UNIVERSITY OF WITWATERSRAND
JOHANNESBURG

A heuristic tool for indoor radio-wave propagation prediction

Brian Whitaker

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, January 2005

Declaration

I declare that this is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering at the University of the Witwatersrand, Johannesburg. It has not been submitted for any degree or examination at any other university.

Signed this _____ day of _____ 2005

Brian Whitaker

Abstract

For the effective implementation of a wireless local area network (WLAN) within a building, a complete understanding of indoor signal propagation is required. This paper compares three empirical propagation prediction models with regards to efficiency and accuracy. To achieve this, a software prediction tool was developed using C++ which allows an end user to quickly draw a building floor plan using user specified drawing materials. It also has the ability to calculate the required empirical parameters from entered measurements but this was found to produce results similar to that when theoretical empirical parameters were used. The accuracy of the prediction tool was gauged by comparing its outputs, using the different empirical models, to measurements. In doing so it was determined that two of the models produced functional levels of accuracy in which 93% and 82% of the simulated results were within 15 and 10 dB of the measured results respectively for the most accurate of the models used. All three empirical models were found to have computational times low enough, less than 5 minutes for an average building, as to allow for interactive WLAN design.

Acknowledgements

I would like to thank the following people for their much needed contributions to this project:

Prof. Alan Clark, School of Electrical and Information Engineering, Electromagnetics Department, University of the Witwatersrand, Johannesburg. *For his insight and patience during the course of the work.*

Armscor, The Armaments Corporation of South Africa. *For providing the financial support and resources that made this MSc possible.*

My family and loved ones for their support.

Foreword

This dissertation is presented to the University of the Witwatersrand, Johannesburg, South Africa for the degree of Master of Science in Engineering.

Entitled “A heuristic tool for indoor radio-wave propagation prediction”, this dissertation investigates the feasibility of using three well know empirical propagation prediction models for wireless local area network (WLAN) planning. To achieve this, a C++ propagation predication tool was designed and developed with the ability to implement each of the models on user specified building floor plans. Comparisons of the tools outputs with measurements of the actual signal strengths were made and conclusions drawn as to the effectiveness of each model with regards to its efficiency and accuracy.

The format of this document is such that it complies with the university’s paper format, with the main essence of the research contained within the actual paper. The accompanying appendices present supporting information not covered in detail by the main document.

Appendix A contains a detailed description of both the developed tool’s functional and non-functional requirements.

Appendix B presents and discusses the design decisions taken during the development of the tool’s user interfaces.

Appendix C documents the tool’s various classes and their interrelationships.

Appendix D describes in detail each of the tool’s various methods by class.

Appendix E presents some measurement data and describes a sample tool output.

Appendix F contains a compact disc with the tool’s source and compiled program files.

Table of Contents

List of Figures

List of Tables

A heuristic tool for indoor radio-wave propagation prediction

B. H. Whitaker

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: For the effective implementation of a wireless local area network (WLAN) within a building, a complete understanding of indoor signal propagation is required. This paper compares three empirical propagation prediction models with regards to efficiency and accuracy. To achieve this, a software prediction tool was developed using C++ which allows an end user to quickly draw a building floor plan using user specified drawing materials. It also has the ability to calculate the required empirical parameters from entered measurements but this was found to produce results similar to that when theoretical empirical parameters were used. The accuracy of the prediction tool was gauged by comparing its outputs, using the different empirical models, to measurements. In doing so it was determined that two of the models produced functional levels of accuracy in which 93% and 82% of the simulated results were within 15 and 10 dB of the measured results respectively for the most accurate of the models used. All three empirical models were found to have computational times low enough, less than 5 minutes for an average building, as to allow for interactive WLAN design.

Key words: Indoor signal propagation, Empirical prediction models, Interactive planning, Accuracy levels, Computation times

1. INTRODUCTION

Many individuals and corporations have already, or are becoming, heavily reliant on the Internet and public/private data networks for their daily operation. The increased mobility and ease of installation make wireless communication networks the preferred choice in most applications, with increasing demand for wireless local area networks (WLAN's) which when combined with other technologies such as voice over IP (VoIP), wireless CCTV cameras and control systems can supply a complete private/corporate operational package utilising a single network. In most WLAN deployment environments the overall strength and quality of the received signal at any given location is dominated by the many scattering processes due to layout and nature of various obstacles within the deployment environment [1]. Thus, the reliability and cost-effectiveness of WLAN technology relies heavily on the ability to predict the effects obstacles have in the deployment environment and to this end, numerous propagation models and methodologies have been developed, each having their associated advantages and disadvantages.

In this paper three different empirical propagation prediction models are compared with regards to prediction accuracy and efficiency using both theoretical and measured empirical coefficients. Each empirical model was chosen for its straightforwardness and ability to produce accurate results [2] given fairly limited and inaccurate building information and computational resources. For the comparison of the three chosen models a propagation prediction tool, using a unique simulation mechanism, was developed and found to produce good predicted signal strength accuracy levels at high efficiency rates.

The format of the paper is as follows. Section 2 presents and examines literature in the field of indoor radio wave propagation prediction. This is done to illustrate the reasons behind the adoption of the chosen prediction models. The chosen empirical prediction models are then discussed in Section 3 whilst Section 4 describes the developed prediction tool used to implement the adopted prediction models. The measurement techniques used to gauge the accuracy of each selected model along with its resulting accuracy and efficiency are presented in Section 5. Conclusions are included in Section 6.

2. BACKGROUND

Most if not all of the currently available propagation models can be categorised either as being empirical/statistical or deterministic so grouped as they are usually implemented in the same way. In the empirical/statistical approach, on-site measurement data is used to fit either an empirical [2] or statistical [1] propagation model to the building which is then used to determine the signal strength values at unmeasured points of interest. Initially time consuming, since measurement data is required, the resulting models are straight forward to apply, fairly accurate and computationally efficient [2], [1]. Deterministic models, on the other hand, can be divided into both small and large scale subcategories. Small scale deterministic models are generally concerned with the propagation effects of individual objects and rely heavily on the numerically exact solutions to electromagnetic scattering related equations. These can include method of moments (MoM) using both surface and volume integrals [3], [4] as well as various hybrid techniques which combine the method of moments approach with that of geometrical optics (GO) [5]. Of these small scale numerically exact methods, none can be effectively implemented at today's communication frequencies and/or over a meaningful area of interest (with regards to WLAN planning) due to the excessive processing and memory capacity they would require [5]. This is a consequence of the fact that as frequency increases, the size of an object capable of

scattering an electromagnetic wave becomes smaller which results in a drastic increase in complexity. By considering the 802.11a standard which operates at around 5.8 GHz, all in-building obstacles larger than the operating wavelength will need modelling. This is approximately 5 cm and the resulting complexity from having to model obstacles such as coffee cups and telephones make numerically exact solutions impractical.

