoustics library for the new CSAA product.

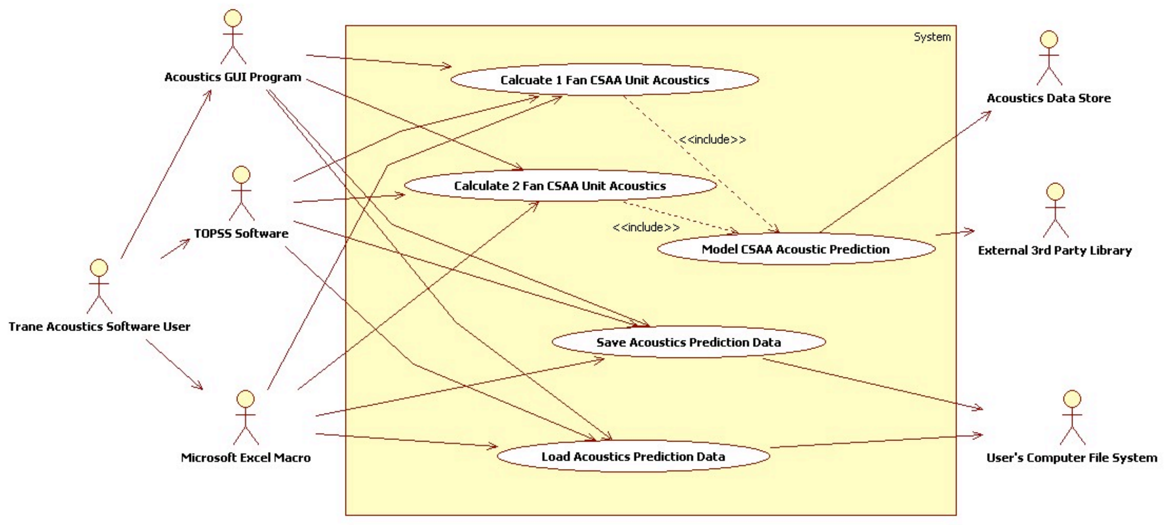

System
Acoustics GUI Program
TOPSS Software
Trane Acoustics Software User
Microsoft Excel Macro
Calcuate 1 Fan CSAA Unit Acoustics
Calculate 2 Fan CSAA Unit Acoustics
<<include>>
<<include>>
Model CSAA Acoustic Prediction
Save Acoustics Prediction Data
Load Acoustics Prediction Data
Acoustics Data Store
External 3rd Party Library
User's Computer File System

Figure 1. CSAA Implementation Use Case Diagram

This use case diagram shows the three actors which directly invoke behavior on the acoustics software system. Each of these actors is an external software system that is used by a Trane acoustics software user in order to calculate acoustics model predictions. The use cases and the APIs for each of these actor systems are identical. They are shown as three separate actors in order to emphasize the need for an API that can support these three independent actors. The right hand side of the diagram displays the three secondary actors that the acoustics software library interacts with.

For the new CSAA implementation of this system, there are two “calculate unit acoustics” use cases that are shown. The first use case calculates a CSAA unit with a single fan, while the second calculates a CSAA unit with two fans. Both of these use cases include a single inner use case which calculates the acoustics generically for any CSAA unit and produces the desired outputs. A very similar use case diagram will exist for each Trane product which eventually gets implemented with the Trane acoustics library.

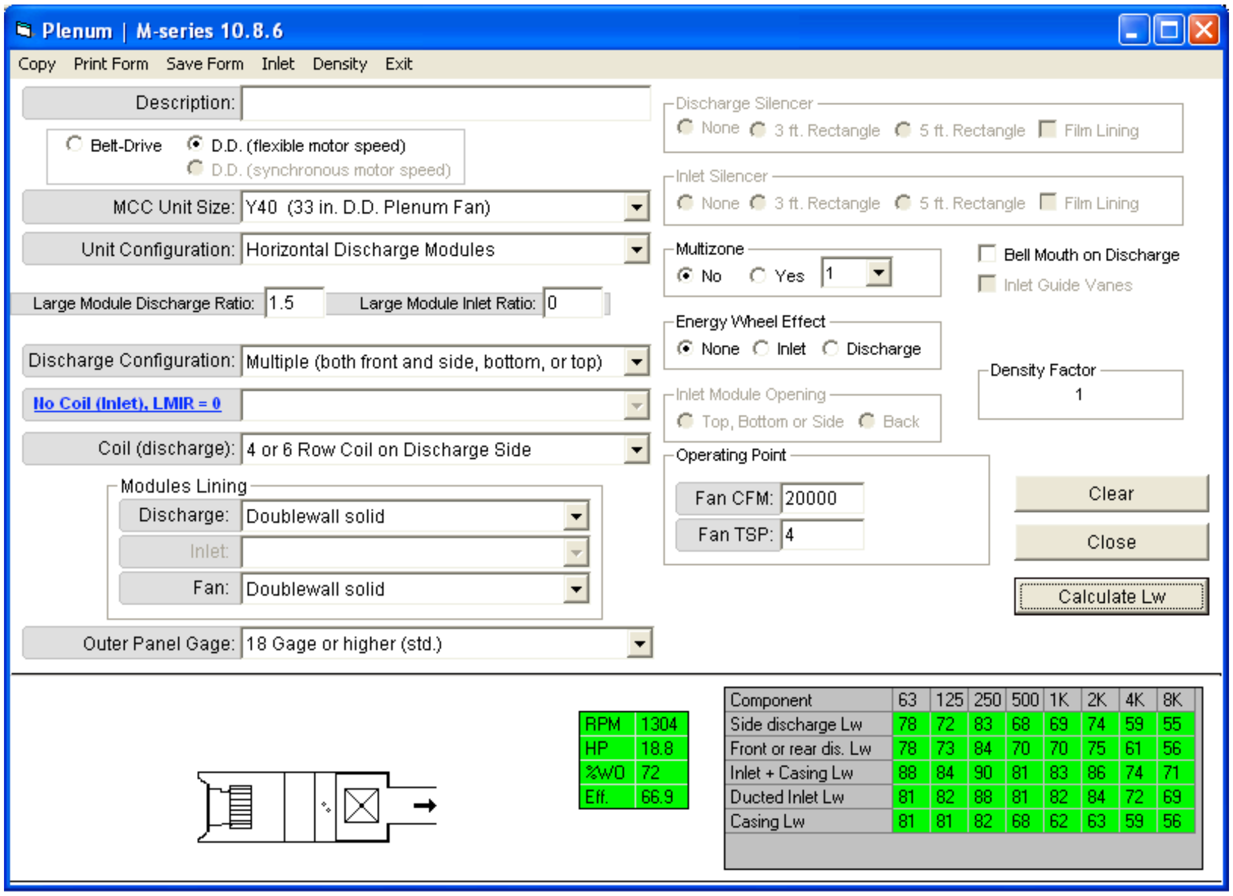
While the number of use cases for each product implementation is quite small the complexity of each use case is very high. This is due to the complexity and number of the inputs as well as the complexity and number of steps involved in completing each of the use cases.

2.2 GUI Requirements

One requirement of the acoustics software project was to design a graphical user interface that would allow users to directly access the functionalities of the acoustics library via some basic input controls and to get a graphical display of the output data that can be easily read and understood. Previous acoustics modeling projects have included a graphical user interface program that accomplished this goal.

The acoustics calculation GUI program is a separate piece of software from the acoustics library. The acoustics library provides all the necessary functionality to do acoustics prediction calculations and GUI interface is purely a graphical interface which uses the acoustics library API to calculate acoustic performance data which is then displayed to the user.

A few examples of the previous GUI programs are shown in the Figures 2 and 3 below.



Description:

Belt-Drive [✓ D.D. (flexible motor speed)] D.D. (synchronous motor speed)

MCC Unit Size: Y40 (33 in. D.D. Plenum Fan)

Unit Configuration: Horizontal Discharge Modules

Large Module Discharge Ratio: 1.5 Large Module Inlet Ratio: 0

Discharge Configuration: Multiple (both front and side, bottom, or top)

No Coil (Inlet), LMIR = 0

Coil (discharge): 4 or 6 Row Coil on Discharge Side

Modules Lining
Discharge: Doublewall solid
Inlet:
Fan: Doublewall solid

Outer Panel Gage: 18 Gage or higher (std.)

Discharge Silencer: None 3 ft. Rectangle 5 ft. Rectangle Film Lining

Inlet Silencer: None 3 ft. Rectangle 5 ft. Rectangle Film Lining

Multizone: [✓ No] Yes 1

Bell Mouth on Discharge
Inlet Guide Vanes

Energy Wheel Effect: [✓ None] Inlet Discharge

Density Factor: 1

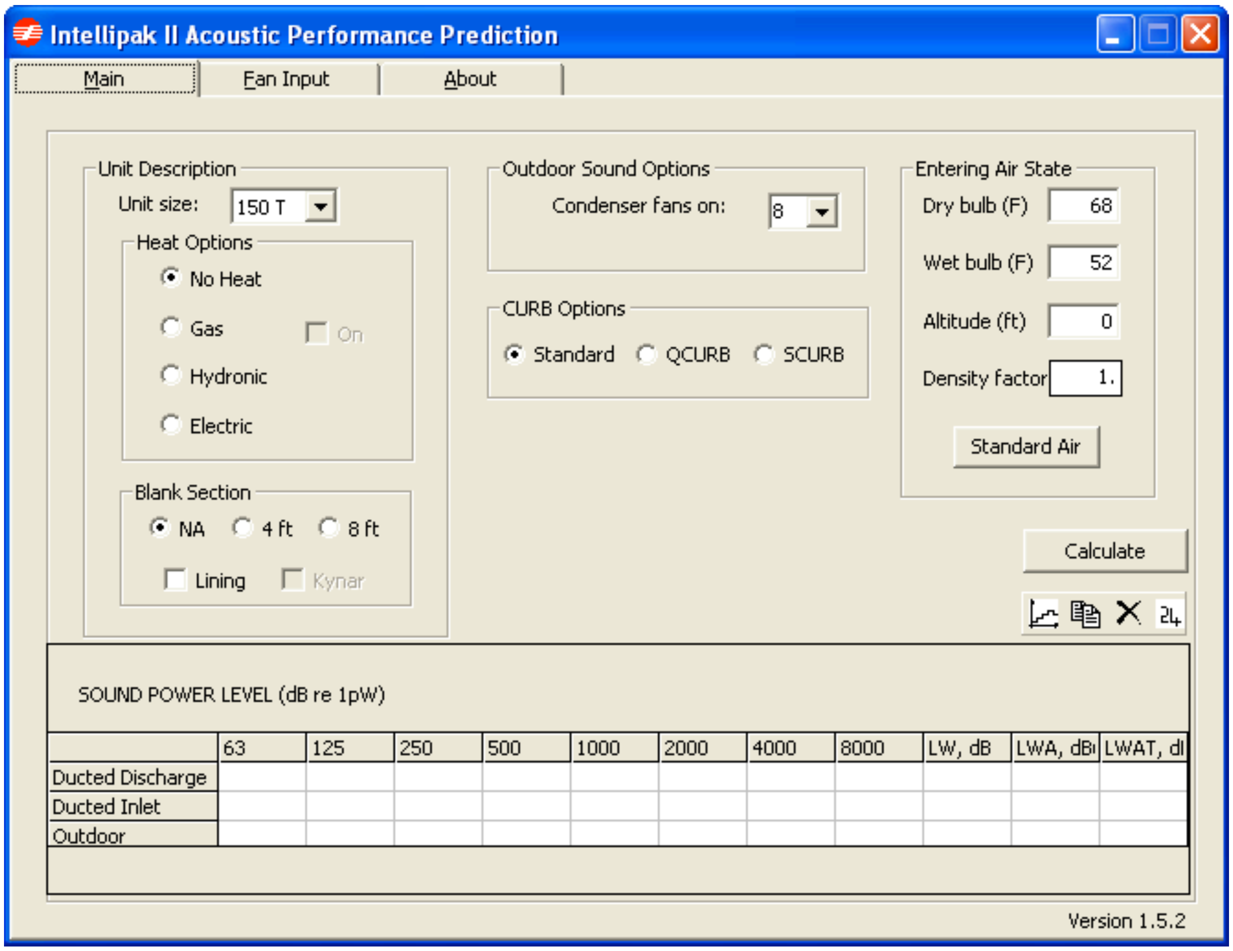
Inlet Module Opening: Top, Bottom or Side Back

Operating Point
Fan CFM: 20000
Fan TSP: 4

RPM	1304
HP	18.8
%WO	72
Eff.	66.9

Component	63	125	250	500	1K	2K	4K	8K
Side discharge Lw	78	72	83	68	69	74	59	55
Front or rear dis. Lw	78	73	84	70	70	75	61	56
Inlet + Casing Lw	88	84	90	81	83	86	74	71
Ducted Inlet Lw	81	82	88	81	82	84	72	69
Casing Lw	81	81	82	68	62	63	59	56

Figure 2. CLCHLw Graphical User Interface Example



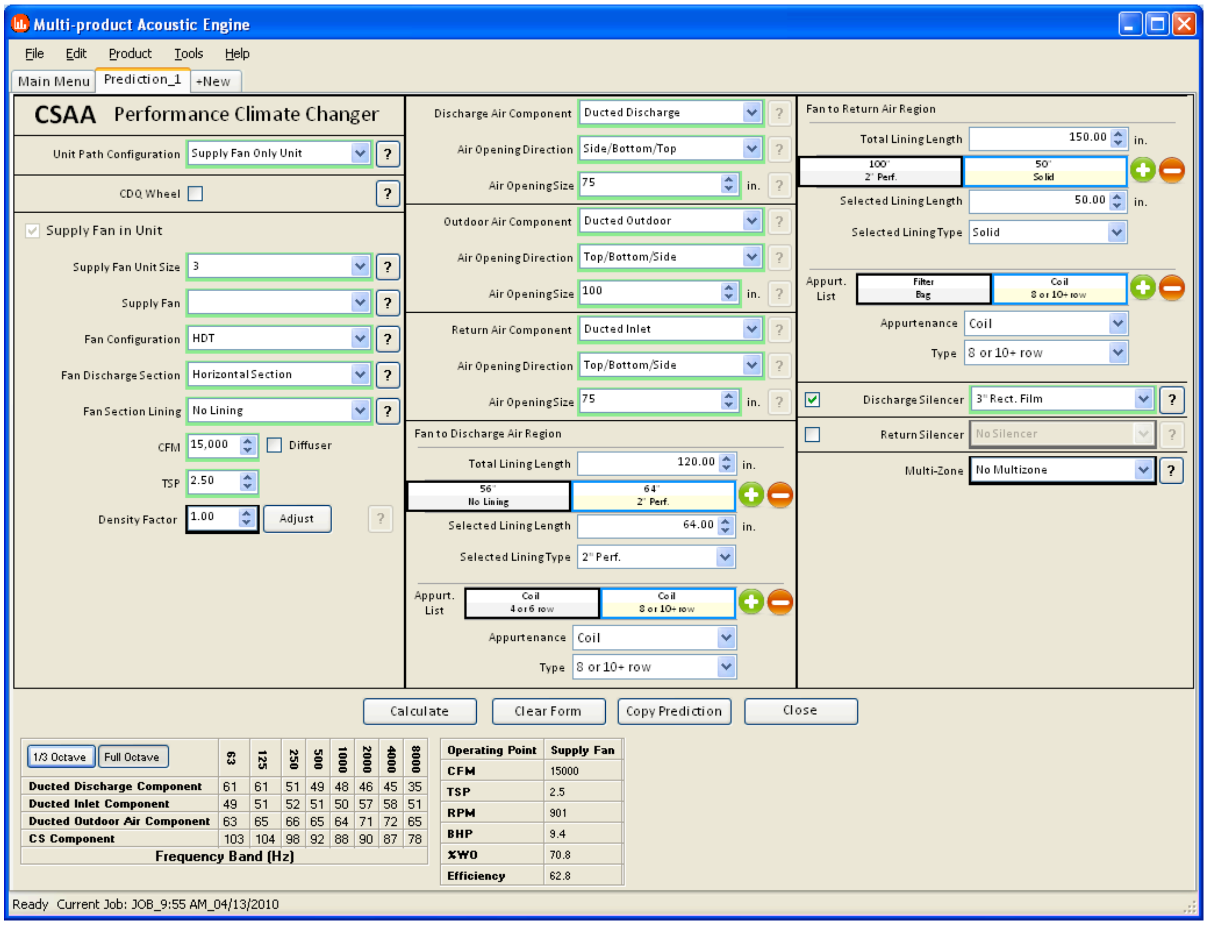
Unit Description
Unit size: 150 T
Heat Options: [✓ No Heat] Gas Hydronic Electric; On
Blank Section: [✓ NA] 4 ft 8 ft; Lining Kynar
Outdoor Sound Options
Condenser fans on: 8
CURB Options: [✓ Standard] QCURB SCURB
Entering Air State
Dry bulb (F): 68
Wet bulb (F): 52
Altitude (ft): 0
Density factor: 1.

SOUND POWER LEVEL (dB re 1pW)

	63	125	250	500	1000	2000	4000	8000	LW, dB	LWA, dB	LWAT, dI
Ducted Discharge											
Ducted Inlet											
Outdoor											

Figure 3. Intellipak II Graphical User Interface Example

The ideal acoustic calculation GUI program would be designed with enough flexibility that it could be used for multiple Trane air conditioning products. Each product has a unique number and type of inputs so this user interface would have to allow for the dynamic adding and removing of input controls for each product line. In addition to a dynamic control pane for input controls, there would need to be an output pane that would display sound power values in either full or one-third octave band sound spectra as well as a grid of performance data for the unit fans. An example of the graphical user interface that would be required for CSAA model calculations is shown in the Figure 4 below.



CSAA Performance Climate Changer

Unit Path Configuration	Supply Fan Only Unit
CDQ Wheel	☐
Supply Fan in Unit	✓
Supply Fan Unit Size	3
Supply Fan	
Fan Configuration	HDT
Fan Discharge Section	Horizontal Section
Fan Section Lining	No Lining
CFM	15,000
Diffuser	☐
TSP	2.50
Density Factor	1.00

Discharge Air Component	Ducted Discharge
Air Opening Direction	Side/Bottom/Top
Air Opening Size	75 in.
Outdoor Air Component	Ducted Outdoor
Air Opening Direction	Top/Bottom/Side
Air Opening Size	100 in.
Return Air Component	Ducted Inlet
Air Opening Direction	Top/Bottom/Side
Air Opening Size	75 in.

Fan to Discharge Air Region

Total Lining Length	120.00 in.
	56" No Lining / 64" 2" Perf.
Selected Lining Length	64.00 in.
Selected Lining Type	2" Perf.
Appurt. List	Coil 4 or 6 row / Coil 8 or 10+ row
Appurtenance	Coil
Type	8 or 10+ row

Fan to Return Air Region

Total Lining Length	150.00 in.
	100" 2" Perf. / 50" Solid
Selected Lining Length	50.00 in.
Selected Lining Type	Solid
Appurt. List	Filter Bag / Coil 8 or 10+ row
Appurtenance	Coil
Type	8 or 10+ row
✓ Discharge Silencer	3" Rect. Film
☐ Return Silencer	No Silencer
Multi-Zone	No Multizone

	63	125	250	500	1000	2000	4000	8000
Ducted Discharge Component	61	61	51	49	48	46	45	35
Ducted Inlet Component	49	51	52	51	50	57	58	51
Ducted Outdoor Air Component	63	65	66	65	64	71	72	65
CS Component	103	104	98	92	88	90	87	78
Frequency Band (Hz)								

Operating Point	**Supply Fan**
CFM	15000
TSP	2.5
RPM	901
BHP	9.4
XWO	70.8
Efficiency	62.8

Ready Current Job: JOB_9:55 AM_04/13/2010

Figure 4. New Product Graphical User Interface Example

3 Design

The design process for this project was handled according to the iterative lifecycle model selected for this project. The initial high level design was completed immediately following the initial requirements phase. This high level design process included defining the high level system architecture and re-engineering the existing acoustics software design using UML class diagrams as a tool. The initial high level architecture process revealed three individual architecture layers that are essential to the acoustics software system. Each of these layers required a significant amount of additional detailed design work that was completed in iterative phases. As each iterative phase was tackled the detailed design for the next piece of the acoustics library was revised. Throughout the duration of the project the design of each section was updated several times in order to keep the entire design consistent with newly discovered requirements and design issues.

The foundation for the design of the acoustics library was derived from the design of the previous acoustics software systems. However, since many of the core problems with the legacy acoustics software had stemmed directly from poor design decisions, a lot of care was given to extracting the key logic from the previous designs while re-engineering the design in a way that would improve the overall quality of the system design. The logic for the generic core of the acoustics library was already well defined according to an acoustics calculation algorithm described which is described in internal Trane documentation. However, the design of the system which translates a unit configuration definition into an abstract and generic model that can be acoustically evaluated did not exist in previous acoustics software and thus had to be designed from the ground up for this project.

The design of the acoustics calculation GUI program was also largely a design from scratch effort rather than a re-engineering effort. While previous acoustics calculation programs were used as a starting point, many requested features had never been implemented in a Trane software program and had to therefore be designed from the ground up. No previous acoustics GUI program was dynamically flexible enough to accommodate the vast array of configuration options which may apply to a CSAA unit.

3.1 High Level System Architecture

The ideal high level architecture for acoustics calculation system consists of three main layers. A conceptual view of this architecture is shown in Figure 5 below:

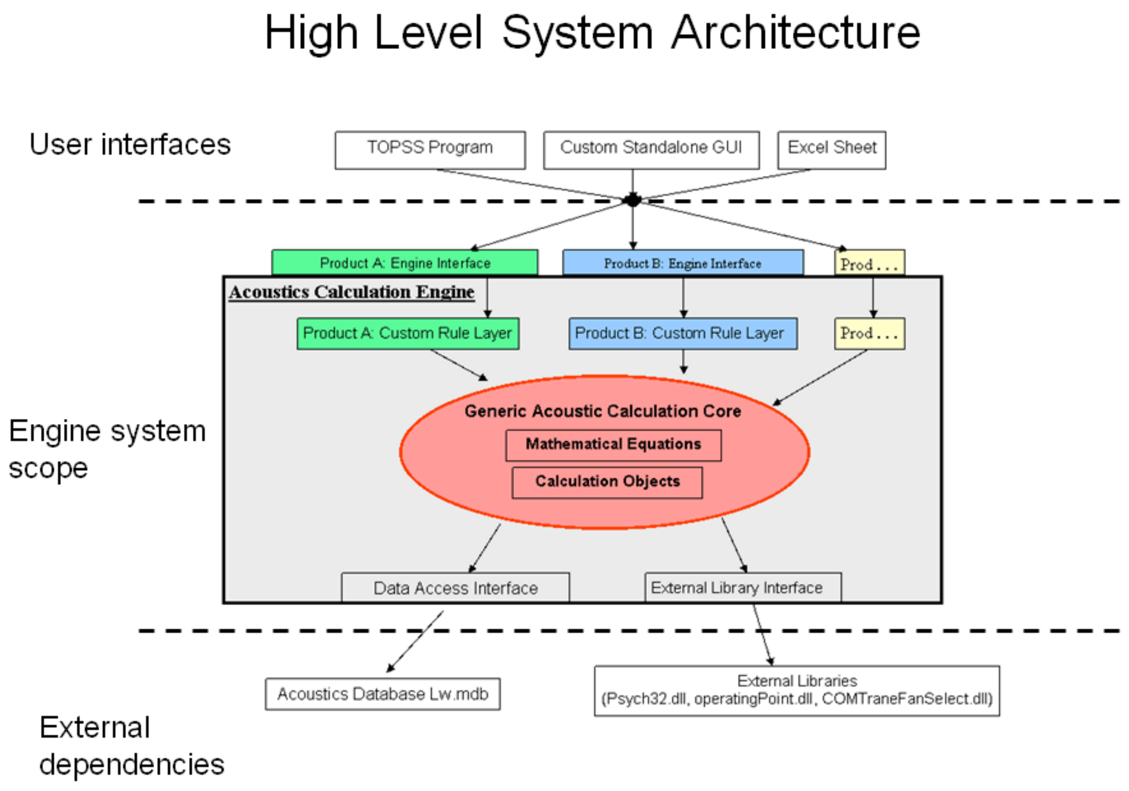

High Level System Architecture
User interfaces
TOPSS Program
Custom Standalone GUI
Excel Sheet
Product A: Engine Interface
Product B: Engine Interface
Prod . . .
Acoustics Calculation Engine
Product A: Custom Rule Layer
Product B: Custom Rule Layer
Prod . . .
Engine system scope
Generic Acoustic Calculation Core
Mathematical Equations
Calculation Objects
Data Access Interface
External Library Interface
Acoustics Database Lw.mdb
External Libraries (Psych32.dll, operatingPoint.dll, COMTraneFanSelect.dll)
External dependencies

Figure 5. High Level System Architecture View

The top layer of the architecture consists of the user interfaces which allow user input and display of calculated output values. The acoustics calculation library provides functionality to three separate user interfaces. The first user interface is a widely used Trane product selection tool known as TOPSS. TOPSS allows users to configure a unit and calculate various technical details on the performance of that particular unit. One piece of the data reported by TOPSS is acoustic sound power, which is calculated through the acoustics library. The second user interface is the custom designed acoustics GUI program which is designed specifically as a front end to the acoustics calculation library. This GUI is designed for use by acoustics engineers and product engineers who have some knowledge of acoustics. The third user interface which makes use of the acoustics calculation library is a collection of Microsoft® Excel® macro programs which are embedded in various custom designed engineering tool workbooks. These tools are built by Trane engineers and are used for various work tasks.

The most significant layer within the system architecture is the middle layer of the system, which is the acoustics calculation library or "software engine." This "engine" contains all of the core functionality needed for performing acoustic prediction calculations. This library provides an API to the user interface layers which allow them make requests for acoustic prediction calculations. Each of the various Trane products that are modeled by the acoustics library require a unique programming interface that is tailored for its unique set of inputs. The product specific inputs that are passed through these interfaces are processed by parts of the acoustics library, known as "custom product rule" layers. These product rule layers are designed to handle the product specific details of acoustics calculations. These layers rely on a generic core “engine” in the acoustics library which handles the non-product related parts of the calculations, relating to the physics and mechanics of the acoustics model calculations.

The bottom layer of the acoustics calculation library consists of an acoustics data store and a few dependencies on external Trane standard libraries which handle tasks like unit conversion, physics calculations and mathematical calculations. There is a data access layer built into the acoustics library to handle database requests. There are also several third party library software wrappers written to allow easy and consistent interfacing from the acoustics library to these external libraries.

3.2 Design of the Core Acoustics Library

The design of the core acoustics library consists of a public facing interface as well as several internal core classes which are meant to be either implemented for specific product lines or used as a means of handling the generic parts of the acoustics prediction algorithm. The foundational structure of this architecture is shown below in the Figure 6.

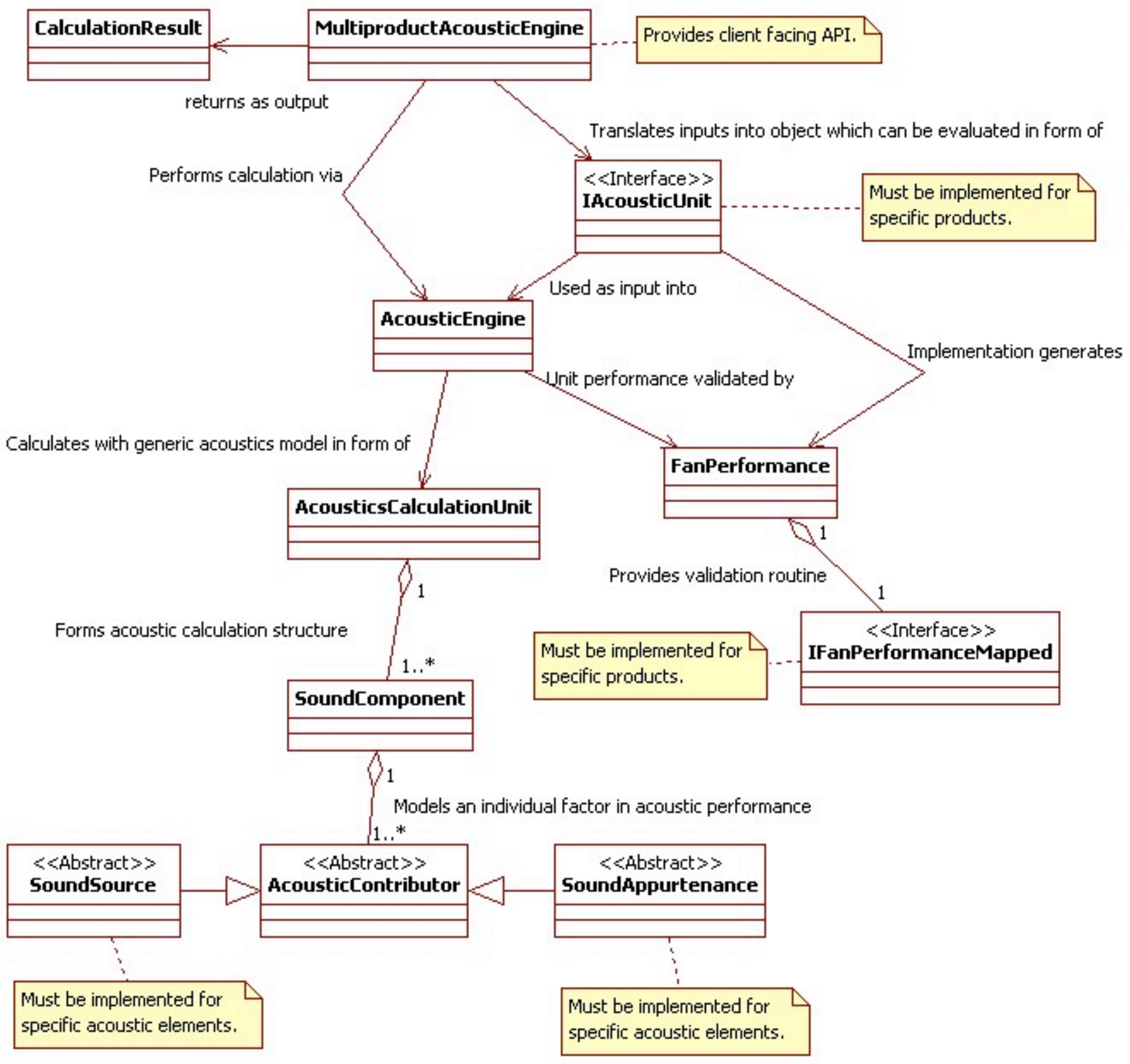
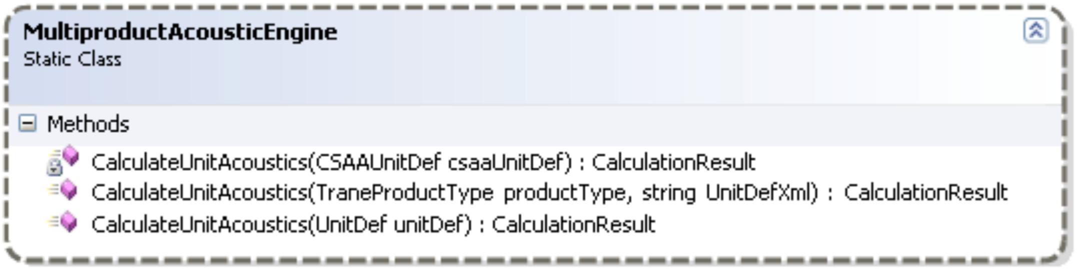

CalculationResult
MultiproductAcousticEngine
Provides client facing API.
returns as output
Translates inputs into object which can be evaluated in form of
Performs calculation via
<<Interface>>
IAcousticUnit
Must be implemented for specific products.
Used as input into
AcousticEngine
Implementation generates
Unit performance validated by
Calculates with generic acoustics model in form of
FanPerformance
AcousticsCalculationUnit
1
Provides validation routine
1
1
Forms acoustic calculation structure
<<Interface>>
IFanPerformanceMapped
Must be implemented for specific products.
1..*
SoundComponent
1
Models an individual factor in acoustic performance
1..*
<<Abstract>>
SoundSource
<<Abstract>>
AcousticContributor
<<Abstract>>
SoundAppurtenance
Must be implemented for specific acoustic elements.
Must be implemented for specific acoustic elements.

Figure 6. Core Acoustics Library Architecture Class Diagram

A list of the detailed class views for each class in this diagram can be found in Appendix A.

3.3 Design of the Acoustics API

The API for the acoustics library was designed to allow client software to input the definition of a Trane air conditioning unit and get back a data structure that describes the acoustic performance of particular air conditioning unit. Each product line has unique set of variables that effect acoustic performance. Within each product line there are a large number of these variables, many of which only apply to specific configuration scenarios. Although the number and type of input variables varies widely, the acoustics API for performing acoustics calculations was designed to consist of a single generic programming call that ensures consistency for client software. The MultiproductAcousticsEngine class was designed to serve as a public facing static programming interface that is used for making acoustics prediction calculations. A detailed class diagram of the MultiproductAcousticsEngine is shown in Figure 7 below.



```
MultiproductAcousticEngine
Static Class

Methods
CalculateUnitAcoustics(CSAAUnitDef csaaUnitDef) : CalculationResult
CalculateUnitAcoustics(TraneProductType productType, string UnitDefXml) : CalculationResult
CalculateUnitAcoustics(UnitDef unitDef) : CalculationResult
```

Figure 7. Multiproduct Acoustics Engine Detailed Class Diagram

The acoustics library provides clients with a set of public classes that are designed to define the configuration of Trane units so that they can be evaluated for acoustic prediction. These unit definition classes must be implemented for each Trane product line in the acoustic library in order to allow client code to access all of the options for that product line. A set of these unit definition classes was defined for the CSAA product line, which was the first product line to be implemented in the Trane acoustics library. A class architecture diagram for the CSAA unit definition classes is shown in Figure 8 below.

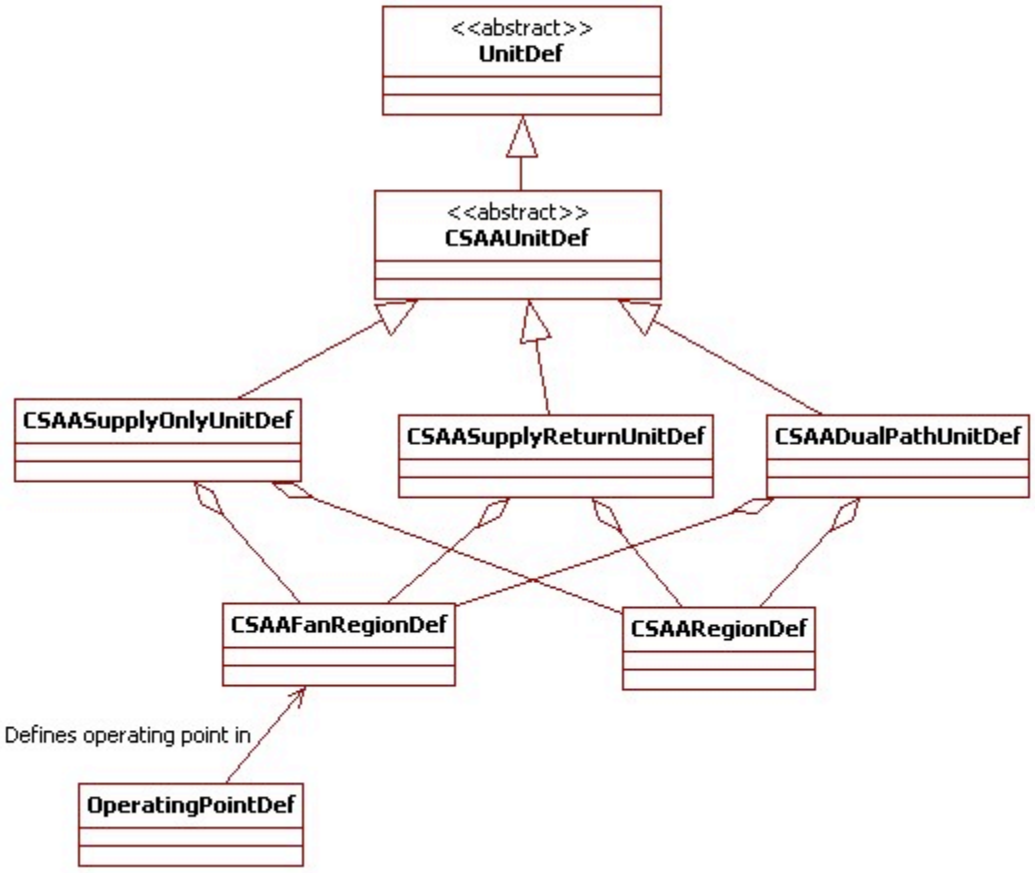
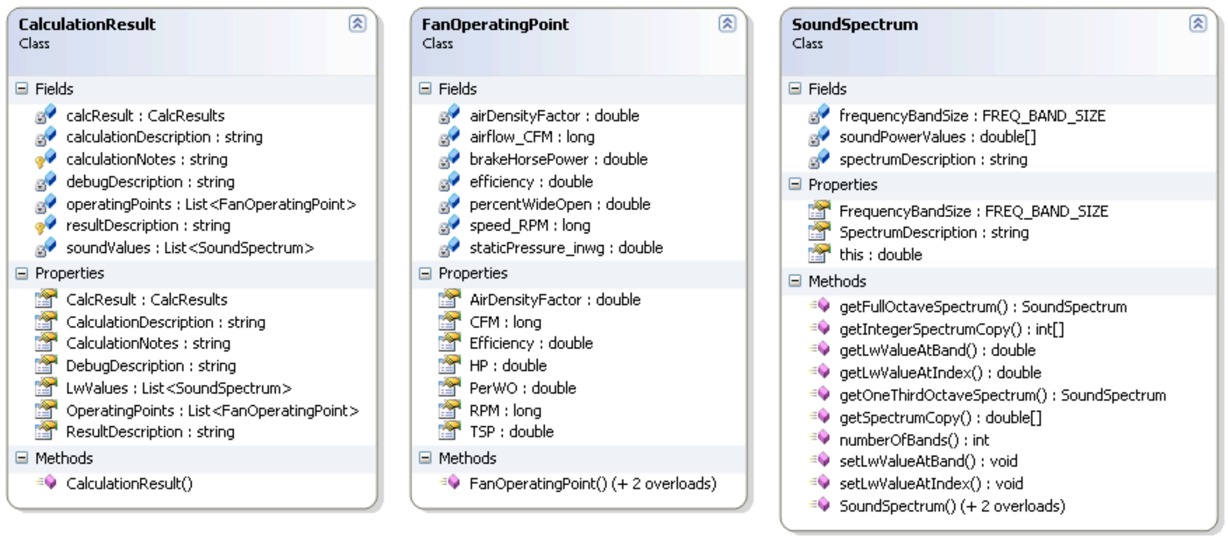


Figure 8. Architecture for CSAA Unit Definition Classes

The UnitDef class is the public abstract parent class of all unit definition classes which are accessible to client code. Any future product implementation will require the developer to write a subclass of the UnitDef class. The CSAAUnitDef class is the abstract child class that defines all Trane units in the CSAA product line. This class was made concrete through being sub classed by three other CSAA specific classes that define the three unique configurations that CSAA units are offered in. Detailed class diagrams for each of these classes can be found in Appendix B.

The requirements of the acoustics library specify that the output of an acoustics prediction must include a collection of sound power value spectra, a collection of operating point values and some textual information that is associated with the sound power data. This associated information includes descriptions of any industry standards that are used in calculating acoustic performance, the type of unit components that are being evaluated and a description of the frequency bands used for the acoustic data produced by the acoustics calculation. The

CalculationResult class was created to serve as this type of general purpose return structure. A detailed class diagram for CalculationResult can be seen in Figure 9 below.



CalculationResult
Class

Fields
- calcResult : CalcResults
- calculationDescription : string
- calculationNotes : string
- debugDescription : string
- operatingPoints : List<FanOperatingPoint>
- resultDescription : string
- soundValues : List<SoundSpectrum>

Properties
- CalcResult : CalcResults
- CalculationDescription : string
- CalculationNotes : string
- DebugDescription : string
- LwValues : List<SoundSpectrum>
- OperatingPoints : List<FanOperatingPoint>
- ResultDescription : string

Methods
- CalculationResult()

FanOperatingPoint
Class

Fields
- airDensityFactor : double
- airflow_CFM : long
- brakeHorsePower : double
- efficiency : double
- percentWideOpen : double
- speed_RPM : long
- staticPressure_inwg : double

Properties
- AirDensityFactor : double
- CFM : long
- Efficiency : double
- HP : double
- PerWO : double
- RPM : long
- TSP : double

Methods
- FanOperatingPoint() (+ 2 overloads)

SoundSpectrum
Class

Fields
- frequencyBandSize : FREQ_BAND_SIZE
- soundPowerValues : double[]
- spectrumDescription : string

Properties
- FrequencyBandSize : FREQ_BAND_SIZE
- SpectrumDescription : string
- this : double

Methods
- getFullOctaveSpectrum() : SoundSpectrum
- getIntegerSpectrumCopy() : int[]
- getLwValueAtBand() : double
- getLwValueAtIndex() : double
- getOneThirdOctaveSpectrum() : SoundSpectrum
- getSpectrumCopy() : double[]
- numberOfBands() : int
- setLwValueAtBand() : void
- setLwValueAtIndex() : void
- SoundSpectrum() (+ 2 overloads)

Figure 9. CalculationResult Detailed Class Diagram

The CalculationResult objects that are output from the acoustics library are used by client software in order to access all the data that is produced while making an acoustic prediction. They are also used by client code in order to store useful information in debug and error files that are used for maintenance of the acoustics library.