Ray-tracing techniques form the majority of the large scale, communication frequency, deterministic techniques [6] and often rely on electromagnetic theory such as the uniform theory of diffraction (UTD) [2]. Therefore they require detailed building information pertaining to the layout and electromagnetic properties of its various construction materials and in-building obstacles which is not always readily available [7]. Propagation models based on ray-tracing can predict propagation phenomena such as diffraction and reflection [2] and as a consequence they tend to be the most accurate of all the currently available models, but are computationally complex and result in large computation times which prevent interactive WLAN planning.

In light of the above, it can be seen that by using empirically based prediction models will satisfy the requirements for a propagation prediction tool that is not heavily reliant on the availability and accuracy of building information whilst being simple to use, efficient and accurate.

3. THE EMPIRICAL MODELS USED

By definition, the path loss at a distance d from a transmitter is given by:

$$PathLoss = P_r(d_{\text{ref}}) - P_r(d) \qquad (1)$$

Where d_{ref} is an arbitrary reference distance and P_r(d) is the received power at a distance d from the transmitting antenna. From [8] it is stated that by setting d_{ref} to 1m, only propagation effects are included in equation (1) by normalising the path loss at any distance to that which occurs at d_{ref}, effectively removing any antenna related parameters such as gain and transmitting power. By using equation (1) as a starting point and with the knowledge that it predicts free space path loss, it can be intuitively extended as follows:

$$PathLoss = P_r(d_{\text{ref}}) - P_r(d) - \sum_{i=1}^{m} \text{OAF(i)} \qquad (2)$$

Where OAF and m are the obstacle attenuation factor and number of obstacles between the point of measurement and transmitter respectively. Equation (2) in decibel notation is as follows [2]:

$$PathLoss_B(d) = 10\log_{10}\left(\frac{d}{d_{\text{ref}}}\right)^n + \sum_{i=1}^{m} \text{OAF(i)} \quad (3)$$

henceforth referred to as the Basic log-distance model, the empirical parameters n and OAF are found by best fitting the model to measurement data.

To extend this model, the OAF parameters dependence on the angle of incidence of the incoming electromagnetic wave needs consideration. This stems from the commonly observed phenomenon in which obliquely incident waves transmit less power than those that occur a normal incidences, see Figure 1.

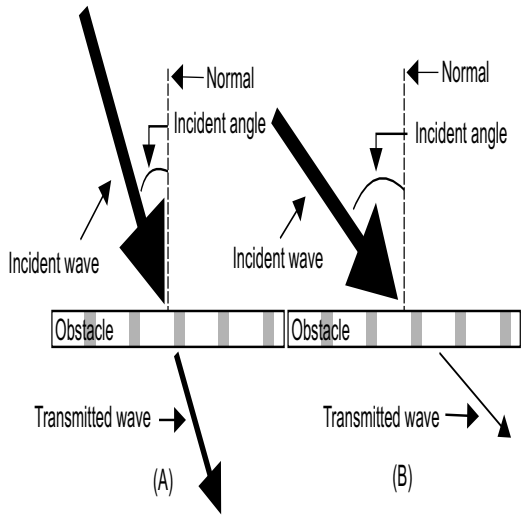

Figure 1 : Obliquely incident waves (B) transmit less power than waves that occur at more normal incidences (A).

The results of the work by [2] established the following OAF dependance on the incident angle with respect to the normal, θ, and forms the basis of the Intermediate log-distance model:

$$PathLoss_I(d) = 10\log_{10}\left(\frac{d}{d_{\text{ref}}}\right)^n + \sum_{i=1}^{m} \frac{\text{OAF(i)}}{cos(\theta)} \quad (4)$$

This was verified by conducting measurements on a piece of metal-tinted glass and results [9] can be seen in Figure 2 .

To further improve the accuracy of the Intermediate log-distance model, the empirical parameter n's dependance on the distance from the transmitting antenna was investigated. From [7] it was found that n as a function of distance has two distinct values, one

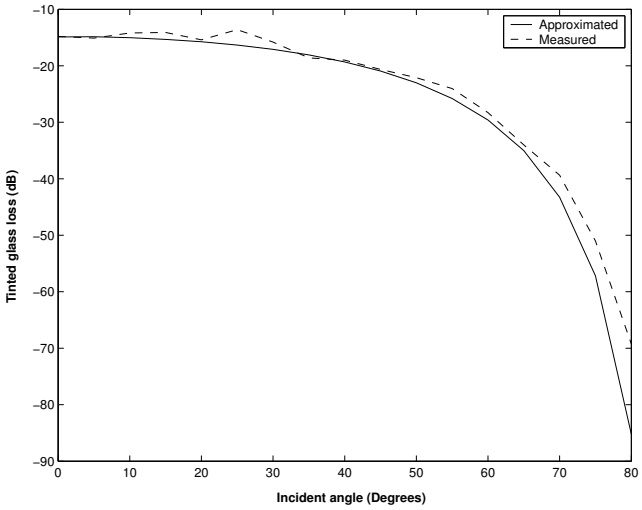

Figure 2 : Tinted glass attenuation as a function of the angle θ. The approximation represented is OAF/cos(θ).

value when within the first Fresnel zone and another once the first Fresnel zone has become obstructed. This can be understood by considering [2] in which it is stated that at distances close to the transmitting antenna (within the first Fresnel zone), obstacles do not significantly effect propagation therefore it can be considered as free-space propagation. In light of this, equation (4) can be rewritten to constitute the Advanced log-distance model as follows:

$$PathLoss_A(d) = 10\log_{10}\left(\frac{d}{d_{\text{ref}}}\right)^{n_{(d)}} + \sum_{i=1}^{m}\frac{\text{OAF(i)}}{cos(\theta)} \tag{5}$$

Where:

$n_{(d)}$ = 2 if within the first Fresnel zone.
$n_{(d)}$ = 2-5 if first Fresnel zone is obstructed.

The distance Z_f at which the first Fresnel zone becomes obstructed is given by the following equation [7]:

$$Z_f \approx \sqrt{\lambda L} \tag{6}$$

Where the parameter L is the maximum diameter, in meters, of an unobstructed ellipse drawn in the vertical plane of a building floor as shown in Figure 3. Thus for any given building, the diameter of the first fresnel zone can be anywhere from 5-25 m depending on the height of the ceiling above the floor as well as the nature, layout and density of the various obstacles within the building environment. The value of $n_{(d)}$, once the first Fresnel zone has become obstructed, usually varies between 2 and 5 and can be found by best fitting on-site measurement data to equation 5.

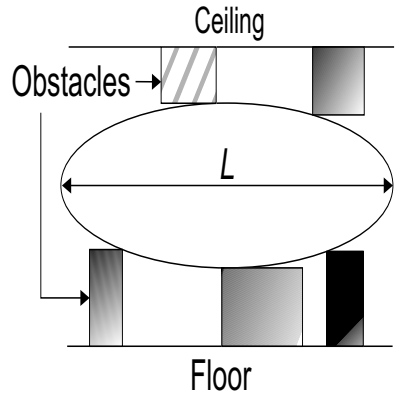

Figure 3 : Graphical illustration of the L parameter used to determine the diameter of the first Fresnel zone.