3.4 Design of the CSAA Section of the Acoustics Library

In order to actually calculate acoustic predictions for a Trane unit, a set of classes must be implemented within the acoustics library that can model the various options available for that product and calculate the impact that those options have on the overall unit acoustics. The design of the acoustics library allows these internal product specific classes to be implemented by simply sub classing one of the existing generic AcousticContributor classes which were designed for efficient, generic handling of acoustic calculations. This is exactly what was done for the CSAA implementation of the acoustics library.

The most complex component of any air conditioning unit acoustic calculation is the fan. The fan is the primary source of sound within the unit and the sound levels that it produces varies greatly depending on a large number of factors. In order to accurately model a fan operating in a unit, a class called FanRegression was created as a subclass of the abstract SoundSource class. Because many Trane products have fans with similar characteristics, the FanRegression class was designed to be product neutral. It however, contains components that are product specific and must be implemented for each product that is implemented in the acoustics library. The class architecture of the FanRegression class and its components is shown below in Figure 10 below.

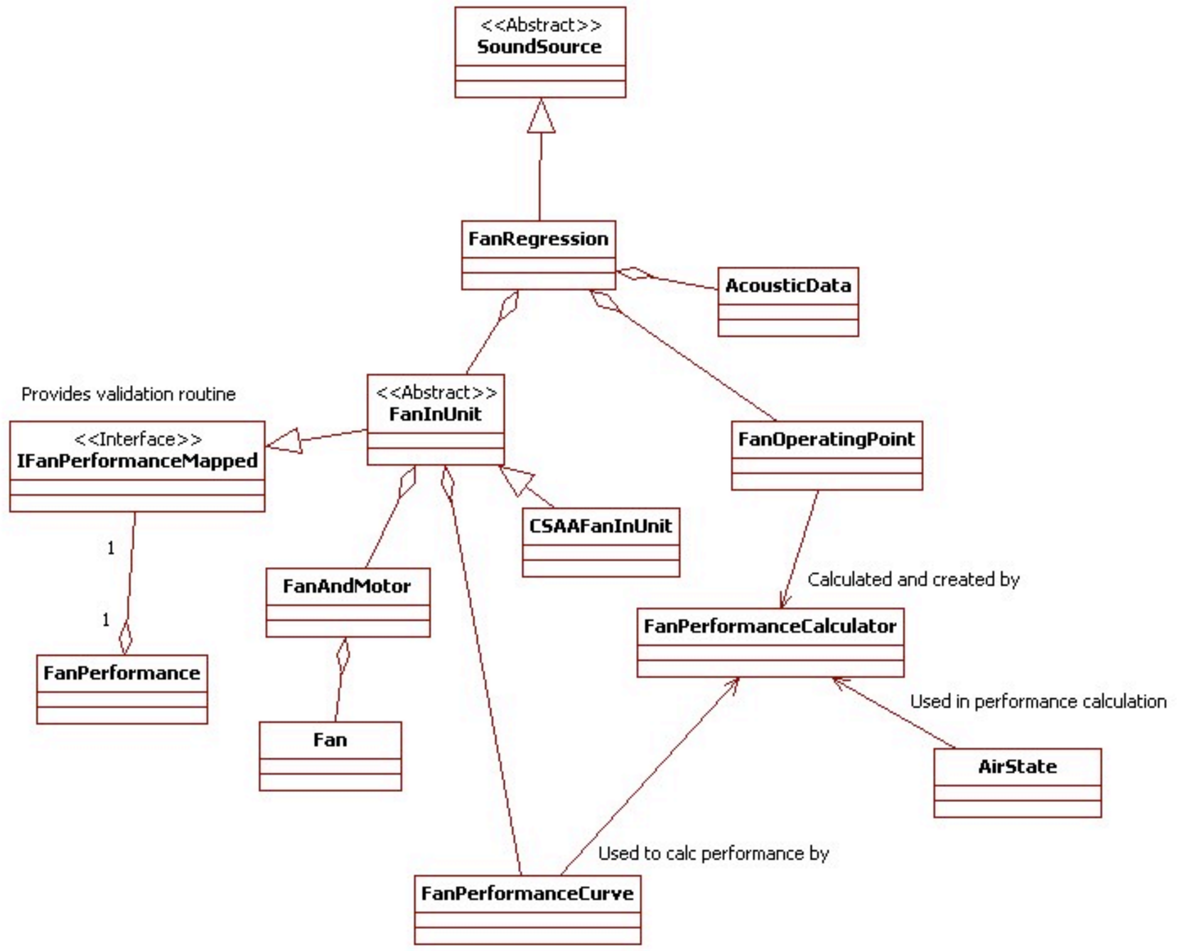


Figure 10. FanRegression Class Architecture

Another interesting part of the acoustic library design is the modeling of acoustic silencers. Acoustic silencers are devices that may be installed inside an air conditioning unit in order to help control and shape the acoustic performance of a unit. Silencers have the unique property of both reducing and creating sound at the same time. An installed silencer will reduce the sound that enters it. This effect is known as "insertion loss." At the same time the silencer disturbs the air in a way that creates additional noise known as "regenerated noise." In order to model this dual effect, it was necessary to create two separate classes to represent silencer effects. A subclass of SoundSource was created and called SilencerRegen. A subclass of SoundAppurtenance was created and called SilenecerInsertionLoss. Because the calculation of these two effects is complex and inter-related, a third general Silencer class was created to model the actual physical properties of a silencer. The Silencer class is used as a component of both the SilencerInsertionLoss and the SilnecerRegen classes. The class structure of this design can be seen in Figure 11 below.

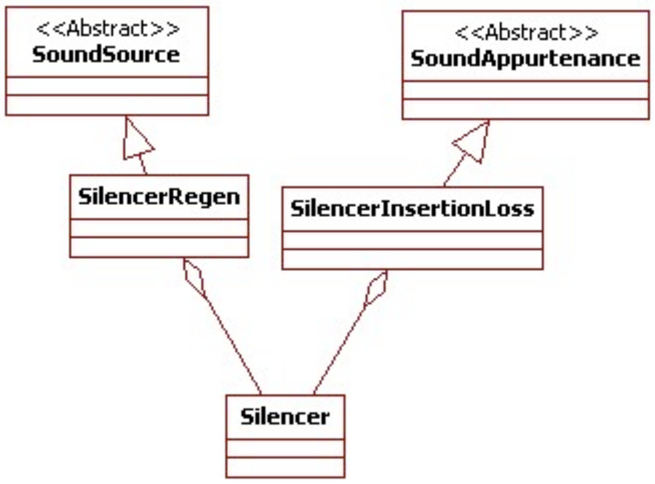


Figure 11. Acoustic Silencer Class Diagram

The other type of component that has an effect on acoustic performance is called a sound appurtenance. The CSAA unit options include a large number of sound appurtenance effects. These effects are modeled as subclasses of the abstract SoundAppurtenance class. Each of these CSAA specific appurtenance classes encapsulates the logic for calculating the acoustic effect that it has on the unit. All of the CSAA related SoundAppurtenance classes are listed in Table 1 below.

DischargeOpeningSizeEffect	The effect produced by the size of the unit's discharge air opening.	Any Trane unit.
Return OpeningSizeEffect	The effect produced by the size of the unit's return air opening.	Any Trane unit.
DischargeDirectionEffect	The effect produced by the directional orientation of a unit's discharge air	Any Trane unit.
OutdoorOpeningSizeEffect	The effect produced by the size of the unit's outdoor air opening.	Any Trane unit that includes an outdoor air opening.
ExhaustOpeningSizeEffect	The effect produced by the size of the unit's exhaust air opening.	Any Trane unit that includes an exhaust air opening.
	orientation of a unit's return air opening.	
CSAAFanDischargeConfigEffect	The effect produced by the configuration	Any Trane CSAA unit.
OutdoorAirDirectionEffect	The effect produced by the directional orientation of a unit's outdoor air opening.	Any Trane unit that includes an outdoor air opening.
ExhaustAirDirectionEffect	The effect produced by the directional orientation of a unit's exhaust air	Any Trane unit that includes an exhaust air opening

CSAAFanDischargeConfigEffect	The effect produced by the configuration of the fan discharge.	Any Trane CSAA unit.
CoilEffect	The effect produced by a heating or cooling coil in the airflow path.	Any Trane unit.
FilterEffect	The effect produced by an air filter in the airflow path.	Any Trane CSAA unit.
CDQWheelEffect	The effect produced by a CDQ wheel in the unit's airflow path.	Any Trane CSAA unit.
CSAALining	The effect produced by lining a CSAA unit with a specific type of panel lining.	Any Trane CSAA unit.
EnergyWheelEffect	The effect that exists in unit configurations which include energy wheels.	Any Trane CSAA unit.
CSAADiffuserEffect	The effect produced when a diffuser is installed on the fan.	Any Trane unit.
MultiZoneEffect	The effect that is produced when a unit has multiple air discharge paths.	Any Trane unit.
InletCasingDischargeConfigEffect	The effect that is produced in the inlet + casing acoustic component by the	Any Trane unit.

Table 1. SoundAppurtance Classes

The class relationship of the appurtenance classes can be seen in Figure 12 below.

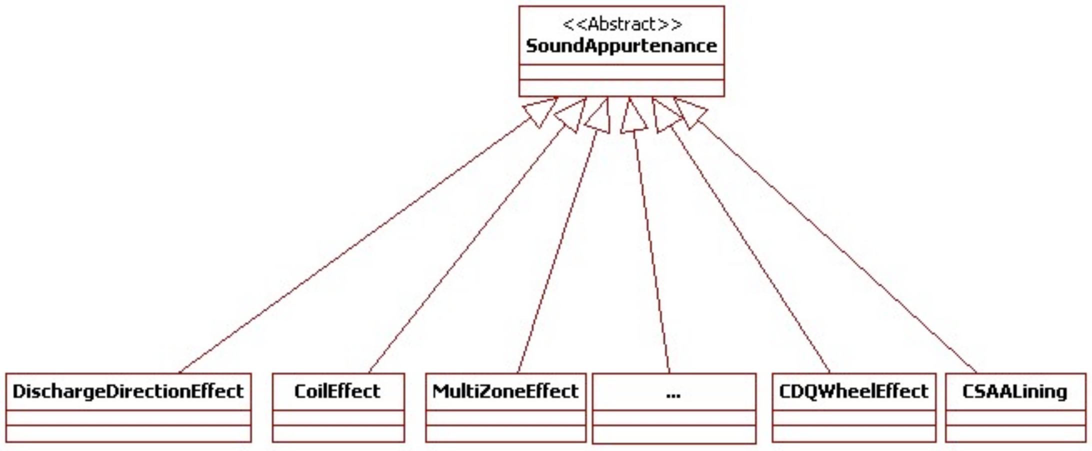
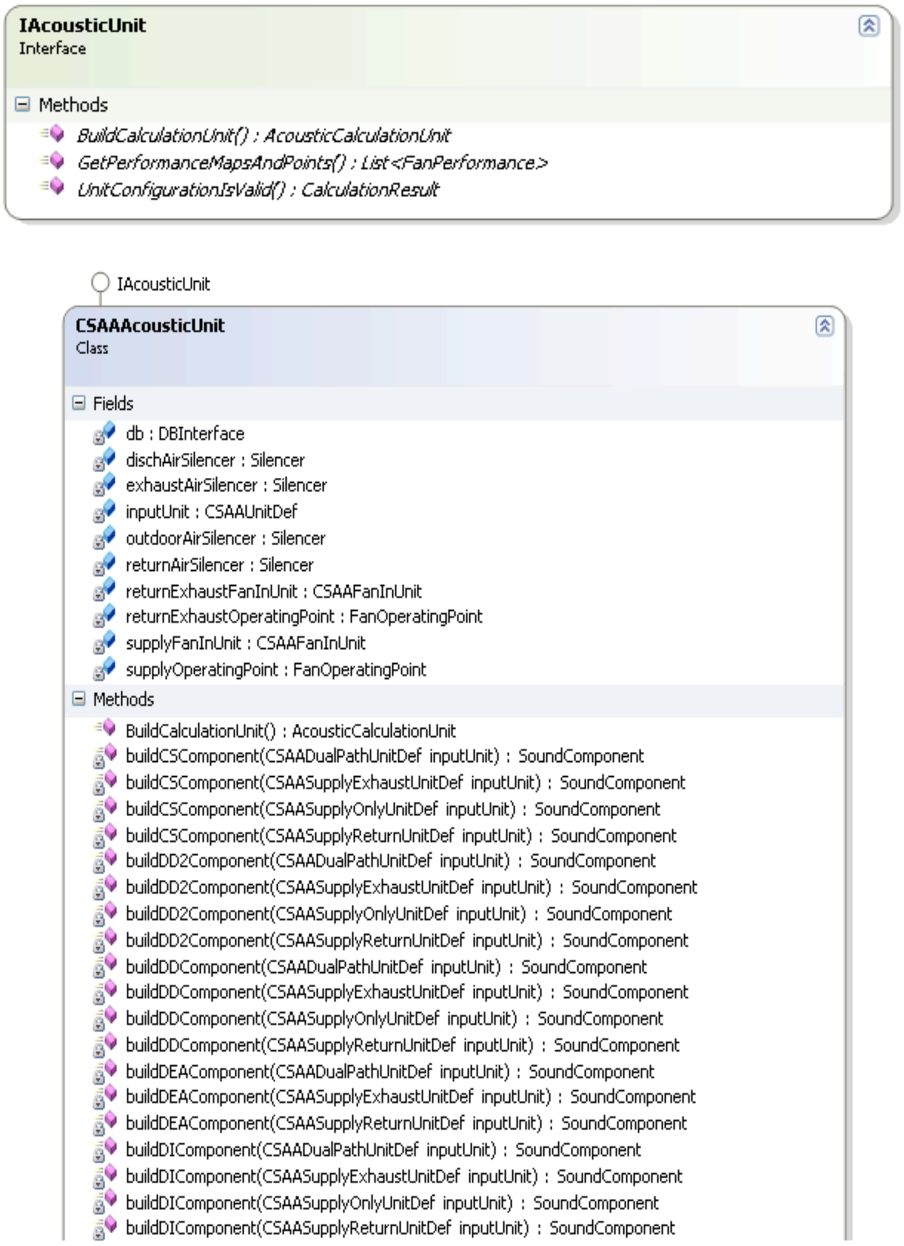


Figure 12. Acoustic Sound Appurtenance Classes

For products that are complex like the CSAA units, there are many business logic rules that are required in order to decide which of all the AcousticContributors apply for a given configuration. This logic should be stored in a CSAA-specific implementation class. In order to allow for the product specific business logic that is needed for acoustic predictions, the acoustics library provides the IAcousticUnit interface which is meant to be implemented by product specific classes. The IAcousticUnit interface exposes three functions that are needed for transforming the definition of an acoustic unit into a generic AcousticCalculationUnit object. The AcousticCalculationUnit object is the generic abstracted representation of an air conditioning unit that can be acoustically evaluated. For CSAA acoustic predictions the CSAAAcousticUnit class was implemented to fulfill this need. This is a heavy weight class which contains all of the business logic for transforming the complex CSAA unit configurations into the AcousticsCalculationUnit objects which can be acoustically evaluated. A detailed view of the IAcousticUnit interface and its CSAA specific implementation can be seen in Figure 13 below.

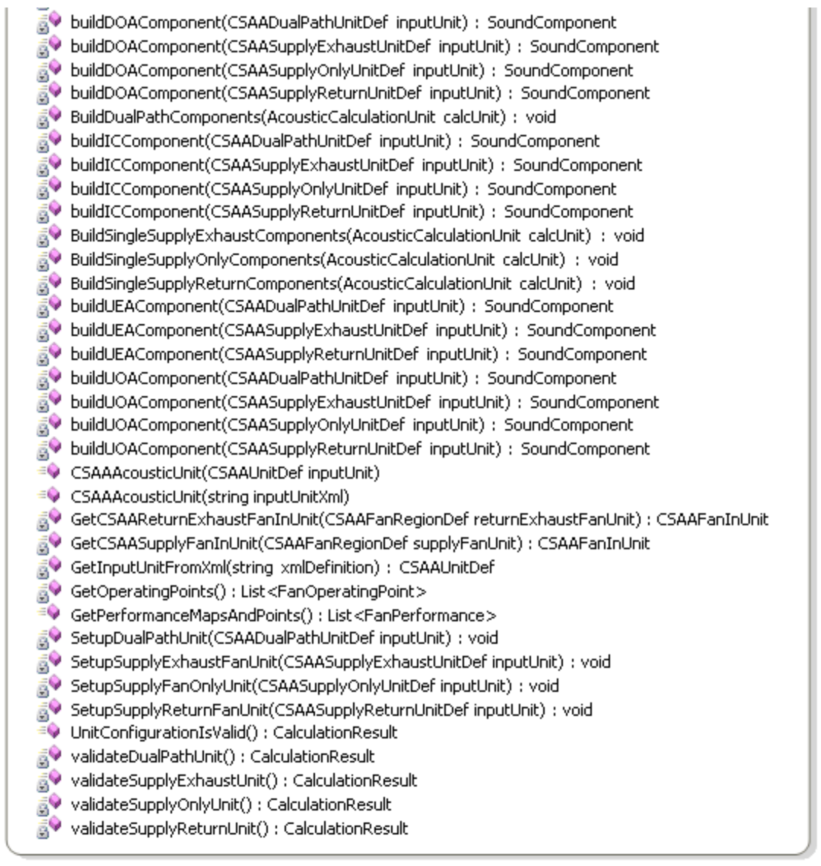


```
IAcousticUnit
Interface

Methods
  BuildCalculationUnit() : AcousticCalculationUnit
  GetPerformanceMapsAndPoints() : List<FanPerformance>
  UnitConfigurationIsValid() : CalculationResult
```

```
IAcousticUnit
CSAAAcousticUnit
Class

Fields
  db : DBInterface
  dischAirSilencer : Silencer
  exhaustAirSilencer : Silencer
  inputUnit : CSAAUnitDef
  outdoorAirSilencer : Silencer
  returnAirSilencer : Silencer
  returnExhaustFanInUnit : CSAAFanInUnit
  returnExhaustOperatingPoint : FanOperatingPoint
  supplyFanInUnit : CSAAFanInUnit
  supplyOperatingPoint : FanOperatingPoint
Methods
  BuildCalculationUnit() : AcousticCalculationUnit
  buildCSComponent(CSAADualPathUnitDef inputUnit) : SoundComponent
  buildCSComponent(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
  buildCSComponent(CSAASupplyOnlyUnitDef inputUnit) : SoundComponent
  buildCSComponent(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
  buildDD2Component(CSAADualPathUnitDef inputUnit) : SoundComponent
  buildDD2Component(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
  buildDD2Component(CSAASupplyOnlyUnitDef inputUnit) : SoundComponent
  buildDD2Component(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
  buildDDComponent(CSAADualPathUnitDef inputUnit) : SoundComponent
  buildDDComponent(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
  buildDDComponent(CSAASupplyOnlyUnitDef inputUnit) : SoundComponent
  buildDDComponent(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
  buildDEAComponent(CSAADualPathUnitDef inputUnit) : SoundComponent
  buildDEAComponent(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
  buildDEAComponent(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
  buildDIComponent(CSAADualPathUnitDef inputUnit) : SoundComponent
  buildDIComponent(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
  buildDIComponent(CSAASupplyOnlyUnitDef inputUnit) : SoundComponent
  buildDIComponent(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
```

(Figure 13 continued on next page)