4. THE SIMULATION MECHANISM

To implement the empirical models presented in Section 3 a software application was developed using C++, chosen due to the speed at which it is able to carry out repetitive algorithms. Listed below is some of the softwares functionality. For a full listing, please see [10].

- Is backed by an Access® materials database that allows for the editing, addition and deletion of user specified materials.
- Allows the end-user to draw a building plan as well as position various in-building obstacles using either a CAD like or mouse pointer driven drawing interface [11].
- A simulation settings dialog that allows for the specification of the required simulation constants as well as the various empirical parameters outlined in Section 3 [11].
- A drawing settings dialog that allows for the specification of the required drawing area size [11]. This automatically scales the representative pixel size.
- Has two different simulation mechanisms. The first (off-site) implements the propagation models using user defined empirical parameters whilst the second (on-site) calculates the empirical parameters from user-entered measurement results on the drawing plan.

A raster as opposed to a vector approach was used to ensure the lowest possible simulation computation times. This stemmed from the fact that intersection testing, using vector graphics, requires the solution of numerous modelling equations for objects that might not even be in the direction in question. In the raster methodology used, intersection testing was simplified by representing the required building floor plans, in-building obstacles and the transmitting antenna as corresponding values in a matrix. Agents (the data

structures responsible for the propagation prediction) then move through the matrix in all directions, having originated at the location of the transmitting antenna, setting the corresponding matrix entries to the required signal strength values. To move the Agents, Bresenhams line drawing algorithm was used as it only requires integer addition, subtraction and multiplication by 2 which computers can perform efficiently [12]. On each move, an Agent is faced with the decisions outlined in Figure 4.

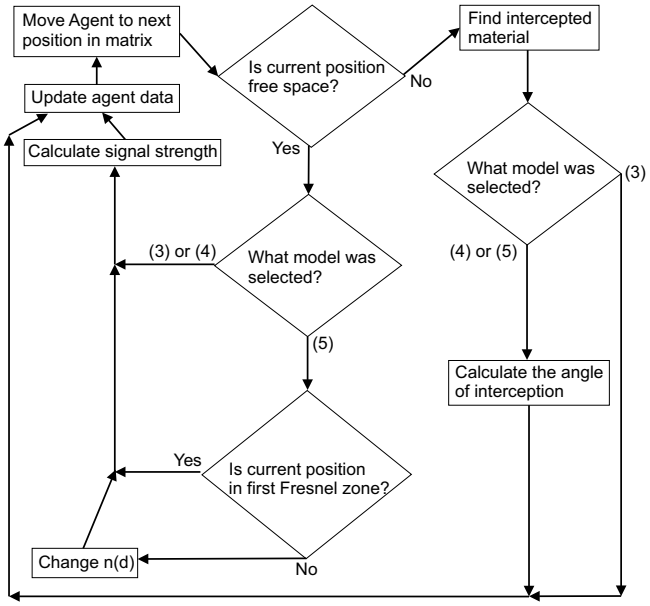

Figure 4 : The agent decision tree.

To improve the efficiency of the simulation mechanism, agents are preprogrammed to stop functioning at a user specified signal strength level or when they leave a user specified simulation boundary. Furthermore, a weighted averaging algorithm is also employed to calculate the signal strength at regions where no agent activity was present.

5. MEASUREMENT TECHNIQUE AND RESULTS

Propagation predictions were made using the software for two different buildings using models (3), (4) and (5). The buildings in question were dissimilar in the fact that the one was primarily constructed from bricks and mortar whilst the other from reinforced concrete and dry walling. From visual inspection, rough sketches of the buildings were entered into the application. For the off-site simulation mechanism the required empirical parameters were determined as follows:

- For models (3) and (4), n was set at 2 (free space approximation).
- For model (5), $n_{(d)}$ within the first Fresnel zone was set at 2 and 2.5 once the first Fresnel zone became obstructed [2]. Z_L was found using equation (6).
- The OAF's used in all the models were found from [13] and off-site measurements.

This process was dissimilar to that used in the determination of the empirical parameters for the on-site simulation mechanism. For models (3) and (4) n was set to 2 whilst for model (5) $n_{(d)}$ was set to 2 when within the first Fresnel zone and 2.35 once the first Fresnel zone became obstructed. These values were arrived at by best fitting on-site measurement data to the respective model equation. Z_L was found using equation (6) as in the off-site mechanism and the OAF's for the various building materials were found by the application using user entered measurement data on the building floor plan.
For the actual measurements, a client card, attached to a laptop, was used in combination with an access point (AP). The client card had a maximum gain of 0 dBi and due to the non-uniformity of its radiation pattern in the plane of interest, the orientation of the laptop (with respect to the AP) whilst taking measurements was kept constant. The accuracy of the measurement system was gauged by comparing its reported path loss over a set distance in an anechoic chamber, to that of a known system. In doing so it was determined that the measurement system was fairly accurate with its reported path loss values being within 2 dB of the known values. For the comparison between the model predictions and measurements, more than 200 measurements of the actual signal strength were taken at random positions within the two buildings. Fast fading effects were initially considered by taking the average of a selection of measurements within a 20λ area around the selected random points [8], but was found to produce a similar results to a single measurement in most to all instances. Measurements were also taken at a distance to prevent the measurer from affecting their integrity.

5.1 Application and model efficiencies

By recording the simulation times for various agent counts and floor plan areas in square pixels (AISP), the curve in *Figure 5* showing the relationship was sketched. In doing so it was also determined that the approximate run-times of the different models were similar and thus can be represented by the same set of curves. This is a result of the fact that free space agent movement calculations greatly dominated the overall simulation run-time in the tested building environments. Also shown in Figure 5 is that at lower agent counts (less than 50 per octant), the weighted average algorithm dominates the overall simulation run-time i.e. lower agent counts result in less calculated signal strength values and hence more averaging is required is achieve an approximately continuous signal strength distribution. As agent counts rise, the time taken to move them around the raster increases and coupled with the fact that they produce more calculated signal strength values causes their movement algorithm

to dominate over the overall simulation run-time.

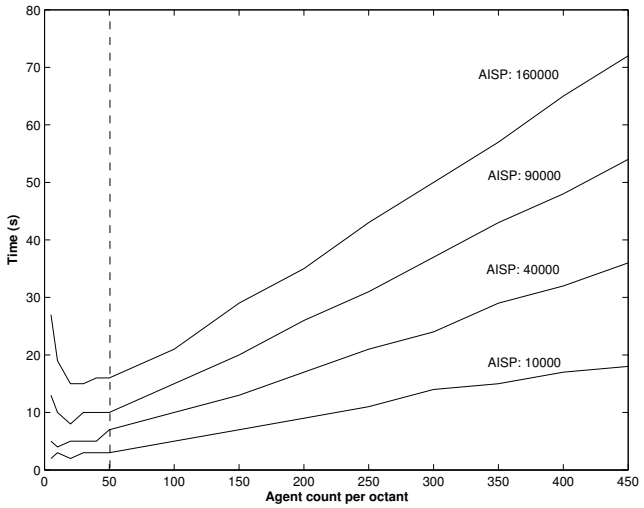

Figure 5 : Simulation time vs. agent count for various AISP's.