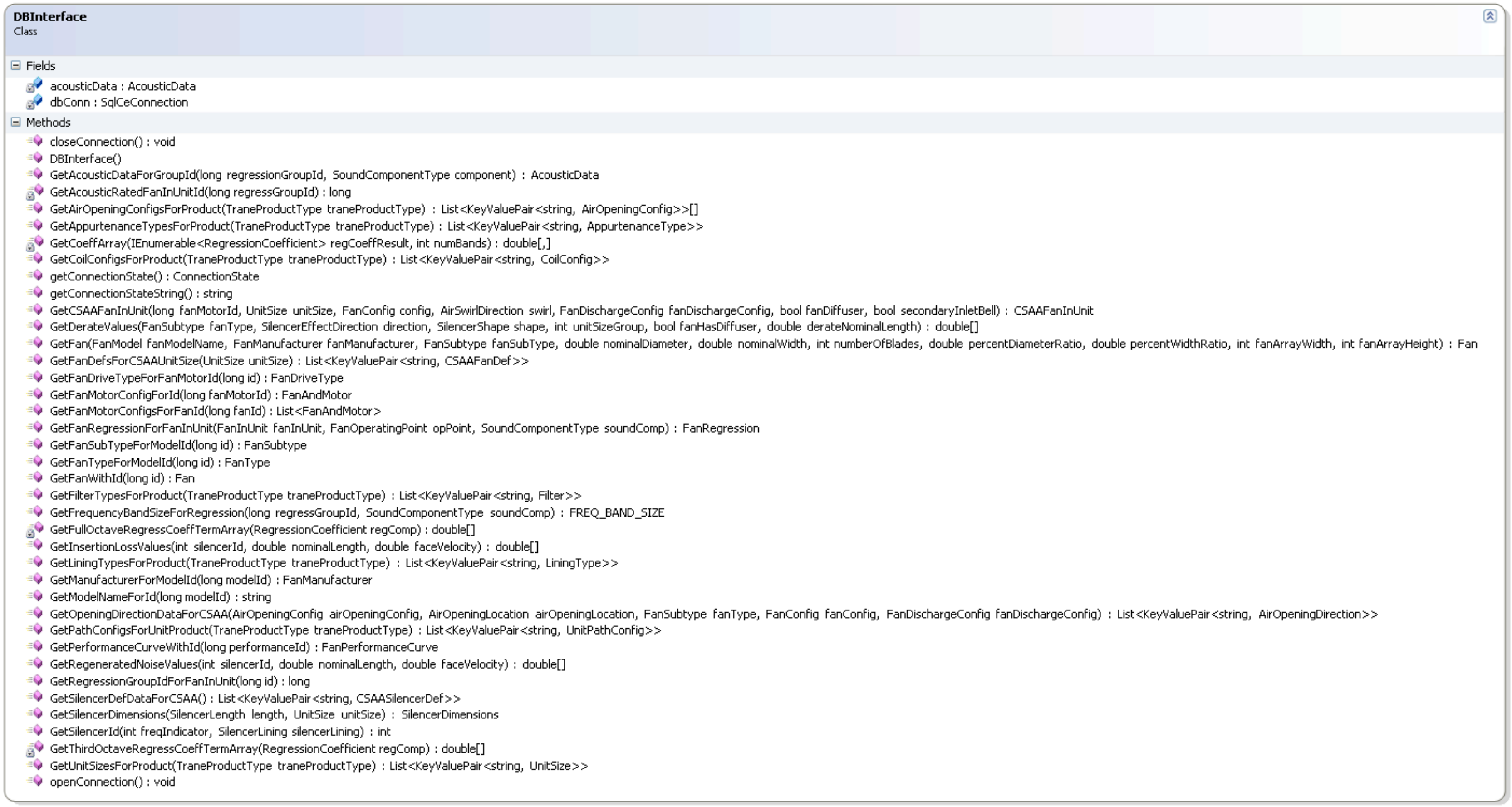
Figure 13. IAcousticUnit Interface Class Details



```
buildDOAComponent(CSAADualPathUnitDef inputUnit) : SoundComponent
buildDOAComponent(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
buildDOAComponent(CSAASupplyOnlyUnitDef inputUnit) : SoundComponent
buildDOAComponent(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
BuildDualPathComponents(AcousticCalculationUnit calcUnit) : void
buildICComponent(CSAADualPathUnitDef inputUnit) : SoundComponent
buildICComponent(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
buildICComponent(CSAASupplyOnlyUnitDef inputUnit) : SoundComponent
buildICComponent(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
BuildSingleSupplyExhaustComponents(AcousticCalculationUnit calcUnit) : void
BuildSingleSupplyOnlyComponents(AcousticCalculationUnit calcUnit) : void
BuildSingleSupplyReturnComponents(AcousticCalculationUnit calcUnit) : void
buildUEAComponent(CSAADualPathUnitDef inputUnit) : SoundComponent
buildUEAComponent(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
buildUEAComponent(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
buildUOAComponent(CSAADualPathUnitDef inputUnit) : SoundComponent
buildUOAComponent(CSAASupplyExhaustUnitDef inputUnit) : SoundComponent
buildUOAComponent(CSAASupplyOnlyUnitDef inputUnit) : SoundComponent
buildUOAComponent(CSAASupplyReturnUnitDef inputUnit) : SoundComponent
CSAAAcousticUnit(CSAAUnitDef inputUnit)
CSAAAcousticUnit(string inputUnitXml)
GetCSAAReturnExhaustFanInUnit(CSAAFanRegionDef returnExhaustFanUnit) : CSAAFanInUnit
GetCSAASupplyFanInUnit(CSAAFanRegionDef supplyFanUnit) : CSAAFanInUnit
GetInputUnitFromXml(string xmlDefinition) : CSAAUnitDef
GetOperatingPoints() : List<FanOperatingPoint>
GetPerformanceMapsAndPoints() : List<FanPerformance>
SetupDualPathUnit(CSAADualPathUnitDef inputUnit) : void
SetupSupplyExhaustFanUnit(CSAASupplyExhaustUnitDef inputUnit) : void
SetupSupplyFanOnlyUnit(CSAASupplyOnlyUnitDef inputUnit) : void
SetupSupplyReturnFanUnit(CSAASupplyReturnUnitDef inputUnit) : void
UnitConfigurationIsValid() : CalculationResult
validateDualPathUnit() : CalculationResult
validateSupplyExhaustUnit() : CalculationResult
validateSupplyOnlyUnit() : CalculationResult
validateSupplyReturnUnit() : CalculationResult
```

Figure 13. IAcousticUnit Interface Class Details

Most of the product specific classes including the CSAAAcousticUnit class need access to a database which contains actual test data for the units. This data is used to make the acoustics prediction calculations. Access to this data is provided using the data access object design pattern. The DBInterface class serves as the interface layer between the acoustics library and the acoustics data store. A detailed view of the DBInterface class can be seen in Figure 14 below.



```
DBInterface
Class

⊟ Fields
    acousticData : AcousticData
    dbConn : SqlCeConnection
⊟ Methods
    closeConnection() : void
    DBInterface()
    GetAcousticDataForGroupId(long regressionGroupId, SoundComponentType component) : AcousticData
    GetAcousticRatedFanInUnitId(long regressGroupId) : long
    GetAirOpeningConfigsForProduct(TraneProductType traneProductType) : List<KeyValuePair<string, AirOpeningConfig>>[]
    GetAppurtenanceTypesForProduct(TraneProductType traneProductType) : List<KeyValuePair<string, AppurtenanceType>>
    GetCoeffArray(IEnumerable<RegressionCoefficient> regCoeffResult, int numBands) : double[,]
    GetCoilConfigsForProduct(TraneProductType traneProductType) : List<KeyValuePair<string, CoilConfig>>
    getConnectionState() : ConnectionState
    getConnectionStateString() : string
    GetCSAAFanInUnit(long fanMotorId, UnitSize unitSize, FanConfig config, AirSwirlDirection swirl, FanDischargeConfig fanDischargeConfig, bool fanDiffuser, bool secondaryInletBell) : CSAAFanInUnit
    GetDerateValues(FanSubtype fanType, SilencerEffectDirection direction, SilencerShape shape, int unitSizeGroup, bool fanHasDiffuser, double derateNominalLength) : double[]
    GetFan(FanModel fanModelName, FanManufacturer fanManufacturer, FanSubtype fanSubType, double nominalDiameter, double nominalWidth, int numberOfBlades, double percentDiameterRatio, double percentWidthRatio, int fanArrayWidth, int fanArrayHeight) : Fan
    GetFanDefsForCSAAUnitSize(UnitSize unitSize) : List<KeyValuePair<string, CSAAFanDef>>
    GetFanDriveTypeForFanMotorId(long id) : FanDriveType
    GetFanMotorConfigForId(long fanMotorId) : FanAndMotor
    GetFanMotorConfigsForFanId(long fanId) : List<FanAndMotor>
    GetFanRegressionForFanInUnit(FanInUnit fanInUnit, FanOperatingPoint opPoint, SoundComponentType soundComp) : FanRegression
    GetFanSubTypeForModelId(long id) : FanSubtype
    GetFanTypeForModelId(long id) : FanType
    GetFanWithId(long id) : Fan
    GetFilterTypesForProduct(TraneProductType traneProductType) : List<KeyValuePair<string, Filter>>
    GetFrequencyBandSizeForRegression(long regressGroupId, SoundComponentType soundComp) : FREQ_BAND_SIZE
    GetFullOctaveRegressCoeffTermArray(RegressionCoefficient regComp) : double[]
    GetInsertionLossValues(int silencerId, double nominalLength, double faceVelocity) : double[]
    GetLiningTypesForProduct(TraneProductType traneProductType) : List<KeyValuePair<string, LiningType>>
    GetManufacturerForModelId(long modelId) : FanManufacturer
    GetModelNameForId(long modelId) : string
    GetOpeningDirectionDataForCSAA(AirOpeningConfig airOpeningConfig, AirOpeningLocation airOpeningLocation, FanSubtype fanType, FanConfig fanConfig, FanDischargeConfig fanDischargeConfig) : List<KeyValuePair<string, AirOpeningDirection>>
    GetPathConfigsForUnitProduct(TraneProductType traneProductType) : List<KeyValuePair<string, UnitPathConfig>>
    GetPerformanceCurveWithId(long performanceId) : FanPerformanceCurve
    GetRegeneratedNoiseValues(int silencerId, double nominalLength, double faceVelocity) : double[]
    GetRegressionGroupIdForFanInUnit(long id) : long
    GetSilencerDefDataForCSAA() : List<KeyValuePair<string, CSAASilencerDef>>
    GetSilencerDimensions(SilencerLength length, UnitSize unitSize) : SilencerDimensions
    GetSilencerId(int freqIndicator, SilencerLining silencerLining) : int
    GetThirdOctaveRegressCoeffTermArray(RegressionCoefficient regComp) : double[]
    GetUnitSizesForProduct(TraneProductType traneProductType) : List<KeyValuePair<string, UnitSize>>
    openConnection() : void
```

Figure 14. DBInterface Class Details

3.5 Design of the Acoustics Data Store

The process of calculation acoustic predictions for an air conditioning product depends on using data from actual laboratory tests in order to model the acoustic effect of each component in the unit. Several different types of data must be stored in a database including fan performance data, sound regression data, acoustic appurtenance effect data, unit dimension data and unit configuration data.

The previous Trane acoustic software engines relied on a single acoustics database. All of the data for acoustic predictions was stored in a Microsoft® Access® file which was accessed by the acoustics engine through a data provider. The data in this file was input haphazardly with very little attention to any kind of database design principles. Data was added as needed often by just adding a new table which would serve simply as a lookup table. Database logic was unnecessarily duplicated across many similar tables. Many tables in the old acoustics database have become obsolete and serve to bloat the database file which gets deployed with the old acoustics library.

One goal of this project was to improve the database design for the acoustics library by creating a well designed relational database that can easily be expanded for future acoustics software projects. The ideal database design leverages the logic that is shared between the various products and their data storage needs. The design for the database was updated as the various project phases where completed. An E.R diagram for the database can be found in Appendix C.

The old database was analyzed in order to find places where the database logic was redundant. For instance, there were multiple Fan tables within the previous database. Each of these tables supported one of the various acoustic programs that needed fan data. However, most of the properties of a fan are not related to a specific product line and there are instances in which having a single fan table would be beneficial. The new database was designed by extracting the general properties of all fans and created a single Fan table where all the fans used in Trane products can be found. If additional product related information is a needed for a given fan this data will be stored in a product specific table which might have a foreign key relationship with the primary fan table.

Another way that the database design was improved was by using proper relationships between tables in order to avoid unnecessary data duplication. For instance, rather than duplicating fan information in the Unit table which needs some information about fans, the data would be stored in the fan table and a foreign key relationship would be used in the Unit table to access it when needed. Another example of refactoring that was done is with the regression data which is now stored in a single regression data table rather than several program specific tables. The various product lines may have a product specific UnitConfiguration table which allows some number of unit configuration properties to be used as a composite key for identifying a unique regression data set. These product specific unit configuration tables have a reference to the regression data table through a foreign key relationship. In a similar manner the unit configuration tables have a foreign key relationship with fan performance tables.

The CSAAFanInUnit table is the single product specific unit configuration table for the CSAA implementation of the acoustics library. The rows of this table uniquely identify a CSAA unit in a specific configuration that is offered by Trane. Each of these configurations is associated with a set of regression data and a set of performance data.

3.6 Design of the Multiproduct Acoustics Prediction Program GUI

A basic GUI program was designed as a front end to the acoustics calculation library in order to allow users to graphically configure units and then calculate and view the acoustic performance data. The GUI design was based largely off of previous Trane acoustics GUI programs and was designed with input from several different groups of customers in order to allow for easy user friendly acoustic calculations to be performed.

The goal of the GUI design was to create a GUI program that would work for all future Trane products which need an acoustic prediction program. In order to fulfill this need a generic two panel layout was designed, where the upper panel contains a flexible grid of user controls pertinent to the product being predicted and the lower panel contains output display controls which are used to display the output and give the user feedback on the acoustic prediction.

The GUI design also provides a menu in order to allow users to load and save acoustic predictions. Images of the final GUI design can be seen in Appendix D.

4 Implementation

The acoustics library and acoustics prediction GUI were implemented in iterative steps according to the selected iterative life cycle model. These iterative implementation steps were completed during the time period from August 2009 - April 2010.

The initial phase of implementation included building the core library classes which serve as the generic engine of the library. A large amount of time was spent carefully implementing these core classes since they are the "moving parts" of the acoustics library and any bugs in the core classes could have a major consequence throughout the entire acoustics library.

After building the core classes of the acoustics library, a small test implementation was built for a simple fictional Trane product. This was done as a "proof of concept" in order to test the design of the core library with an actual product implementation. A set of unit tests was designed and documented using this test implementation and was used to prove the correctness of the core library's design.

After finishing the core library and determining that it was correctly designed and implemented, the third phase of implementation for the CSAA product acoustics components was begun. These components were built in several phases beginning with the complex fan regression classes and then moving onto the simpler acoustic appurtenances.

The implementation of the acoustics database and the database interface layer were completed in parallel with the CSAA components. When data was needed to the test the CSAA component implementation it was added to the database and the proper functionality was added to the data object access layer. The new database was implemented as a Microsoft® SQL Server CE® database file which is a lightweight relational database solution. Part of implementing the acoustics database included designing and implementing a Microsoft® Excel® based tool that is used by the acoustics engineers to enter laboratory test data. This tool uses the data entered by acoustics engineers in order to generate a SQL build script for the acoustics library database.

Implementation of the acoustics prediction GUI program was completed as the final phase of the project. The GUI was implemented using the Microsoft® .Net Windows Forms library. Saving this portion of the project for last didn't allow for as much time as was desired for implementing the GUI. While the basic required functionality was fully implemented, many of the requested features ("bells and whistles") could not be implemented due to time restriction.

Version 1.0 of the acoustics library was analyzed using the NDepend code metrics and analysis tool. The core acoustics library contains 7308 lines of code across 73 classes containing 770 methods. The GUI program contains 6504 lines of code across 15 classes containing 770 methods. The initial version of the database contains 25 tables. The database interface class contains 35 database access methods. This project was implemented in a little under a year from July 2009 to May 2010.

5 Testing

Like many software projects the acoustics library is a complex system that requires robust testing on various levels and at various different stages in the development cycle. Rigorous testing throughout the duration of the development process prevents major bugs from being built into foundational parts of the system. Such bugs can have cascading effects that become very expensive to update.

Unit testing was done on each part of the acoustics library as it was implemented. The NUnit testing framework was utilized in order build a unit testing test suite that could be run as each requirement was implemented. Over 100 unit tests were written for testing the core library functionality. Integration testing was also done at end of each major phase in which the newly implemented features would be combined with the existing code. Several test cases were created which were designed to test the functionalities describe in the use case requirements. No formal coverage calculations were employed for the integration testing process as not enough time was available to be dedicated to the meticulous design of an integration test plan. Some initial system testing was completed by the developer during the last couple months of the project. A small set of system tests were set up using the NUnit framework in order test the system-wide functionality of the acoustics library. Additional system testing was done through the acoustics prediction GUI program.

Formal system testing and regression testing will be completed by other members of the Trane global modeling team. A dedicated quality assurance team exists within the group and has been assigned to develop and execute a formal test plan for the acoustics library software. End user testing is routinely used at Trane by rolling out new builds of software systems to selected groups of expert users who have certain real life use cases that they perform on the software. This method will also be used for future testing of the acoustics library.

6 Deployment

Initially the Trane acoustics library will support and be deployed with two applications. The first is the acoustics prediction GUI program that was developed as a part of this project. This program will be tested and deployed through a Trane engineering tools intranet website that contains similar internal libraries and software tools used by Trane engineers. Trane engineers will be able to download a Windows® installation package for the GUI program that will allow them to install and run the program. This deployment will happen a few months after the conclusion of the project development and testing phases.

Currently, the Trane acoustics prediction GUI program is being deployed over a LAN to a small group of users for GUI user testing.

The second application that the Trane acoustics library will support is the TOPSS application. The acoustics library will be deployed with TOPSS on an integral application server. The deployment of the acoustics library in TOPSS will be done in parallel with the deployment of the acoustics prediction GUI program.

After these two deployments are complete in early summer of 2010, the acoustics library will be used around 10,000 times a month by software that serves Trane engineers and sales people who depend on the functionality provided by the acoustics library for their daily work tasks.

7 Limitations

All of the requirements outline in the project proposal and initial requirements document where implemented in the project. However, there were many feature requests made during the lifetime of the software development cycle that were outside of the scope this project but that would greatly improve the Trane acoustics library and prediction GUI program and will be implemented in the future.

During the course of the project, the data deliverable schedule slipped several times due to competing requests for the Trane engineers who are responsible for producing the data used by the acoustics library. Because of these changes not every size/option originally planned for the CSAA prediction library is currently available in the library. These sizes/options will be rolled into future versions of the acoustics library as the data is made available.

Many basic feature requests were made for the acoustics prediction GUI program which did not fit into the scope of this project and have not yet been implemented. Some of these requests include the following:

> Saving prediction data to a user's hard-drive in a format that can later be loaded back into the program for analysis.
>
> Making predictions for multiple fan units while removing the acoustic effect of one of the fans. This is a technique that is sometimes used to model the acoustic effect of certain custom unit configurations which cannot be modeled with the standard configuration options.
>
> Designing a graphical output for the GUI which displays a diagram of the physical layout of the unit being predicted. This provides less knowledgeable users with a sanity check that the unit inputs they have selected actually describe the unit configuration that is desired.

Setting up the system to handle "batch" runs where a file containing a list of acoustics predictions would be loaded and processed, producing a file listing the acoustic results for all of the units in the batch run file.

Building a robust help system that gives descriptions of each of the input options and instructions on how to decide which option to select. This would greatly improve the learning curve for performing acoustic predictions which is an often confusing and error prone task for beginners.

8 Continuing Work

The acoustics prediction library and acoustics prediction GUI program will be an ongoing project on Trane for years to come as it serves an important and immediate business need. The next stage of development will include adding the remaining data for all sizes/options of the CSAA units. This will be done as data becomes available from the acoustic engineers.

The GUI features requests described in the limitations section will also be implemented in future versions of the acoustics prediction GUI program. The ability to save and load predictions will be high priority as well as the addition of a help system.

The acoustics library was designed with both re-use and extensibility in mind. Requests have already been made to add several additional Trane products to the acoustics library and GUI program. The design of the acoustics library should make this a very straight-forward, low risk task.

In addition to expanding the number and types of acoustic predictions that can be done with the acoustics library, future work will include continued development towards the bigger picture goal of developing an acoustics software system that has the capability to automate the addition of new products and data without the need for a programmer to update the software. This project made significant steps towards that goal by creating an acoustics prediction core library that can be applied to any product with a minimal amount of software development work. However, in order to achieve this bigger goal several tools will need to be developed which automate the process of adding new products to the existing system.