Tests conducted on a 600m^2 section of building produced average run-times of approximately 3 minutes using a P4 2.8 GHz personal computer with 1 GB of memory, illustrating the efficiency of the empirical models used.

5.2 Application and model accuracies

By comparing the obtained measurement results to that of the simulated results for the different empirical models, the results in *Table 1* were obtained.

Table 1 : A comparison of the coinciding simulated and measured points with error less than 5, 10 and 15 dB for the three empirical models used.

	<5 dB	**<10 dB**	**<15 dB**
Basic log-distance (3)	41%	61%	76%
Intermediate log-distance (4)	47%	77%	89%
Advanced log-distance (5)	51%	82%	93%

These results were similar to those found by [2] for the Advance log-distance model. It was also found that the choice of using either the on-site or off-site simulation mechanism, did not significantly alter the general accuracy of the model used. However, under the off-site simulation mechanism a larger standard deviation in the results with error greater than 15 dB was observed. The reason for this is that the objects with high attenuation factors such as steel cabinets, under the off-site mechanism, exaggerate their effects on the signal strength behind them as none of the models account for diffraction and reflection. On the other hand, the on-site simulation mechanism determines an object's attenuation factor from measurements which are affected by indirect signal paths. In doing so, the effects of high attenuation factor objects are underplayed, resulting in more consistent results. The accuracy of the models as a function of agent count was considered and it was determined that agent counts above 200 per octant produced consistently accurate results. This is a consequence of the fact that at lower agent counts, obstacles far away from the transmitting antenna are often not intersected as agents are programmed to move radially outwards from the transmitting antennas location.

The effects of diffraction and reflection were not included in any of the models to eliminate the need for vector additions which require a high degree of accuracy and precision with regards to electromagnetic properties of the building materials used and the building dimensions respectively. Both of which are time consuming and hence expensive to obtain. As an example, the work done by [13] at 914 MHz, required building blueprint levels of precision to achieve accurate results.

In light of this, the results in Table 1 show that even with fairly inaccurate building floor plans (maximum error in each dimension no greater than 0.2 m) usable levels of accuracy were obtained for both the Intermediate and Advanced log-distance models at interactive computational times.

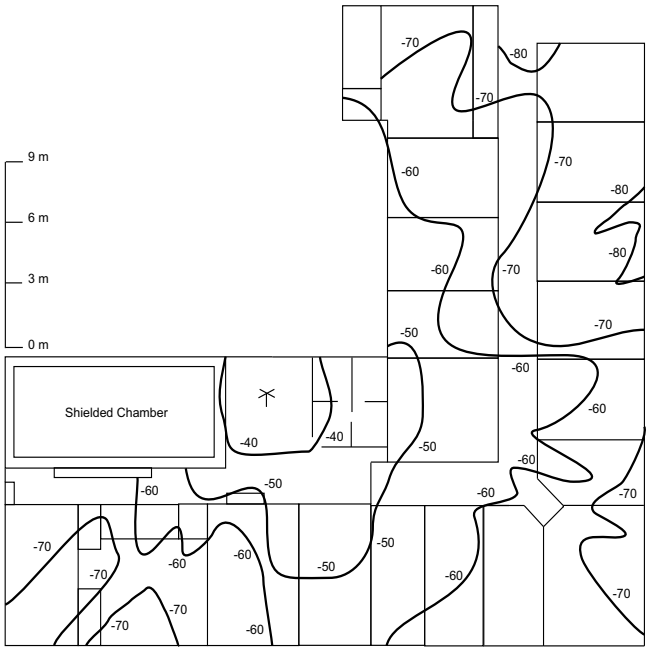

Figure 6 : Simulated signal strength distribution for a single floor section of a building in dB.

This assumption was made as it was found that the vast majority of the simulated results were within 15 dB of the actual signal strengths. Further more, it is backed up by the fact that WLAN's should never be designed down to the sensitivity levels of the receiving antennas. A margin of at least 20 dB should be

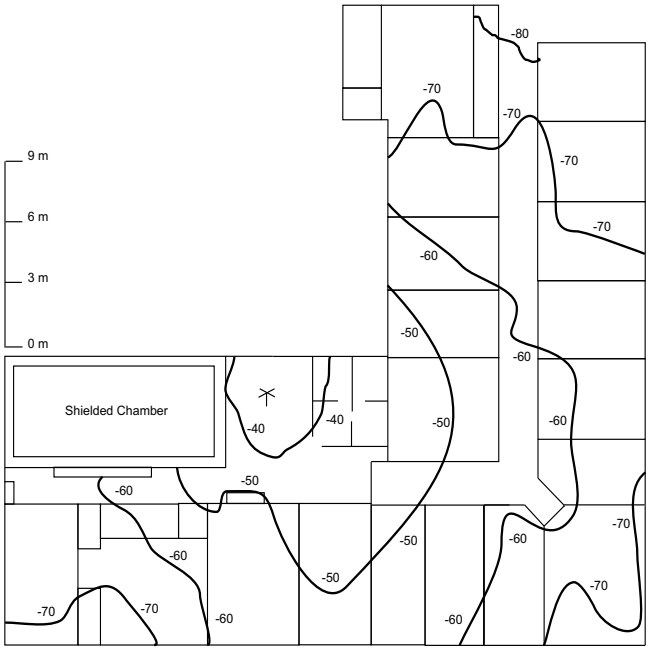

Figure 7 : Measured signal strength distribution for a single floor section of a building in dB.

ensured due to the highly dynamic nature of the resulting channel between the transmitter and receiver. A example of this is found in [14] who found that losses from 6 to 11 dB can result from human body-shadowing effects. Figures 6 and 7 show the similarity in contour plots between the simulated and measured signal strength distributions (using the Advanced log-distance model and the off-site simulation mechanism) for one of the measured buildings. The contour lines for the simulated results were generated by the software and Matlab® was used to obtain contour lines from the measurement data. Of interest in Figure 6 are the regions of rapid signal strength decline caused by the extremely oblique intersection angles with obstacles between the transmitting antenna and region in question. This is not reflected in Figure 7 and thus constitutes a software anomaly that does not drastically alter the softwares overall accuracy.

6. CONCLUSION

By developing an indoor propagation prediction tool for WLAN's, the accuracy and efficiency of three different empirical propagation prediction models were tested. This was done by comparing the tool outputs to measurements and it was found that two of the models produced usable levels of accuracy, as the vast majority of their predicted results were within 15 dB of the actual measured signal strength values. All three of the empirical models were found to be computationally non-time consuming and thus could be used in interactive WLAN planning or in conjunction with access point optimisation algorithms that would require a vast number of simulations to be run. In all of the conducted simulations, the use of fairly inaccurate building floor plans were found to not significantly affect model accuracies.

REFERENCES

[1] T. S. Rappaport. "Characterization of UHF Multipath Radio Channels in Factory Buildings." *IEEE Trans. Antennas Propagat.*, vol. 37, no. 8, pp. 1058–1069, 1989.

[2] K. Cheung, J. H. Sau, and R. D. Murch. "A New Empirical Model for Indoor Propagation Prediction." *IEEE Trans. Veh Technol.*, vol. 47, no. 1, pp. 996–1001, 1998.

[3] S. Govind, D. R. Wilton, and A. W. Glisson. "Scattering from inhomogeneous penetrable bodies of revolution." *IEEE Trans. Antennas Propagat.*, vol. 32, pp. 1163–1173, 1992.

[4] J. H. Richmond. "Scattering by a dielectric-cylinder of arbitrary cross section shape." *IEEE Trans. Antennas Propagat.*, vol. 13, pp. 334–341, May 1965.

[5] R. Bhalla and H. Ling. "Three-dimensional scattering centre extraction using the shooting and bouncing ray technique." *IEEE Trans. Antennas Propagat.*, vol. 44, no. 11, pp. 1445–1453, 1996.

[6] C. Yang, B. Wu, and C. Ko. "A Ray-Tracing Method for Modeling Indoor Wave Propagation and Penetration." *IEEE Trans. Antennas Propagat.*, vol. 46, no. 6, pp. 907–919, 1998.

[7] W. Honcharenko, H. L. Bertoni, J. L. Dailing, J.Qian, and H. D. Yee. "Mechanisms governing UHF propagation on single floors in modern office buildings." *IEEE Trans. Veh Technol.*, vol. 41, no. 4, pp. 496–504, 1992.

[8] D. M. J. Devasirvatham. "A comparison of time delay spread and signal level measurements within two dissimilar office buildings." *IEEE Trans. Antennas Propagat.*, vol. 35, no. 3, pp. 319–324, 1987.

[9] B. H. Whitaker. *A heuristic tool for indoor radio-wave propagation prediction*. Master's thesis, Appendix E: Measurement data and example output, University of the Witwatersrand, 2005.

[10] B. H. Whitaker. *A heuristic tool for indoor radio-wave propagation prediction*. Master's thesis, Appendix A: Application Attributes, University of the Witwatersrand, 2005.

[11] B. H. Whitaker. *A heuristic tool for indoor radio-wave propagation prediction*. Master's thesis, Appendix B: User interface design, University of the Witwatersrand, 2005.

[12] J. E. Bresenham. "Algorithm for computer control of a digital plotter." *IBM Syst. Jour.*, vol. 4, no. 1, pp. 25–30, 1965.

[13] S. Y. Seidel and T. S. Rappaport. "914 MHz Path Loss Prediction Models for Indoor Wireless Communications in Multifloored Buildings." *IEEE Trans. Antennas Propagat.*, vol. 40, no. 2, pp. 207–217, 1992.

[14] S. Obayashi and J. Zander. "A Body-Shadowing Model for Indoor Radio Communication Environments." *IEEE Trans. Antennas Propagat.*, vol. 46, no. 6, pp. 920–926, 1998.

Appendix A:
Application Attributes

1. Introduction

The following document describes the developed WLAN propagation prediction tool by presenting both its functional and non-functional related attributes. In viewing this document it is important to remember that during the development of the tool, proof of concept was a priority and the need to produce a fully functional tool less important.

2. Functional Attributes

In order for proof of concept, the tool has the following functionality:

1. Allows the end user to draw a building plan and any in-building obstacles such as book shelves, filing cabinets etc. The drawing can be done using either a text based drawing interface in which the commands *MoveTo(x,y)* or *LineTo(x,y)* are used where *x* and *y* are in meters or a mouse pointer driven interface.
2. Is backed by an Access® materials data that allows for the editing, addition and deleting of user specified materials. Each material in the materials database is given a name, frequency of operation (as material attenuation is dependent on frequency), transmission coefficient and a unique colour.
3. Allows the end user to control all aspects of the simulation. This is achieved by having a simulation settings dialog that is used to specify the required simulation mechanism, frequency, cut-off level, initial agent count, first and post Fresnel Zone exponents, diameter of first Fresnel Zone and the measured signal strength at a distance of one meter from the transmitting antenna.
4. A drawing settings dialog that allows the end user to specify the required drawing area size. This is done to ensure the maximum level of accuracy is achieved in a simulation, as a raster approach was used in which each pixel represents a physical area of the floor plan. The allowable drawing size range is from 100 m^2 to 25 000000 m^2.
5. Has the ability to zoom in and out to aid in the drawing of a building plan.
6. Allows simulation results to be saved to a Device Independent Bitmap or DIB to allow for easy distribution.
7. Allows work in progress to be fully saved in a .HIP file which is the applications unique file format.
8. Allows the specification of a simulation boundary to reduce the simulation times.
9. Has two different simulation mechanisms. The first (off-site) implements the propagation models using user-defined empirical parameters whilst the second (on-site) calculates the empirical parameters from user-entered measurement results on the drawing plan.

3. Non-functional Attributes.

The main non-functional attributes of the developed tool are that of reliability, maintainability, upgradeability and compatibility. Descriptions and means of achievement for these attributes can be found in Table 2.

Table 2: The non-functional attributes of the developed tool.

Non-functional requirement	Description of requirement	Why it is needed	How it was achieved
Reliability	The tool must perform to the same degree of accuracy and efficiency each and every tine it is used.	A reliable tool will increase user confidence in the results it produces.	Sound programming techniques and extensive testing procedures.
Maintainability	It must be possible and easy to repair any errors/bugs that arise in the application during testing and in its use.	A maintainable tool will continue to be applicable in the event of finding bugs/errors.	By adopting an object orientated programming technique, errors and sources of inaccuracy are easily found and eliminated.
Compatibility	The tool must run on most personal and laptop computers running a Microsoft® operating system.	A compatible tool can be used by a larger and more diverse user group.	By designing the program such that it does not require large amounts of system resources and by ensuring that no platform depending technology was used.
Upgradeability	It must be possible and easy to add new functionality to the application.	An upgradeable tool will be applicable over a larger time span.	By adopting an object orientated programming technique, additional functionality can be easily added.

4. A Basic Operational Scenario

On running the application, the vast majority of its end user will do the following:

1) Browse the well established materials database to ensure that the required building materials are present. If not, a new material will be entered and specified.