9 Conclusion

This manuscript described in detail the development of a software system designed to allow users to calculate the acoustic performance of a Trane air conditioning product. This system was based on existing Trane acoustics software systems but has many significant improvements over the existing systems which were made through a rigorous re-engineering effort which applied quality software engineering principles [4]. This re-engineering effort resulted in a system that is more flexible, easier to extend and cheaper to maintain. The design of new acoustics library will allow for new Trane products to be implemented in a well defined manner by building on top of the existing core acoustics library. This should reduce development time and decrease the number of defects in new versions of the software. In addition the Acoustics Calculation Algorithm was documented in a paper that describes the vision for the generic acoustics library. This paper defines the model that was implemented in the acoustics prediction library.

The end result of this software development effort is a new Trane acoustics prediction library and GUI program which have the capability of accurately calculating acoustic predictions based on acoustic lab data for the Trane CSAA product line. This software system will be deployed and actively used by over 100 Trane engineers and sales people in order to help customers with acoustic issues, give acoustic consultation to clients and make sales for acoustically sensitive projects. The acoustics library software will be run over 10,000 times a month at Trane.

In addition to meeting the immediate needs of providing a means of getting acoustic performance predictions for the CSAA unit, this software library lays the foundational framework for a robust, flexible and generic acoustics software system that will be much easier, faster and cheaper to maintain than the current existing acoustics software.

10 Bibliography

[1] I. Sommerville, *Software Engineering 6th Ed.*, Addison Wesley, 2001

[2] A.M. Davis, *Software Requirements Analysis and Specification*, Prentice Hall, 1990.

[3] Computer Society/Software and Systems Engineering Standards Committee, "Recommended Practice for Software Requirements Specifications", Std. 830-1998, IEEE Standards Association, Piscataway, NJ, September 16, 1997

[4] Hans van Vliet, Software Engineering: Principles and Practice, John Wiley & Sons, Ltd., 2000

[5] Microsoft, *C# Reference*, msdn.microsoft.com, http://msdn.microsoft.com/en-us/library/618ayhy6.aspx, 2010.

[6] Andrew Troelsen, *Pro C# 2008 and the .NET 3.5 Platform 4th Edition*, Apress, 2007

[7] Charles Petzold, *Programming Microsoft Windows Forms (Pro Developer)*, Microsoft Press, 2005

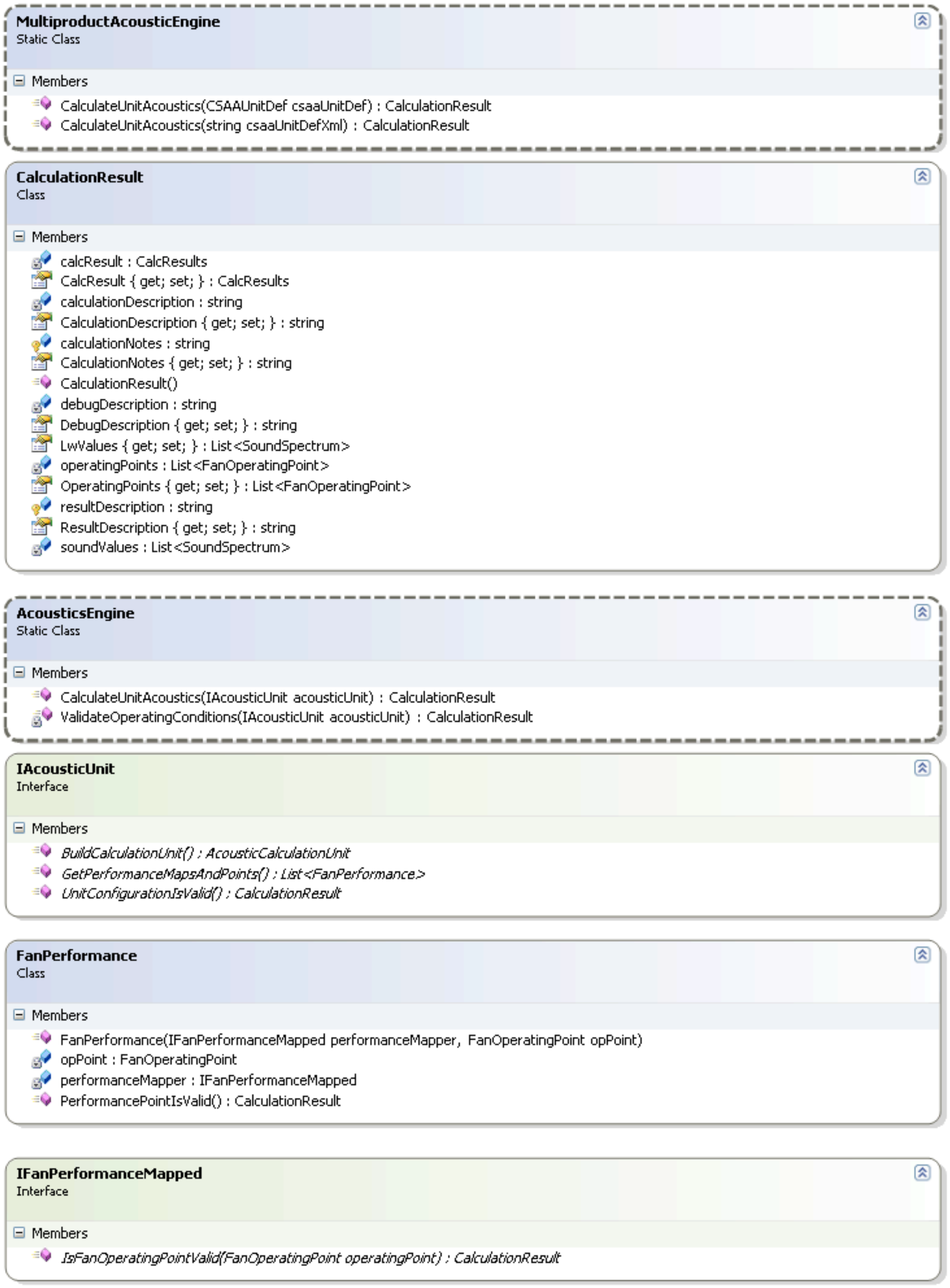
[8] ASHRAE Research Group, *2004 ASHRAE Handbook: HVAC Systems and Equipment*, American Society of Heating, Refrigerating and Air-Conditioning Engineers, 2004

[9] Travis Fischer, *The Acoustic Calculation Algorithm*, June 30th, 2009, Revision 1

11 Appendices

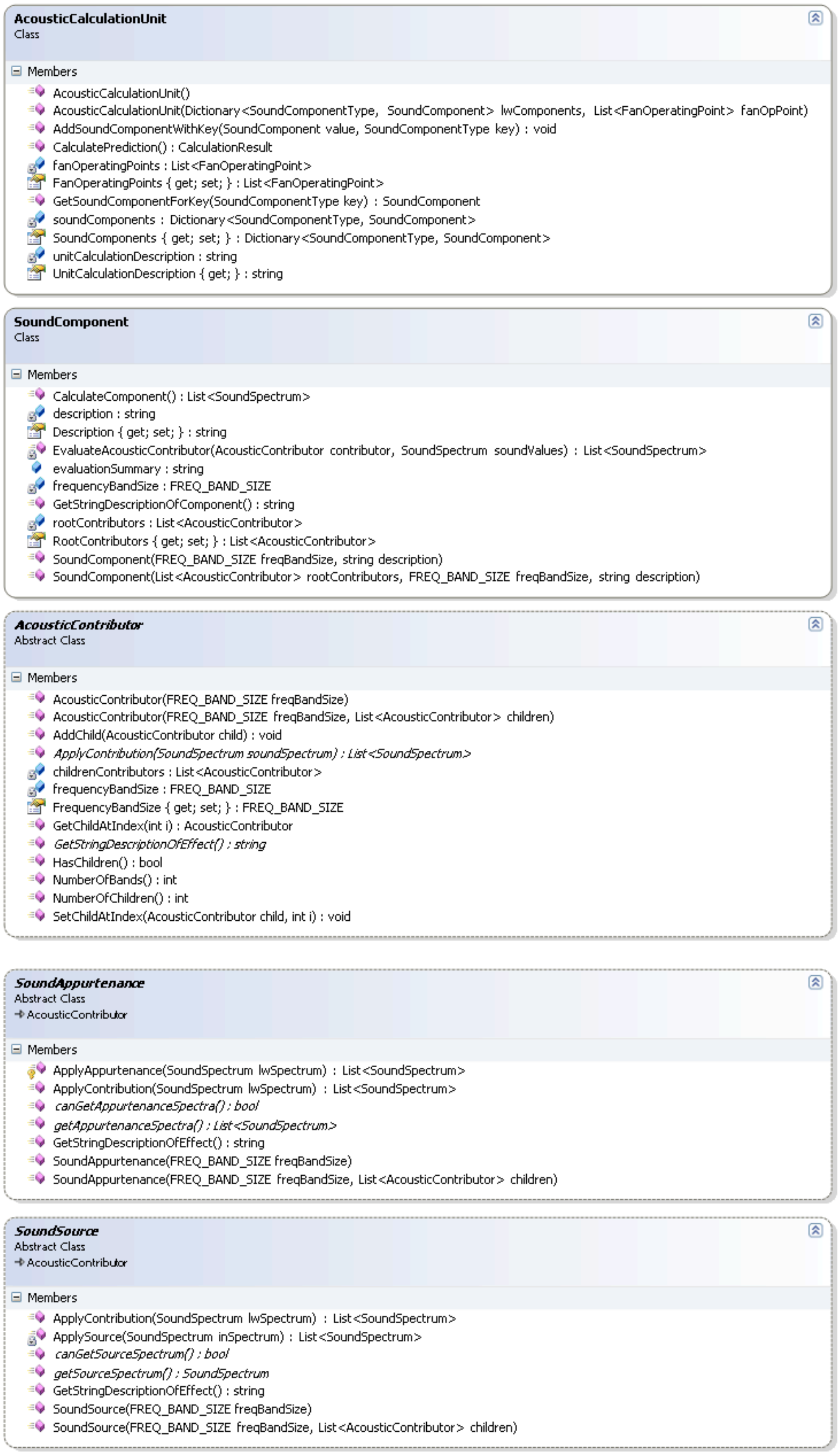
Appendix A. Acoustics Library Core Detailed Class Diagrams

Figure 15. Acoustics Library Core Classes



```
MultiproductAcousticEngine
Static Class

Members
  CalculateUnitAcoustics(CSAAUnitDef csaaUnitDef) : CalculationResult
  CalculateUnitAcoustics(string csaaUnitDefXml) : CalculationResult

CalculationResult
Class

Members
  calcResult : CalcResults
  CalcResult { get; set; } : CalcResults
  calculationDescription : string
  CalculationDescription { get; set; } : string
  calculationNotes : string
  CalculationNotes { get; set; } : string
  CalculationResult()
  debugDescription : string
  DebugDescription { get; set; } : string
  LwValues { get; set; } : List<SoundSpectrum>
  operatingPoints : List<FanOperatingPoint>
  OperatingPoints { get; set; } : List<FanOperatingPoint>
  resultDescription : string
  ResultDescription { get; set; } : string
  soundValues : List<SoundSpectrum>

AcousticsEngine
Static Class

Members
  CalculateUnitAcoustics(IAcousticUnit acousticUnit) : CalculationResult
  ValidateOperatingConditions(IAcousticUnit acousticUnit) : CalculationResult

IAcousticUnit
Interface

Members
  BuildCalculationUnit() : AcousticCalculationUnit
  GetPerformanceMapsAndPoints() : List<FanPerformance>
  UnitConfigurationIsValid() : CalculationResult

FanPerformance
Class

Members
  FanPerformance(IFanPerformanceMapped performanceMapper, FanOperatingPoint opPoint)
  opPoint : FanOperatingPoint
  performanceMapper : IFanPerformanceMapped
  PerformancePointIsValid() : CalculationResult

IFanPerformanceMapped
Interface

Members
  IsFanOperatingPointValid(FanOperatingPoint operatingPoint) : CalculationResult
```

Figure 15. Acoustics Library Core Classes



```
AcousticCalculationUnit
Class

Members
  AcousticCalculationUnit()
  AcousticCalculationUnit(Dictionary<SoundComponentType, SoundComponent> lwComponents, List<FanOperatingPoint> fanOpPoint)
  AddSoundComponentWithKey(SoundComponent value, SoundComponentType key) : void
  CalculatePrediction() : CalculationResult
  fanOperatingPoints : List<FanOperatingPoint>
  FanOperatingPoints { get; set; } : List<FanOperatingPoint>
  GetSoundComponentForKey(SoundComponentType key) : SoundComponent
  soundComponents : Dictionary<SoundComponentType, SoundComponent>
  SoundComponents { get; set; } : Dictionary<SoundComponentType, SoundComponent>
  unitCalculationDescription : string
  UnitCalculationDescription { get; } : string
```

```
SoundComponent
Class

Members
  CalculateComponent() : List<SoundSpectrum>
  description : string
  Description { get; set; } : string
  EvaluateAcousticContributor(AcousticContributor contributor, SoundSpectrum soundValues) : List<SoundSpectrum>
  evaluationSummary : string
  frequencyBandSize : FREQ_BAND_SIZE
  GetStringDescriptionOfComponent() : string
  rootContributors : List<AcousticContributor>
  RootContributors { get; set; } : List<AcousticContributor>
  SoundComponent(FREQ_BAND_SIZE freqBandSize, string description)
  SoundComponent(List<AcousticContributor> rootContributors, FREQ_BAND_SIZE freqBandSize, string description)
```

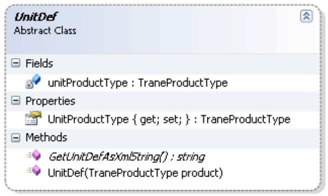
```
AcousticContributor
Abstract Class

Members
  AcousticContributor(FREQ_BAND_SIZE freqBandSize)
  AcousticContributor(FREQ_BAND_SIZE freqBandSize, List<AcousticContributor> children)
  AddChild(AcousticContributor child) : void
  ApplyContribution(SoundSpectrum soundSpectrum) : List<SoundSpectrum>
  childrenContributors : List<AcousticContributor>
  frequencyBandSize : FREQ_BAND_SIZE
  FrequencyBandSize { get; set; } : FREQ_BAND_SIZE
  GetChildAtIndex(int i) : AcousticContributor
  GetStringDescriptionOfEffect() : string
  HasChildren() : bool
  NumberOfBands() : int
  NumberOfChildren() : int
  SetChildAtIndex(AcousticContributor child, int i) : void
```

```
SoundAppurtenance
Abstract Class
→ AcousticContributor

Members
  ApplyAppurtenance(SoundSpectrum lwSpectrum) : List<SoundSpectrum>
  ApplyContribution(SoundSpectrum lwSpectrum) : List<SoundSpectrum>
  canGetAppurtenanceSpectra() : bool
  getAppurtenanceSpectra() : List<SoundSpectrum>
  GetStringDescriptionOfEffect() : string
  SoundAppurtenance(FREQ_BAND_SIZE freqBandSize)
  SoundAppurtenance(FREQ_BAND_SIZE freqBandSize, List<AcousticContributor> children)
```

```
SoundSource
Abstract Class
→ AcousticContributor

Members
  ApplyContribution(SoundSpectrum lwSpectrum) : List<SoundSpectrum>
  ApplySource(SoundSpectrum inSpectrum) : List<SoundSpectrum>
  canGetSourceSpectrum() : bool
  getSourceSpectrum() : SoundSpectrum
  GetStringDescriptionOfEffect() : string
  SoundSource(FREQ_BAND_SIZE freqBandSize)
  SoundSource(FREQ_BAND_SIZE freqBandSize, List<AcousticContributor> children)
```

Appendix B. Unit Definition Detailed Class Diagrams

Figure 16. Unit Definition Detailed Class Diagrams

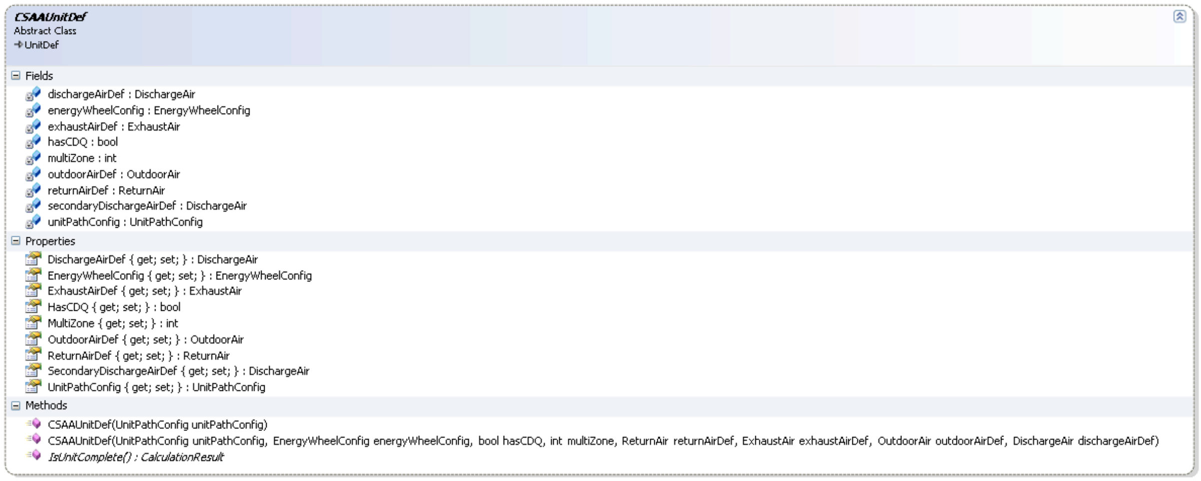


UnitDef
Abstract Class

Fields
- unitProductType : TraneProductType

Properties
- UnitProductType { get; set; } : TraneProductType

Methods
- *GetUnitDefAsXmlString() : string*
- UnitDef(TraneProductType product)

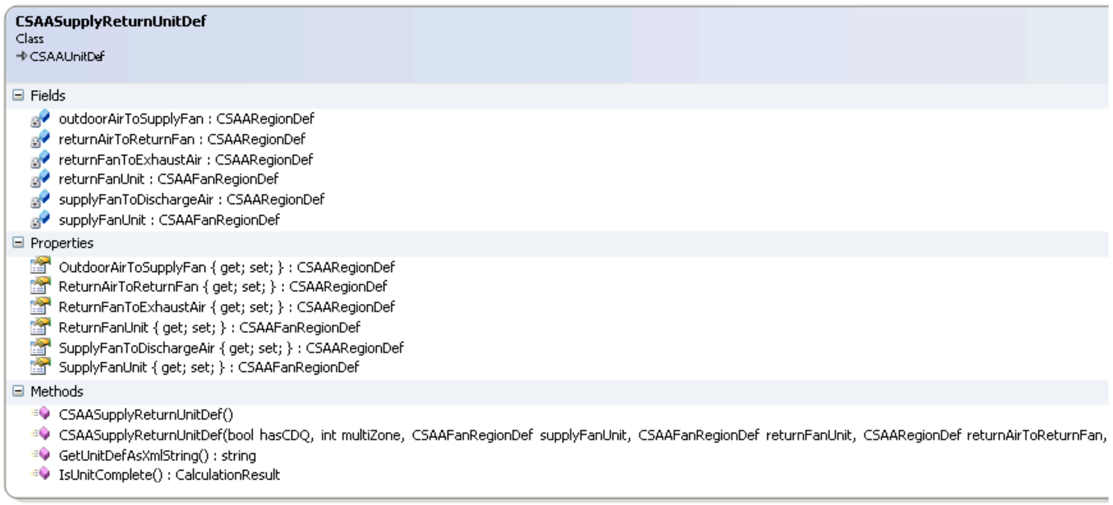


CSAAUnitDef
Abstract Class
→UnitDef

Fields
- dischargeAirDef : DischargeAir
- energyWheelConfig : EnergyWheelConfig
- exhaustAirDef : ExhaustAir
- hasCDQ : bool
- multiZone : int
- outdoorAirDef : OutdoorAir
- returnAirDef : ReturnAir
- secondaryDischargeAirDef : DischargeAir
- unitPathConfig : UnitPathConfig

Properties
- DischargeAirDef { get; set; } : DischargeAir
- EnergyWheelConfig { get; set; } : EnergyWheelConfig
- ExhaustAirDef { get; set; } : ExhaustAir
- HasCDQ { get; set; } : bool
- MultiZone { get; set; } : int
- OutdoorAirDef { get; set; } : OutdoorAir
- ReturnAirDef { get; set; } : ReturnAir
- SecondaryDischargeAirDef { get; set; } : DischargeAir
- UnitPathConfig { get; set; } : UnitPathConfig