2) Using either the text or mouse based drawing facilities, draw the required building plan.
3) Place an antenna on the building plan at the desired location and specify a simulation boundary.
4) Save the building plan to a .HIP file.
5) Run the simulation using either the Basic, Intermediate or Advanced Log Distance simulation mechanism.
6) View the simulation results and repeat steps 3 – 5 until the desired signal strength distribution is reached.
7) Save the simulation results to a DIB.
8) Print a copy and send it to a WLAN installer to indicate the positions of the desired Access Points.

5. Conclusion

From the discussion of the developed applications functional and non-functional attributes it is evident that the only limitation to the proof of concept is whether the chosen empirical propagation prediction models have the required accuracy and efficiency. This results from the fact that the developed application attributes allow the prediction models to be fully and unlimitedly implemented.

Appendix B:
User Interface Design

1. Introduction

A survey carried out in 1992 found that approximately 48% of an applications code was devoted to the user interface and that 50% of the development time was spend implementing the user interface [1]. This is a result of the fact that well designed interfaces have become extremely important in ensuring the overall success of any given software application.

In light of this, the following document presents the design decisions taken in the construction of the various user interfaces for the developed WLAN propagation prediction tool. The user interfaces are presented and their functionality discussed. Conclusions on the degree to which the developed interfaces meet both their functional and non-functional requirements are also given.

2. User Interface Requirements

A good user interface is one in which the balance between user-friendliness and transparent functionality is such that in affords its targeted users a high degree of productivity. As the developed software is targeted at a semi-technical to technical end user, the following user interface requirements were important:

- Be user friendly to the extent that it is easy to use, but must not limit advanced users.
- Promote learning and be consistent.
- Provide all of the applications required functionality.

3. User Interfaces Developed

Using the above outlined requirements, the applications interfaces were designed and developed. On running the application, the end user initially sees the main application interface. The main interface was designed in such a way that all the required functionality to draw a building floor plan is easily available. This can be seen in Figures 8 and 9 which are the left and right hand side of the main interface respectively. The key for Figure 8 is as follows:

A. The applications drawing rulers. They are auto scaling rulers that change depending on the user specified drawing size, see Figure 12. Ruler units are in meters.
B. A simple tool bar that allows easy access to the New, Open, Save, About and Print drawing functions found in the File drop down menu option.
C. Section of the tool bar used to provide feedback during a simulation. Possible feedback includes *Generating agents*, *Moving agents around raster* and *Averaging in progress*.
D. Menu bar containing the applications various drop down menus, see Table 3 for a complete description.
E. Applications drawing area. Having selected a material, a line is drawn by depressing the right mouse button at the start of the line and releasing it at its end.
F. Horizontal scroll bar used to bring unseen areas of a large drawing into view.

G. Materials list box containing the ID and name of all the materials entered in the materials database. A material is selected by single left-clicking it's ID.
H. Antennas list box containing the ID and name of all the antennas in the antennas database.
I. Drawing input bar which responds to the commands *MoveTo(x,y)* and *LineTo(x,y)* where *x* and *y* are in meters.
J. Status bar used to provide feed back on the state of the application.

And the key for Figure 9 is as follows:

A. The current process indicator bar.
B. Vertical scroll bar used to bring unseen areas of large drawing into view.
C. Command history list box which contains the carried out drawing commands. Commands and their resulting action are deleted by double left-clicking the command number.
D. The current length in meters of a line being drawn.
E. The currently selected drawing material.
F. The current x-position in meters of the mouse curser.
G. The current y-position in meters of the mouse curser.

Table 3 below describes the functionality of the applications menu options:

Table 3: The functionality of the applications menu options.

Menu	Menu option	Function
File	New	Clears existing work and opens a new .HIP file.
	Open	Opens an existing .HIP file.
	Save	Saves the open building plan .HIP file.
	Save As	Saves a new building plan .HIP file
	Print	Prints the currently open building plan .HIP file.
	Print Preview	Opens a standard Print Preview interface showing the open building plan .HIP file.
	Print Setup	Opens a standard Print Setup interface.
	Recent File List	Shows the last five worked-on building plan .HIP files.
	Exit	Exits the application.
Zoom	Zoom In	Zooms in on the current building plan by a factor of 2.
	Zoom Out	Zooms out on the current building plan by a factor of 2.
	Zoom Normal	Returns the zoom factor to unity.
View	Toolbar	Shows/Hides the Toolbar.
	Status Bar	Shows/Hides the Status Bar.
	Input Bar	Shows/Hides the Input Bar.
	Display Bar	Shows/Hides the Display Bar.
Databases	Materials	Displays the Materials interface, see Figure 10.
Settings	Simulation Settings	Displays the Simulation Settings interface, see Figure 13.
	Drawing Settings	Displays the Drawing Settings interface, see Figure 12.
Simulate	Calculate Coefficients	Calculates material attenuation factors from entered measurement results
	Basic Log Distance	Initiates the Basic Log Distance simulation mechanism.
	Intermediate Log Distance	Initiates the Intermediate Log Distance simulation mechanism.
	Advanced Log Distance	Initiates the Advanced Log Distance simulation mechanism.
	Save Results	Saves the simulation result in a standard, easily distributable, DIB.
Help	About HIP	Displays a standard About dialog box.