Methods
- CSAAUnitDef(UnitPathConfig unitPathConfig)
- CSAAUnitDef(UnitPathConfig unitPathConfig, EnergyWheelConfig energyWheelConfig, bool hasCDQ, int multiZone, ReturnAir returnAirDef, ExhaustAir exhaustAirDef, OutdoorAir outdoorAirDef, DischargeAir dischargeAirDef)
- *IsUnitComplete() : CalculationResult*

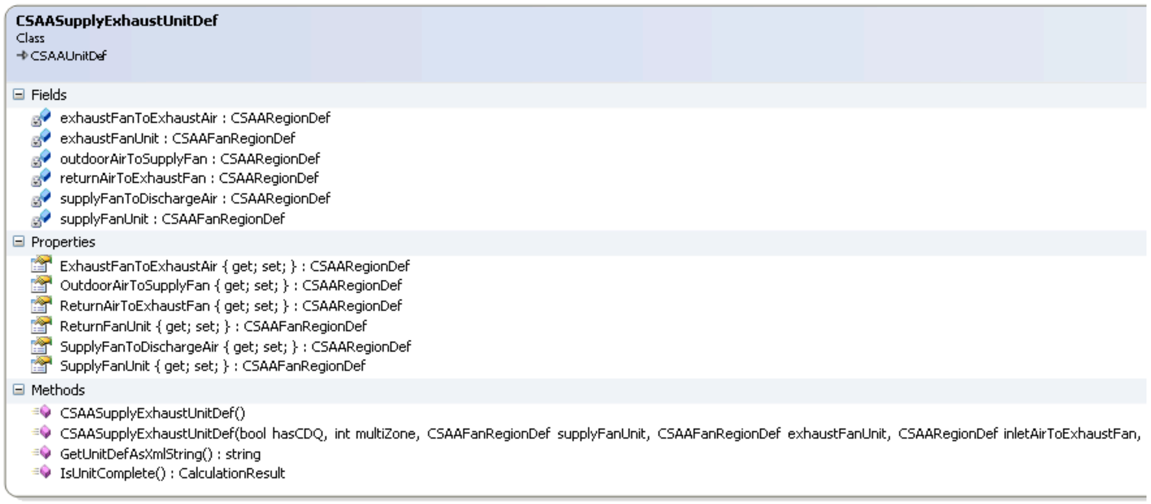


CSAASupplyReturnUnitDef
Class
→CSAAUnitDef

Fields
- outdoorAirToSupplyFan : CSAARegionDef
- returnAirToReturnFan : CSAARegionDef
- returnFanToExhaustAir : CSAARegionDef
- returnFanUnit : CSAAFanRegionDef
- supplyFanToDischargeAir : CSAARegionDef
- supplyFanUnit : CSAAFanRegionDef

Properties
- OutdoorAirToSupplyFan { get; set; } : CSAARegionDef
- ReturnAirToReturnFan { get; set; } : CSAARegionDef
- ReturnFanToExhaustAir { get; set; } : CSAARegionDef
- ReturnFanUnit { get; set; } : CSAAFanRegionDef
- SupplyFanToDischargeAir { get; set; } : CSAARegionDef
- SupplyFanUnit { get; set; } : CSAAFanRegionDef

Methods
- CSAASupplyReturnUnitDef()
- CSAASupplyReturnUnitDef(bool hasCDQ, int multiZone, CSAAFanRegionDef supplyFanUnit, CSAAFanRegionDef returnFanUnit, CSAARegionDef returnAirToReturnFan,
- GetUnitDefAsXmlString() : string
- IsUnitComplete() : CalculationResult

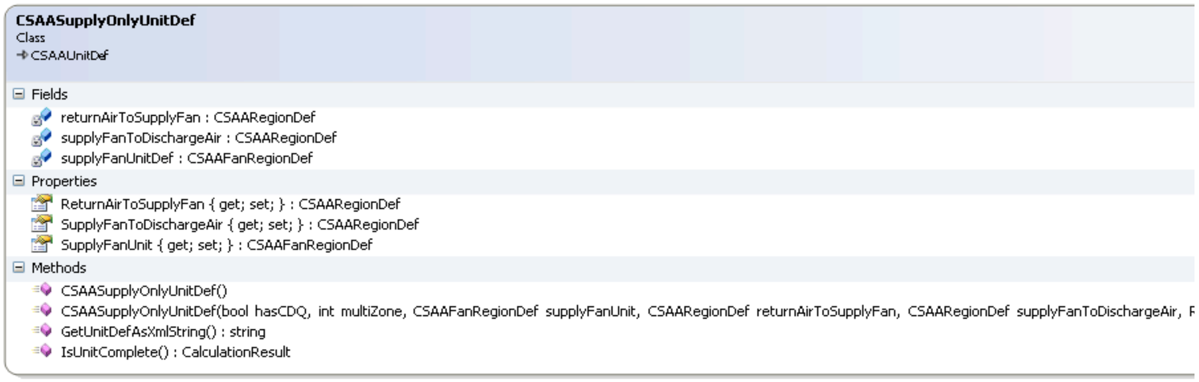


CSAASupplyExhaustUnitDef
Class
→CSAAUnitDef

Fields
- exhaustFanToExhaustAir : CSAARegionDef
- exhaustFanUnit : CSAAFanRegionDef
- outdoorAirToSupplyFan : CSAARegionDef
- returnAirToExhaustFan : CSAARegionDef
- supplyFanToDischargeAir : CSAARegionDef
- supplyFanUnit : CSAAFanRegionDef

Properties
- ExhaustFanToExhaustAir { get; set; } : CSAARegionDef
- OutdoorAirToSupplyFan { get; set; } : CSAARegionDef
- ReturnAirToExhaustFan { get; set; } : CSAARegionDef
- ReturnFanUnit { get; set; } : CSAAFanRegionDef
- SupplyFanToDischargeAir { get; set; } : CSAARegionDef
- SupplyFanUnit { get; set; } : CSAAFanRegionDef

Methods
- CSAASupplyExhaustUnitDef()
- CSAASupplyExhaustUnitDef(bool hasCDQ, int multiZone, CSAAFanRegionDef supplyFanUnit, CSAAFanRegionDef exhaustFanUnit, CSAARegionDef inletAirToExhaustFan,
- GetUnitDefAsXmlString() : string
- IsUnitComplete() : CalculationResult

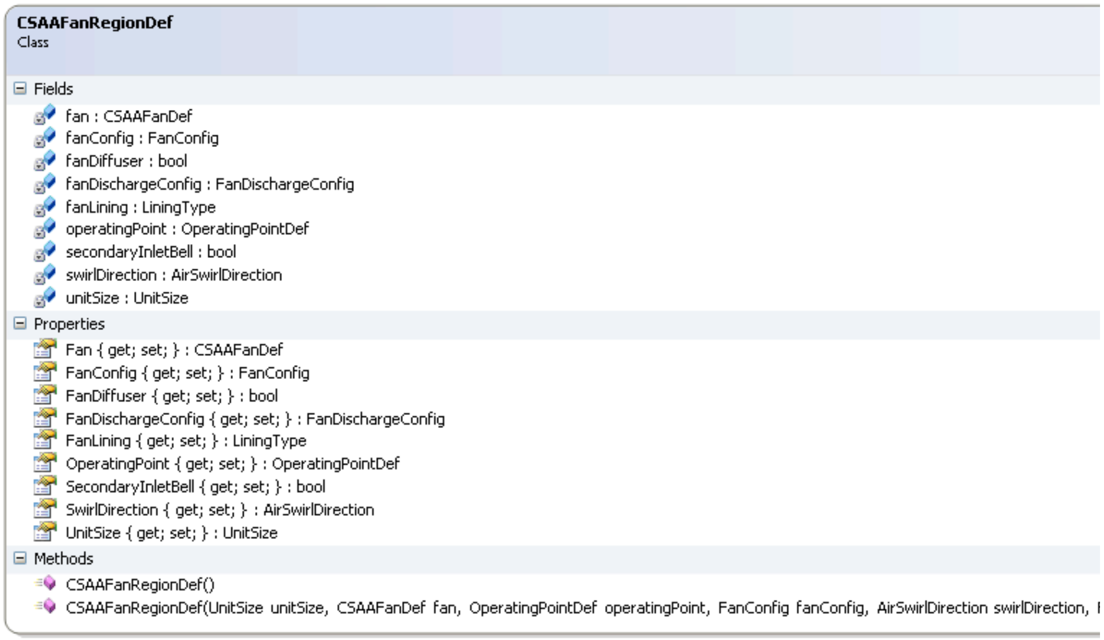


CSAASupplyOnlyUnitDef
Class
→CSAAUnitDef

Fields
- returnAirToSupplyFan : CSAARegionDef
- supplyFanToDischargeAir : CSAARegionDef
- supplyFanUnitDef : CSAAFanRegionDef

Properties
- ReturnAirToSupplyFan { get; set; } : CSAARegionDef
- SupplyFanToDischargeAir { get; set; } : CSAARegionDef
- SupplyFanUnit { get; set; } : CSAAFanRegionDef

Methods
- CSAASupplyOnlyUnitDef()
- CSAASupplyOnlyUnitDef(bool hasCDQ, int multiZone, CSAAFanRegionDef supplyFanUnit, CSAARegionDef returnAirToSupplyFan, CSAARegionDef supplyFanToDischargeAir, F
- GetUnitDefAsXmlString() : string
- IsUnitComplete() : CalculationResult

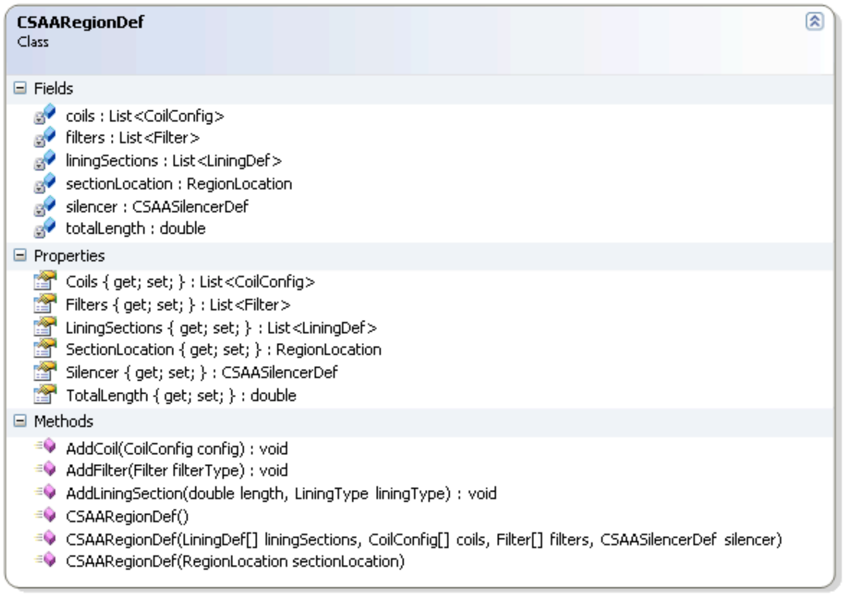


CSAAFanRegionDef
Class

Fields
- fan : CSAAFanDef
- fanConfig : FanConfig
- fanDiffuser : bool
- fanDischargeConfig : FanDischargeConfig
- fanLining : LiningType
- operatingPoint : OperatingPointDef
- secondaryInletBell : bool
- swirlDirection : AirSwirlDirection
- unitSize : UnitSize

Properties
- Fan { get; set; } : CSAAFanDef
- FanConfig { get; set; } : FanConfig
- FanDiffuser { get; set; } : bool
- FanDischargeConfig { get; set; } : FanDischargeConfig
- FanLining { get; set; } : LiningType
- OperatingPoint { get; set; } : OperatingPointDef
- SecondaryInletBell { get; set; } : bool
- SwirlDirection { get; set; } : AirSwirlDirection
- UnitSize { get; set; } : UnitSize

Methods
- CSAAFanRegionDef()
- CSAAFanRegionDef(UnitSize unitSize, CSAAFanDef fan, OperatingPointDef operatingPoint, FanConfig fanConfig, AirSwirlDirection swirlDirection,



CSAARegionDef
Class

Fields
- coils : List<CoilConfig>
- filters : List<Filter>
- liningSections : List<LiningDef>
- sectionLocation : RegionLocation
- silencer : CSAASilencerDef
- totalLength : double

Properties
- Coils { get; set; } : List<CoilConfig>
- Filters { get; set; } : List<Filter>
- LiningSections { get; set; } : List<LiningDef>
- SectionLocation { get; set; } : RegionLocation
- Silencer { get; set; } : CSAASilencerDef
- TotalLength { get; set; } : double

Methods
- AddCoil(CoilConfig config) : void
- AddFilter(Filter filterType) : void
- AddLiningSection(double length, LiningType liningType) : void
- CSAARegionDef()
- CSAARegionDef(LiningDef[] liningSections, CoilConfig[] coils, Filter[] filters, CSAASilencerDef silencer)
- CSAARegionDef(RegionLocation sectionLocation)

Appendix C. Acoustics Library Database Design Diagrams

Figure 17. Database Design E.R.
Diagram

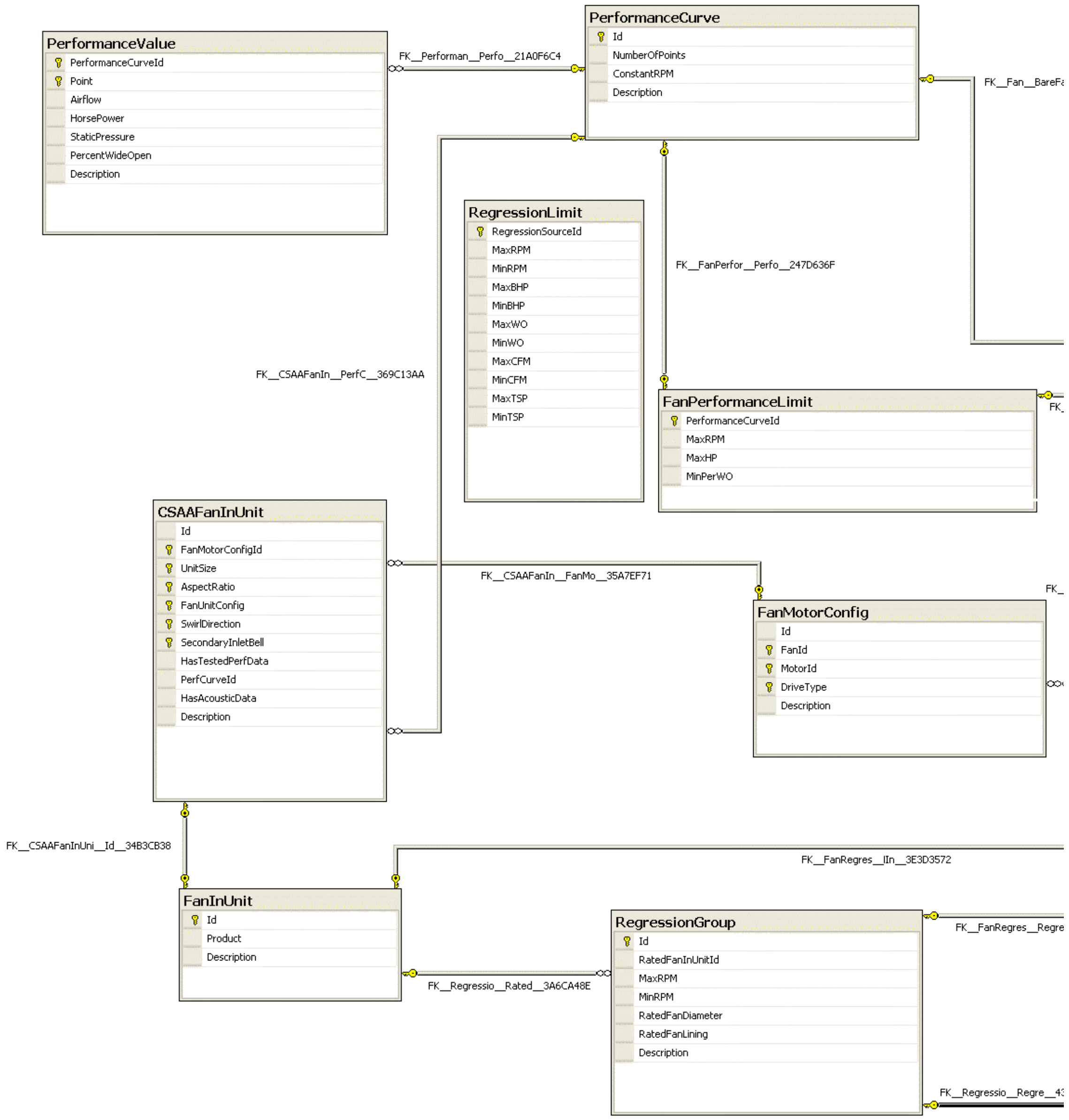


Figure 17.1 Database Design E.R. Diagram

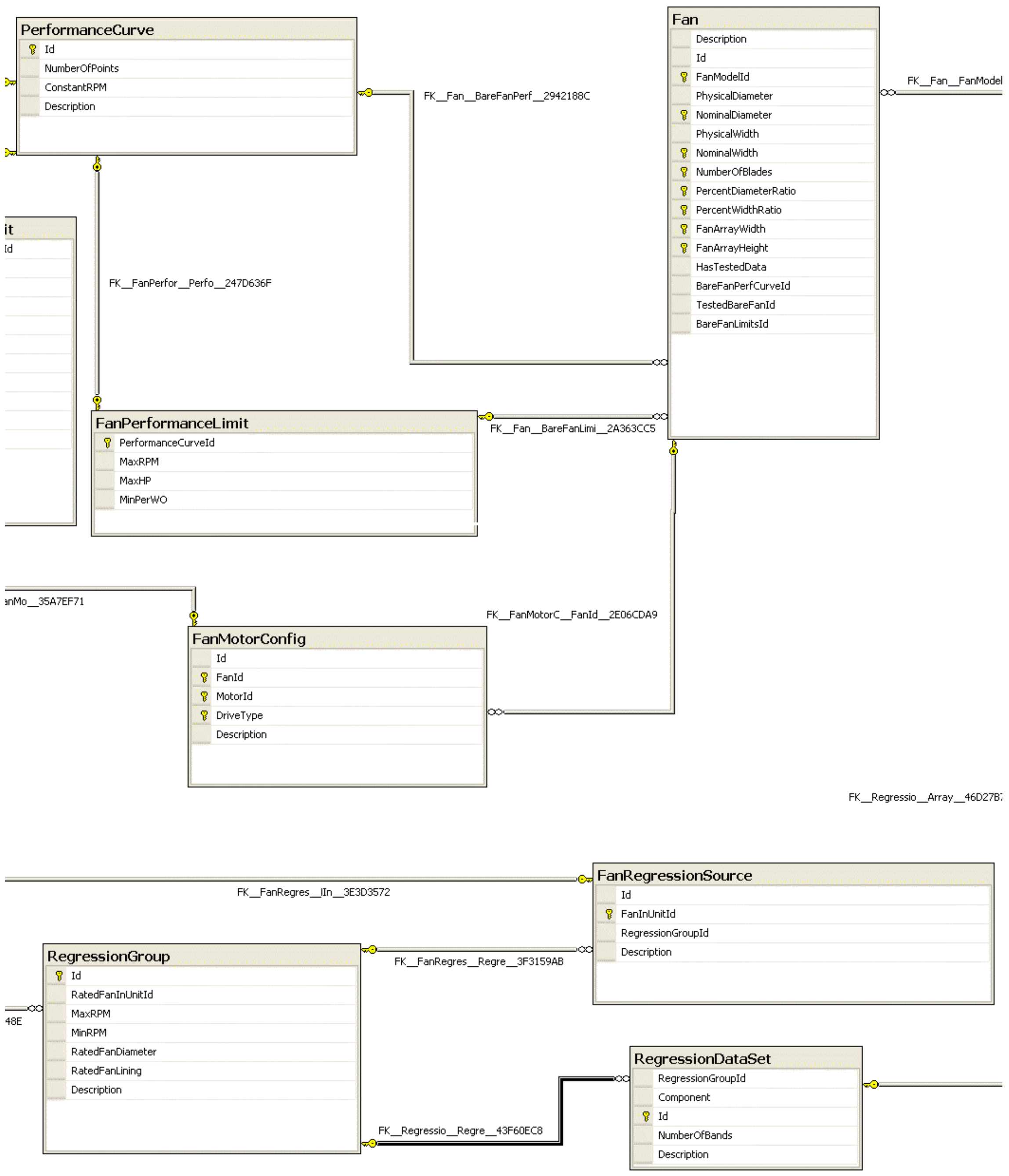


Figure 17.2 Database Design E.R. Diagram

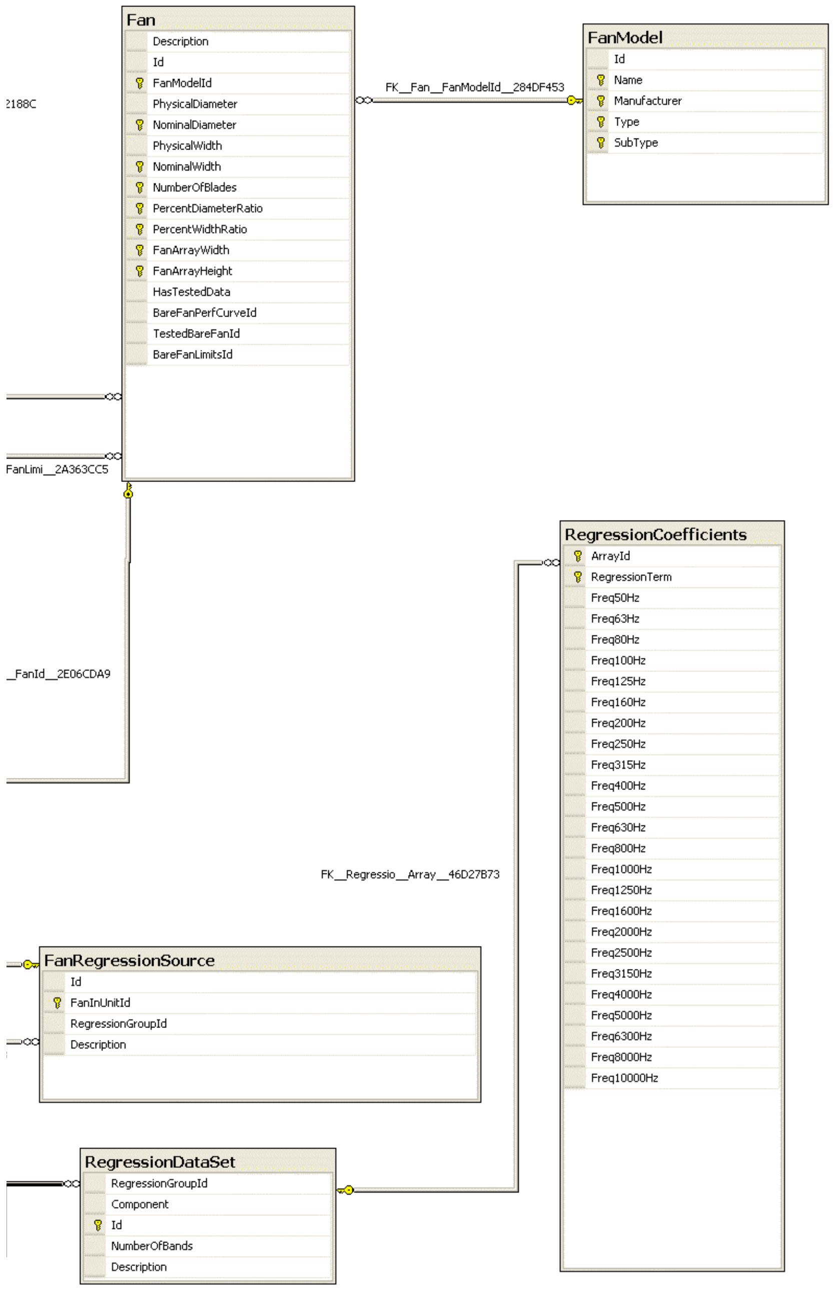


Figure 17.3 Database Design E.R. Diagram

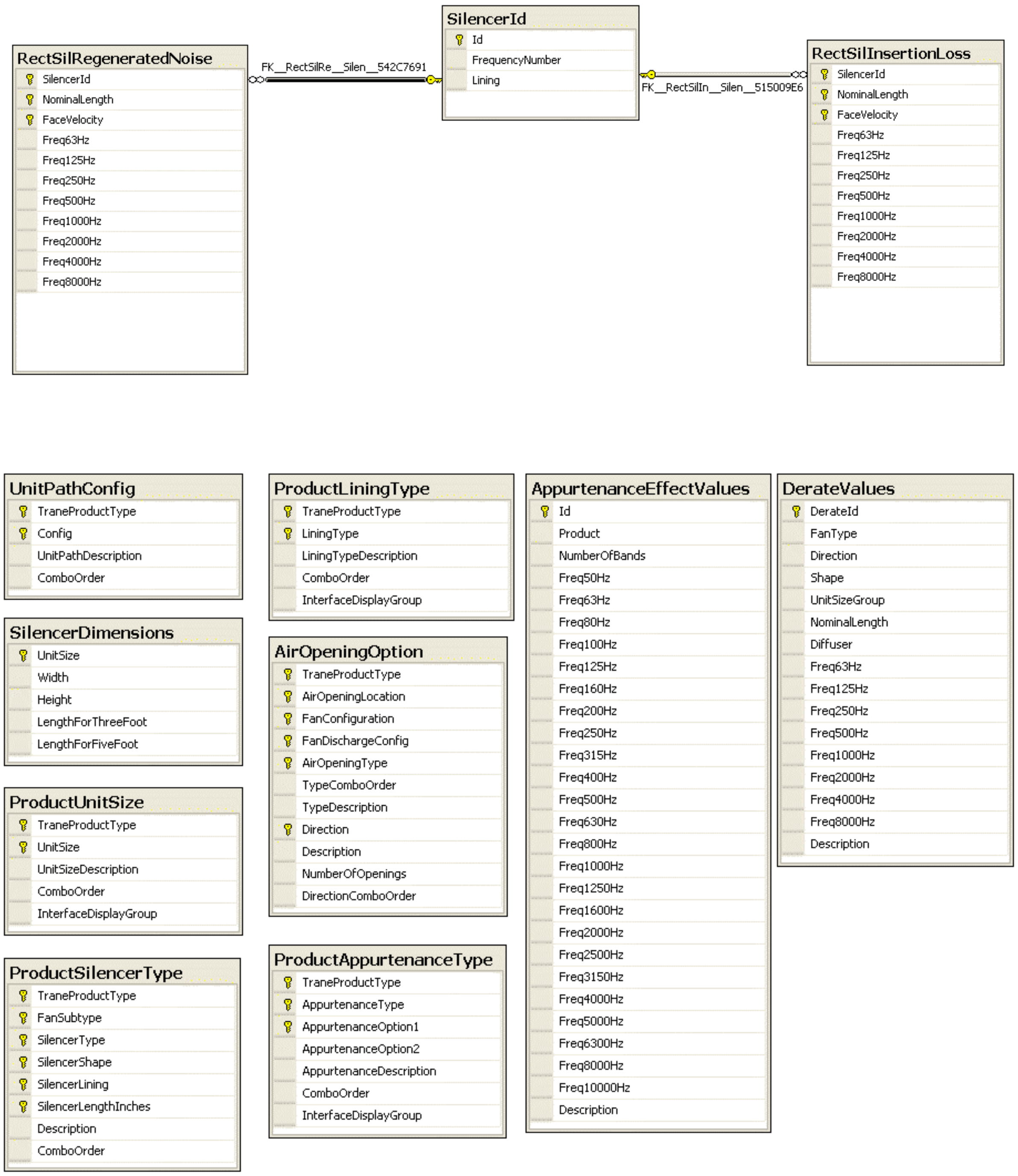
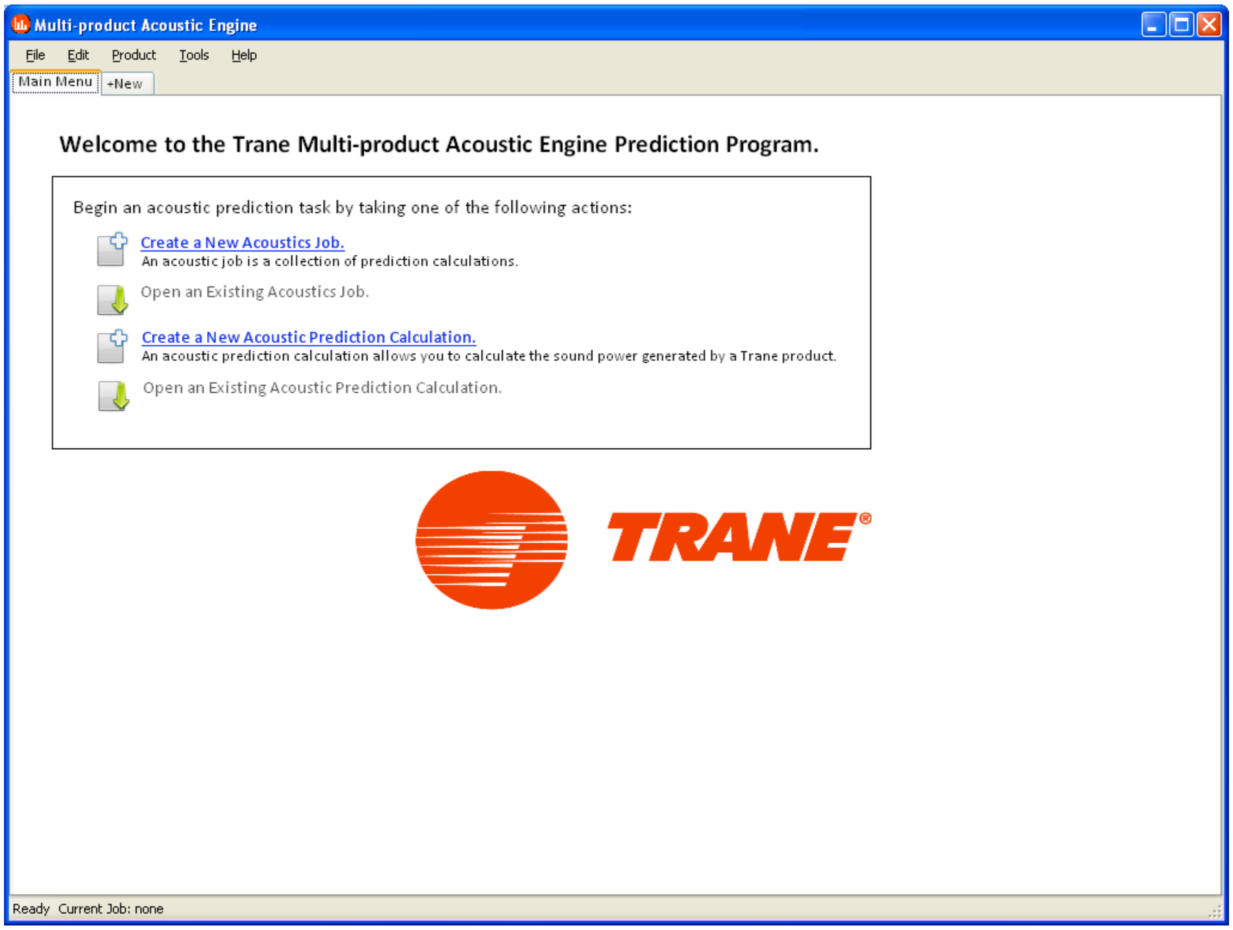


Figure 17.4 Database Design E.R. Diagram

Appendix D. Final GUI Design Screenshots



Welcome to the Trane Multi-product Acoustic Engine Prediction Program.

Begin an acoustic prediction task by taking one of the following actions:

Create a New Acoustics Job.
An acoustic job is a collection of prediction calculations.

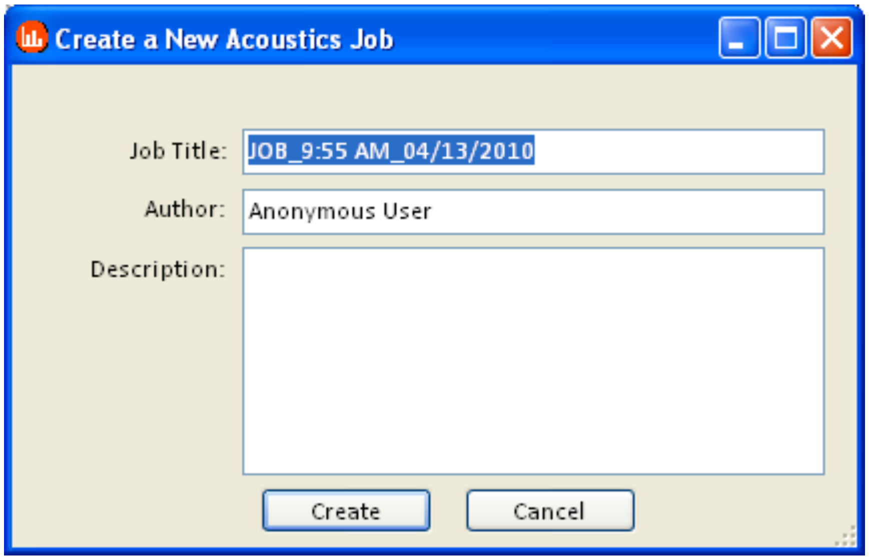
Open an Existing Acoustics Job.

Create a New Acoustic Prediction Calculation.
An acoustic prediction calculation allows you to calculate the sound power generated by a Trane product.

Open an Existing Acoustic Prediction Calculation.

Ready Current Job: none

Figure 18. Welcome Screen Screenshot

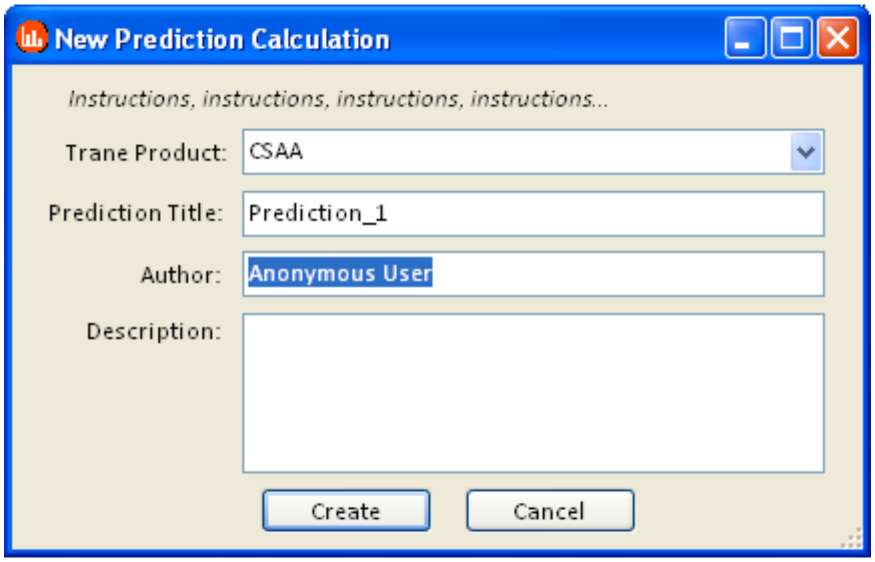

Create a New Acoustics Job

Job Title: JOB_9:55 AM_04/13/2010

Author: Anonymous User

Description:

Figure 19. Create Acoustic Job Dialog Screenshot


New Prediction Calculation

Instructions, instructions, instructions, instructions...

Trane Product: CSAA

Prediction Title: Prediction_1

Author: Anonymous User

Description:

Figure 20. Create Acoustic Prediction Dialog Screenshot

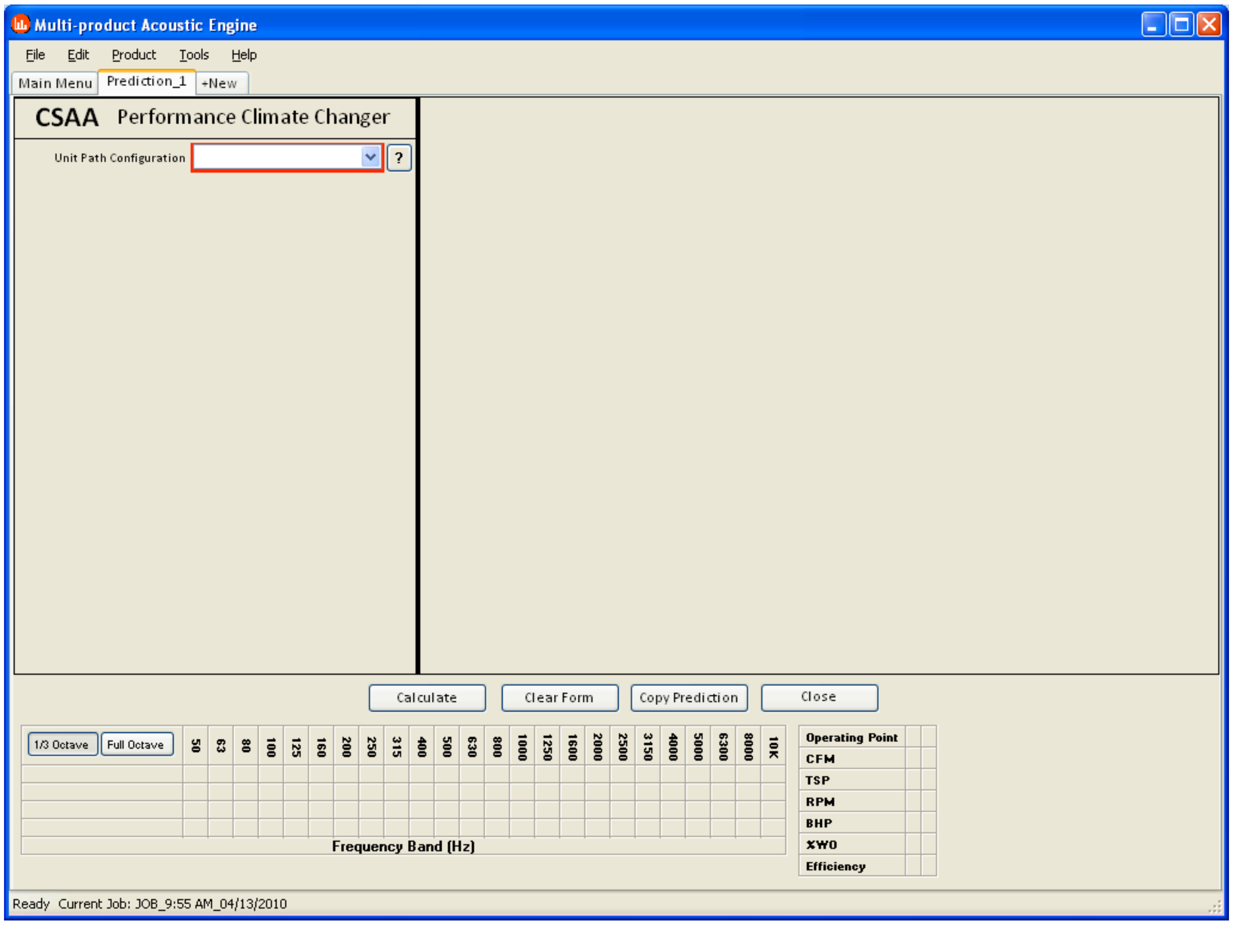
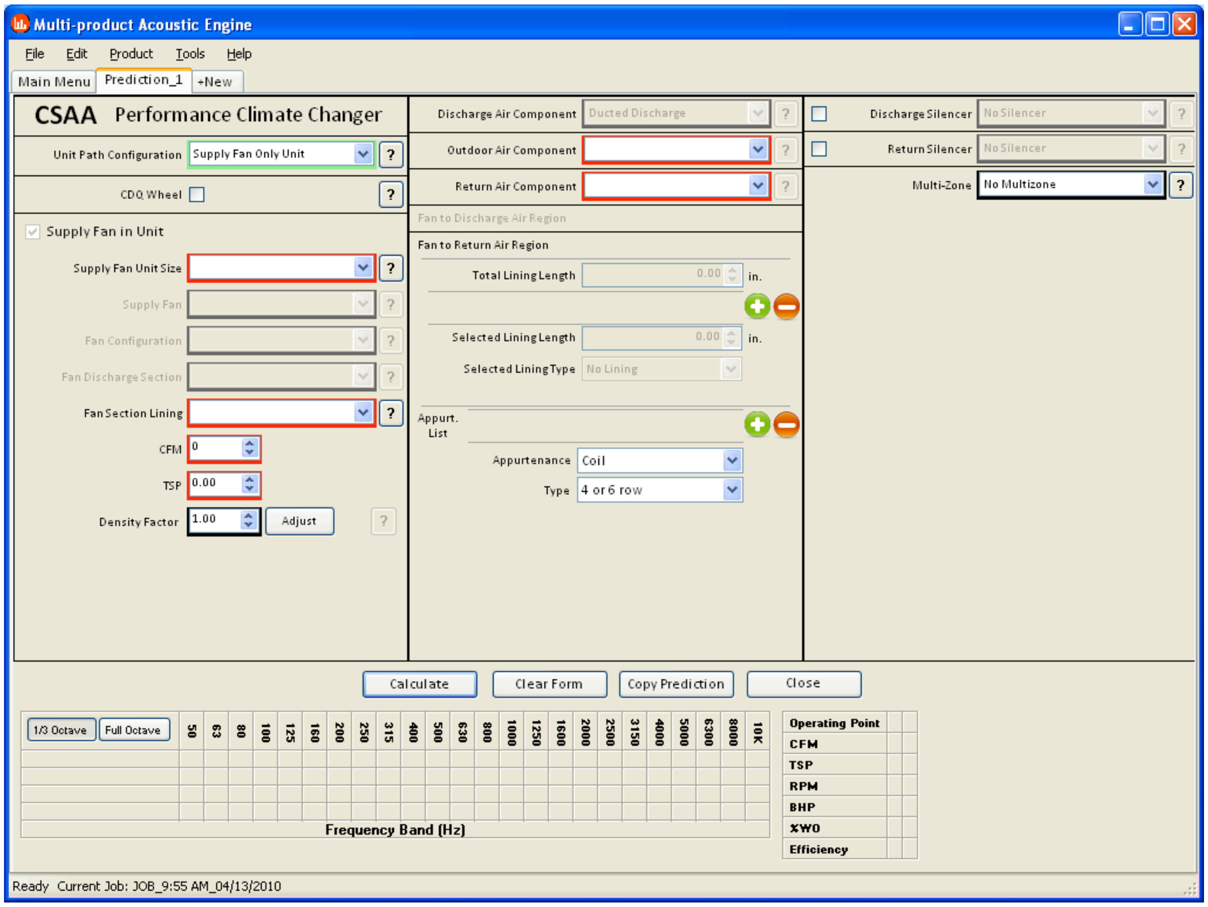