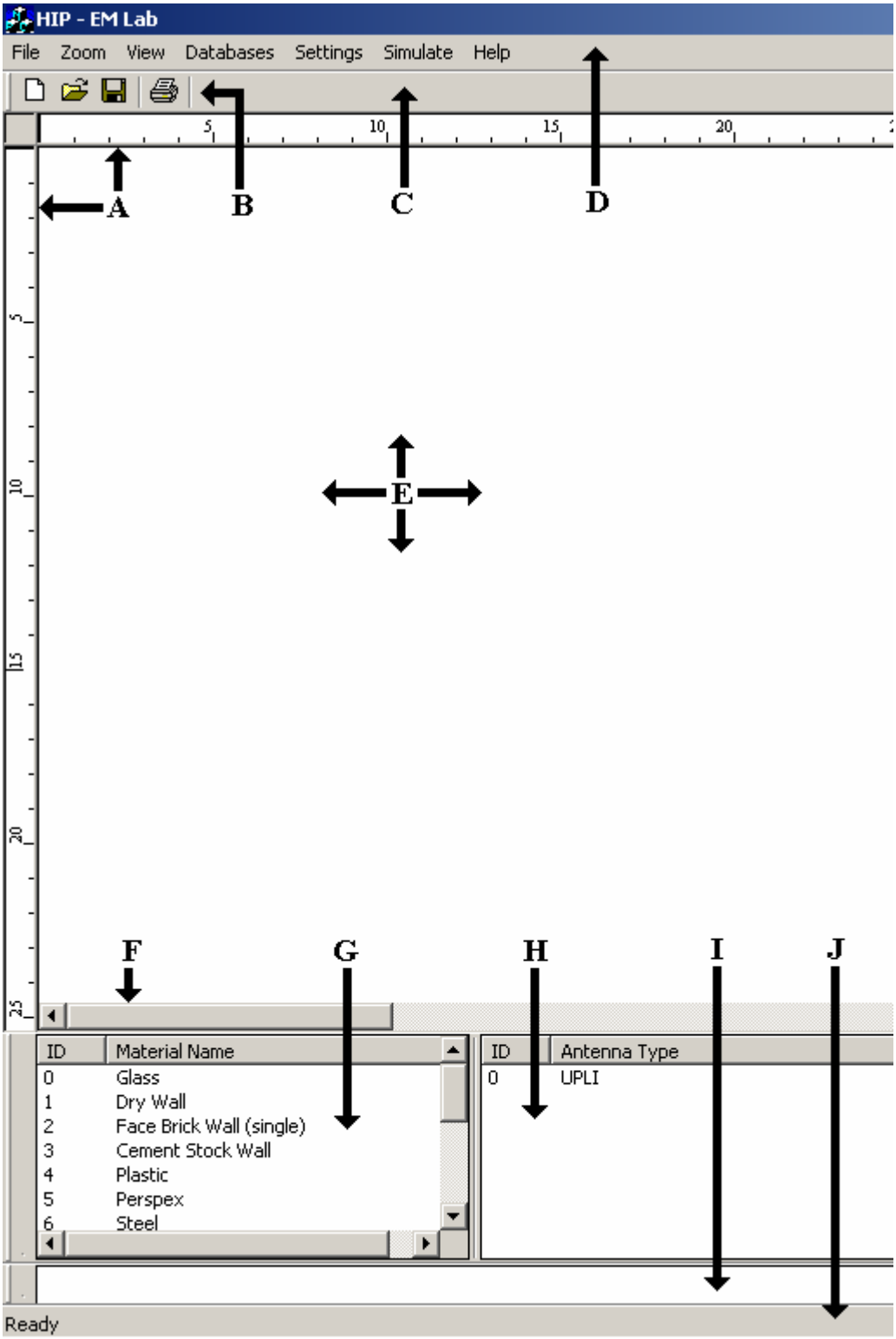

Figure 8: Left hand side of the applications main user interface.

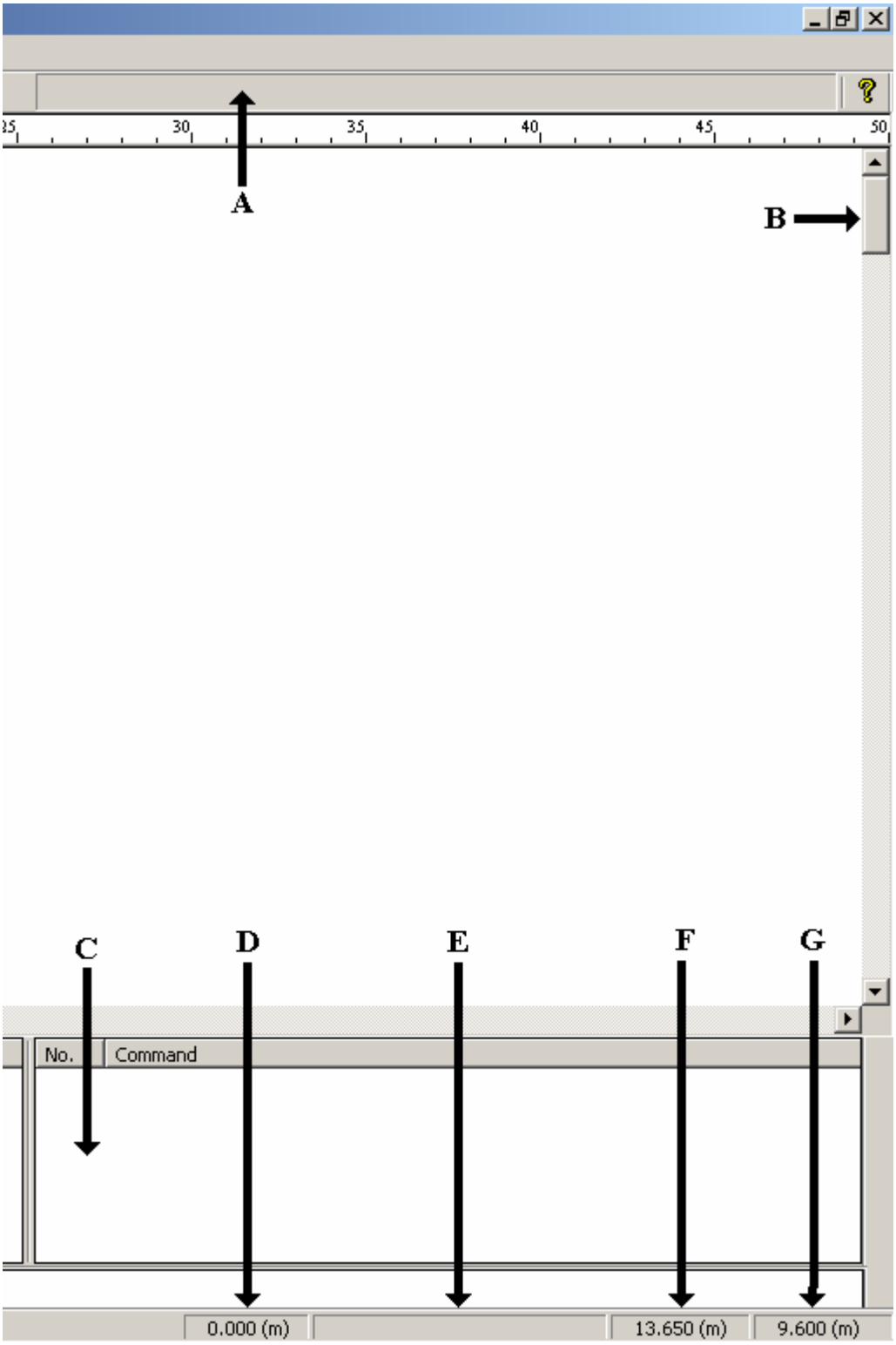

Figure 9: Right hand side of the applications main user interface.

By selecting the Materials menu option from the Databases menu, the interface presented in Figure 10 is displayed. This interface is used to maintain the applications material database.