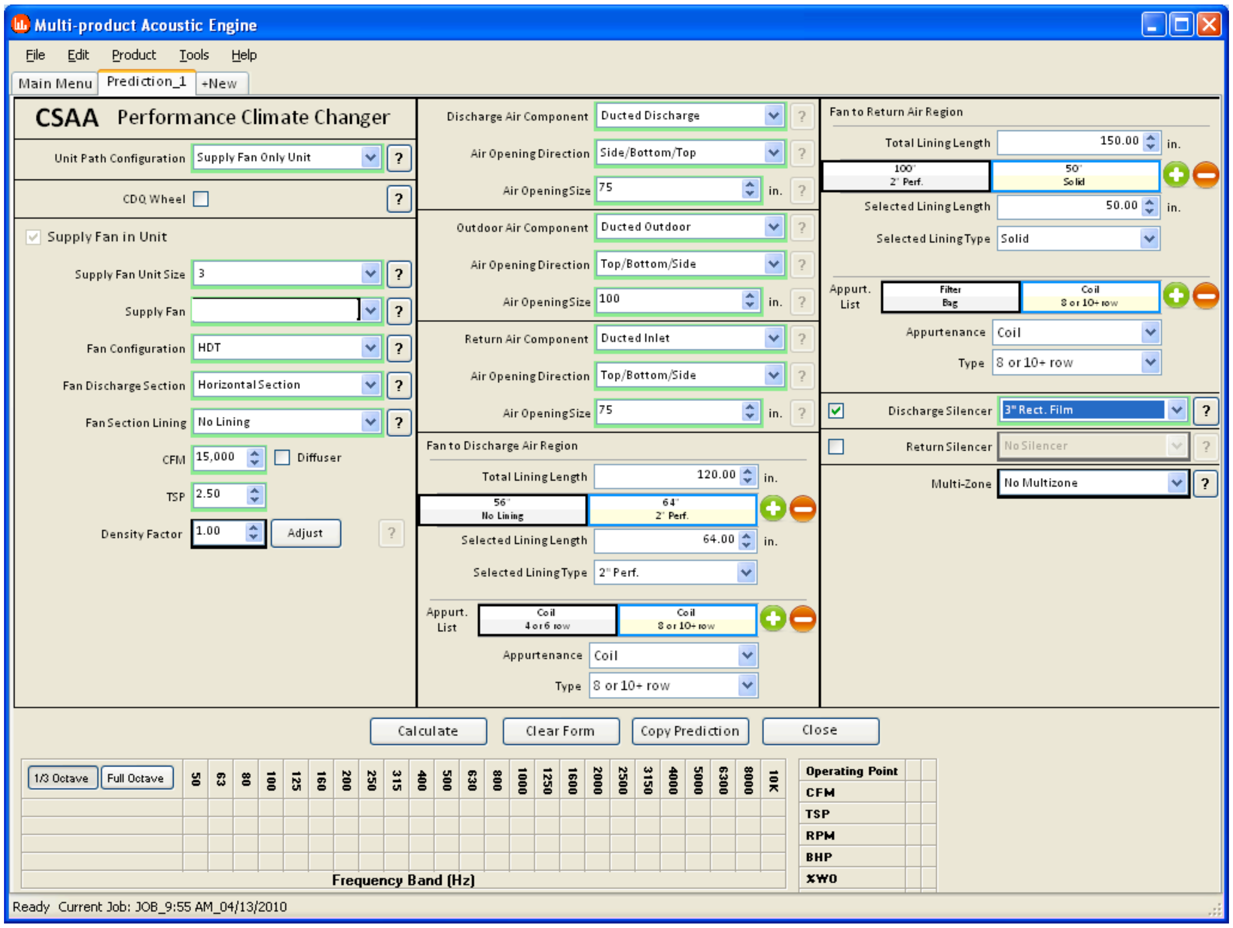
Figure 21. Initial CSAA Prediction Interface Screenshot



Unit Path Configuration	Supply Fan Only Unit
CDQ Wheel	☐
Supply Fan in Unit	✓
CFM	0
TSP	0.00
Density Factor	1.00
Discharge Air Component	Ducted Discharge
Total Lining Length	0.00 in.
Selected Lining Length	0.00 in.
Selected Lining Type	No Lining
Appurtenance	Coil
Type	4 or 6 row
Discharge Silencer	☐ No Silencer
Return Silencer	☐ No Silencer
Multi-Zone	No Multizone

Ready Current Job: JOB_9:55 AM_04/13/2010

Figure 22. Supply Fan Only Prediction Interface Screenshot



CSAA Performance Climate Changer

Unit Path Configuration: Supply Fan Only Unit

CDQ Wheel: ☐

☑ Supply Fan in Unit

Supply Fan Unit Size: 3

Supply Fan:

Fan Configuration: HDT

Fan Discharge Section: Horizontal Section

Fan Section Lining: No Lining

CFM: 15,000 ☐ Diffuser

TSP: 2.50

Density Factor: 1.00 Adjust

Discharge Air Component: Ducted Discharge

Air Opening Direction: Side/Bottom/Top

Air Opening Size: 75 in.

Outdoor Air Component: Ducted Outdoor

Air Opening Direction: Top/Bottom/Side

Air Opening Size: 100 in.

Return Air Component: Ducted Inlet

Air Opening Direction: Top/Bottom/Side

Air Opening Size: 75 in.

Fan to Discharge Air Region

Total Lining Length: 120.00 in.

56" No Lining	64" 2" Perf.

Selected Lining Length: 64.00 in.

Selected Lining Type: 2" Perf.

Appurt. List:

Coil 4 or 6 row	Coil 8 or 10+ row

Appurtenance: Coil

Type: 8 or 10+ row

Fan to Return Air Region

Total Lining Length: 150.00 in.

100" 2" Perf.	50" Solid

Selected Lining Length: 50.00 in.

Selected Lining Type: Solid

Appurt. List:

Filter Bag	Coil 8 or 10+ row

Appurtenance: Coil

Type: 8 or 10+ row

☑ Discharge Silencer: 3" Rect. Film

☐ Return Silencer: No Silencer

Multi-Zone: No Multizone

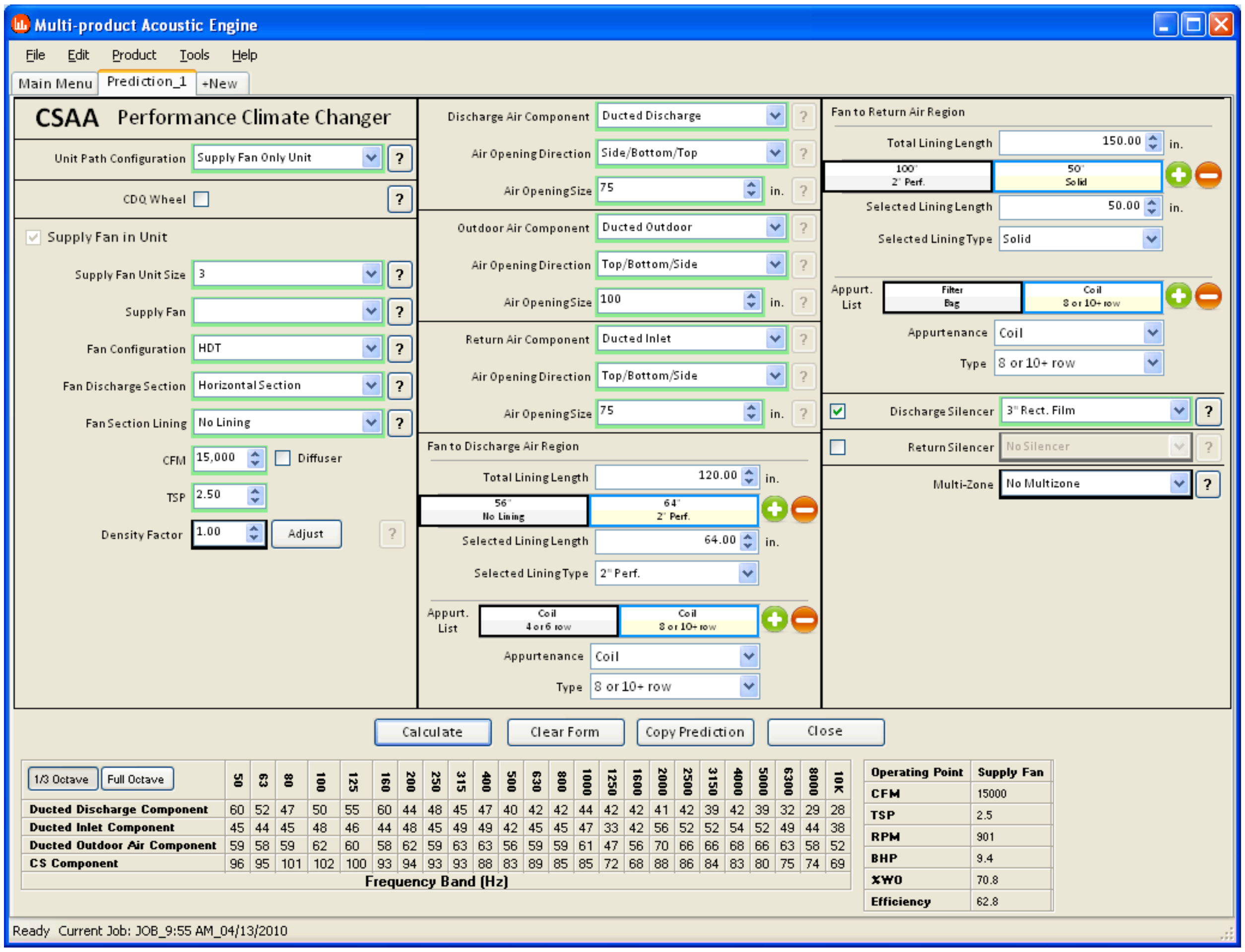
	50	63	80	100	125	160	200	250	315	400	500	630	800	1000	1250	1600	2000	2500	3150	4000	5000	6300	8000	10K

Frequency Band (Hz)

Operating Point		
CFM		
TSP		
RPM		
BHP		
XWO		

Ready Current Job: JOB_9:55 AM_04/13/2010

Figure 23. Single Fan Unit Fully Configured Screenshot



CSAA Performance Climate Changer

Unit Path Configuration	Supply Fan Only Unit
CDQ Wheel	☐
Supply Fan in Unit	✓
Supply Fan Unit Size	3
Supply Fan	
Fan Configuration	HDT
Fan Discharge Section	Horizontal Section
Fan Section Lining	No Lining
CFM	15,000
Diffuser	☐
TSP	2.50
Density Factor	1.00

Discharge Air Component	Ducted Discharge
Air Opening Direction	Side/Bottom/Top
Air Opening Size	75 in.
Outdoor Air Component	Ducted Outdoor
Air Opening Direction	Top/Bottom/Side
Air Opening Size	100 in.
Return Air Component	Ducted Inlet
Air Opening Direction	Top/Bottom/Side
Air Opening Size	75 in.

Fan to Discharge Air Region

Total Lining Length	120.00 in.
	56" No Lining / 64" 2" Perf.
Selected Lining Length	64.00 in.
Selected Lining Type	2" Perf.
Appurt. List	Coil 4 or 6 row / Coil 8 or 10+ row
Appurtenance	Coil
Type	8 or 10+ row

Fan to Return Air Region

Total Lining Length	150.00 in.
	100" 2" Perf. / 50" Solid
Selected Lining Length	50.00 in.
Selected Lining Type	Solid
Appurt. List	Filter Bag / Coil 8 or 10+ row
Appurtenance	Coil
Type	8 or 10+ row
Discharge Silencer ✓	3" Rect. Film
Return Silencer ☐	No Silencer
Multi-Zone	No Multizone

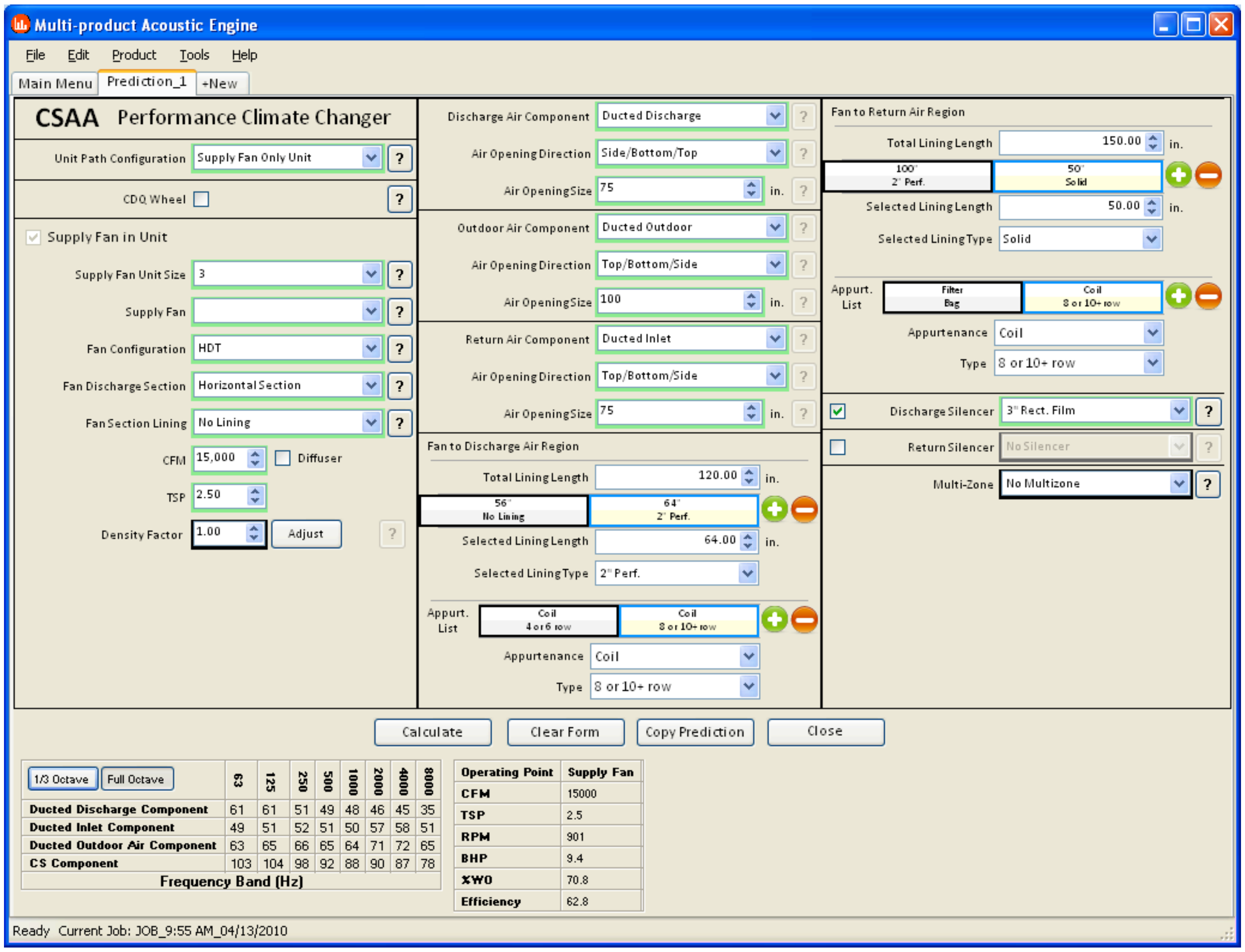
	50	63	80	100	125	160	200	250	315	400	500	630	800	1000	1250	1600	2000	2500	3150	4000	5000	6300	8000	10K
Ducted Discharge Component	60	52	47	50	55	60	44	48	45	47	40	42	42	44	42	42	41	42	39	42	39	32	29	28
Ducted Inlet Component	45	44	45	48	46	44	48	45	49	49	42	45	45	47	33	42	56	52	52	54	52	49	44	38
Ducted Outdoor Air Component	59	58	59	62	60	58	62	59	63	63	56	59	59	61	47	56	70	66	66	68	66	63	58	52
CS Component	96	95	101	102	100	93	94	93	93	88	83	89	85	85	72	68	88	86	84	83	80	75	74	69

Frequency Band (Hz)

Operating Point	Supply Fan
CFM	15000
TSP	2.5
RPM	901
BHP	9.4
XWO	70.8
Efficiency	62.8

Ready Current Job: JOB_9:55 AM_04/13/2010

Figure 24. Single Fan Unit Calculated 1/3 Octave Lw Screenshot


Multi-product Acoustic Engine

CSAA Performance Climate Changer

Unit Path Configuration	Supply Fan Only Unit
CDQ Wheel	☐
Supply Fan in Unit	✓
Supply Fan Unit Size	3
Supply Fan	
Fan Configuration	HDT
Fan Discharge Section	Horizontal Section
Fan Section Lining	No Lining
CFM	15,000
Diffuser	☐
TSP	2.50
Density Factor	1.00

Discharge Air Component	Ducted Discharge
Air Opening Direction	Side/Bottom/Top
Air Opening Size	75 in.
Outdoor Air Component	Ducted Outdoor
Air Opening Direction	Top/Bottom/Side
Air Opening Size	100 in.
Return Air Component	Ducted Inlet
Air Opening Direction	Top/Bottom/Side
Air Opening Size	75 in.

Fan to Discharge Air Region

Total Lining Length	120.00 in.
	56" No Lining / 64" 2" Perf.
Selected Lining Length	64.00 in.
Selected Lining Type	2" Perf.
Appurt. List	Coil 4 or 6 row / Coil 8 or 10+ row
Appurtenance	Coil
Type	8 or 10+ row

Fan to Return Air Region

Total Lining Length	150.00 in.
	100" 2" Perf. / 50" Solid
Selected Lining Length	50.00 in.
Selected Lining Type	Solid
Appurt. List	Filter Bag / Coil 8 or 10+ row
Appurtenance	Coil
Type	8 or 10+ row
✓ Discharge Silencer	3" Rect. Film
☐ Return Silencer	No Silencer
Multi-Zone	No Multizone

	63	125	250	500	1000	2000	4000	8000
Ducted Discharge Component	61	61	51	49	48	46	45	35
Ducted Inlet Component	49	51	52	51	50	57	58	51
Ducted Outdoor Air Component	63	65	66	65	64	71	72	65
CS Component	103	104	98	92	88	90	87	78

Frequency Band (Hz)

Operating Point	**Supply Fan**
CFM	15000
TSP	2.5
RPM	901
BHP	9.4
XWO	70.8
Efficiency	62.8

Ready Current Job: JOB_9:55 AM_04/13/2010

Figure 25. Single Fan Unit Calculated Full Octave Lw Screenshot