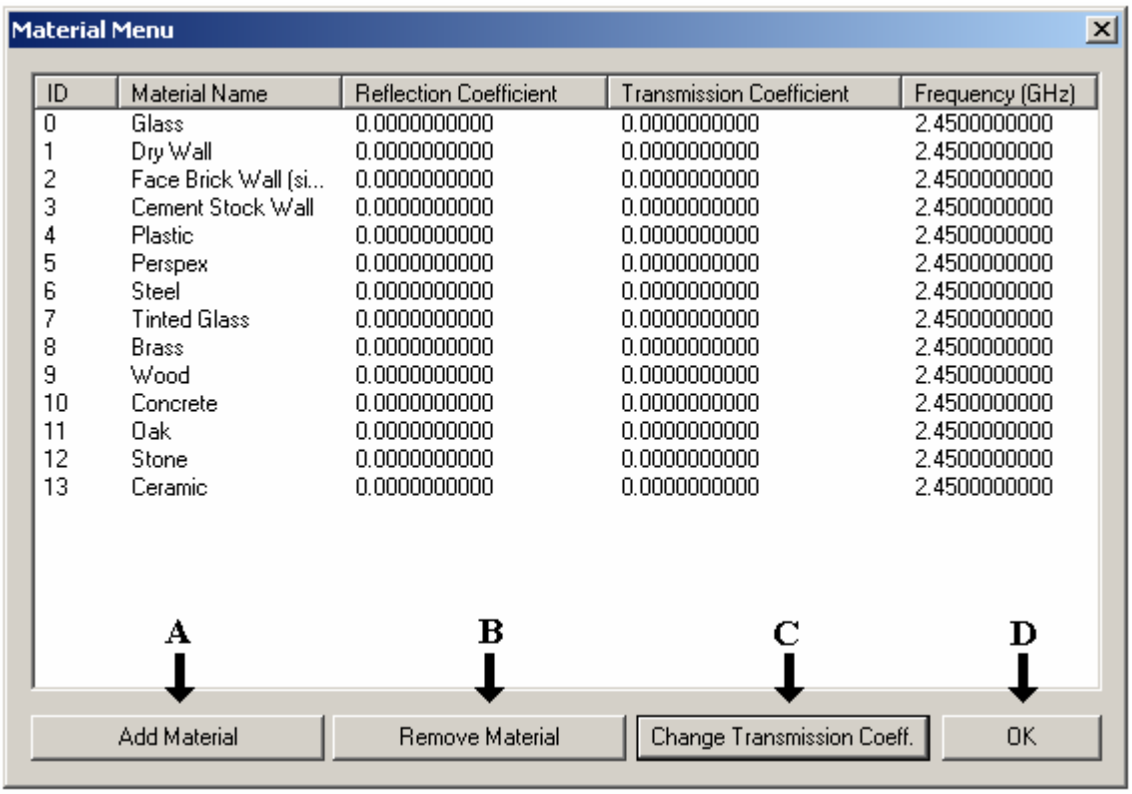

Figure 10: The Material Menu interface.

A. Add Material button. This displays the Add Material interface, see Figure 11, used to add a new material to the materials database.
B. This button removes a material from the materials database. The material to remove is selected by single left-clicking it's ID.
C. This button displays a dialog box which prompts the user to enter the selected materials new transmission coefficient. The material is selected by single left-clicking it's ID.
D. This button closes the interface.

To add new material, the Add Material interface is used, Figure 11. In this interface, the new material name, transmission coefficient, representative colour and frequency of operation is selected.

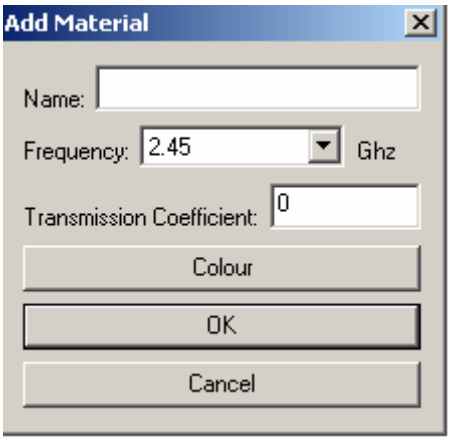

Figure 11: The Add Material interface.

To setup both the drawing and simulation environments, the Drawing Settings and Simulation Settings interfaces are used. These can be seen in Figures 12 and 13 respectively.

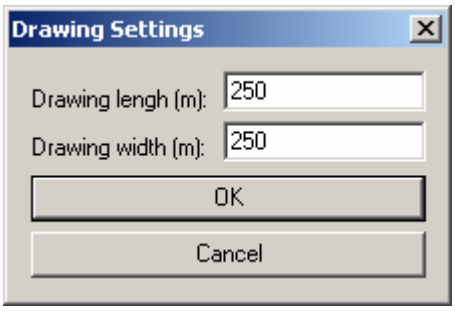

Figure 12: The Drawing Settings interface.

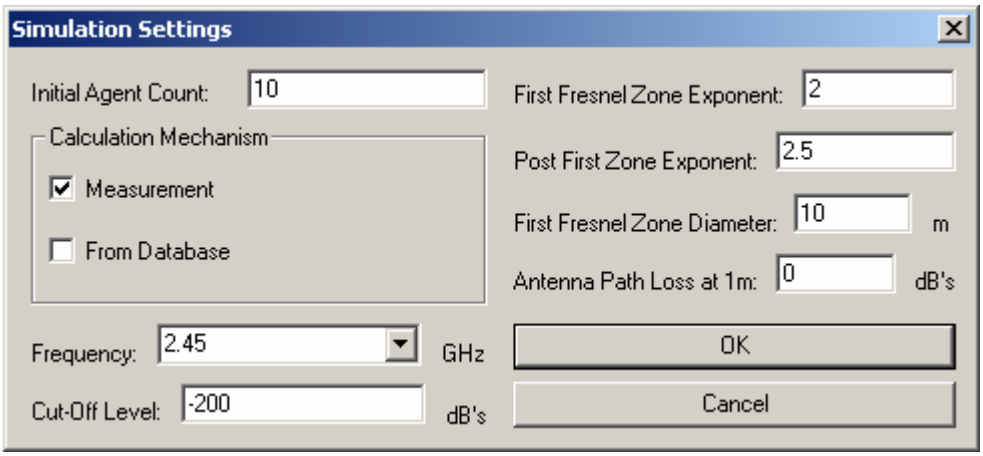

Figure 13: The Simulation Setting interface.

The Drawing Settings interface in used to specify the required drawing size. This is done to maximize the accuracy of the simulated results, by ensuring that the specified drawing size is no bigger than the actual drawn building. The Simulation Settings interface is used to specify the required simulation constants. These include the initial agent count, first and post Fresnel Zone exponents, diameter of the first Fresnel Zone, simulation frequency and cut-off level as well as the antenna lost at one meter. The simulation mechanism can also be set to use either object attenuation factors derived from the database of materials or user entered measurement. User measurements and antennas are entered onto a building plan by double right-clicking the floor plan at the desired position. This displays the Select Item interface, Figure 14, in which either an antenna or specified measurement result is selected.

Having successfully run a simulation, the predicted signal strength will be represented by means of a colour gradient on the actual floor plan. The colour gradient ranges from red (areas of high signal strength) to blue (areas of low signal strength). A more accurate signal strength value can be obtained by pressing Shift and single right-clicking at the desired position. By pressing Shift and single left-clicking at two different positions, a simulation cut-off boundary can also be specified if the user does not want to run the simulation over the entire building plan. This can often save considerable amounts of time.

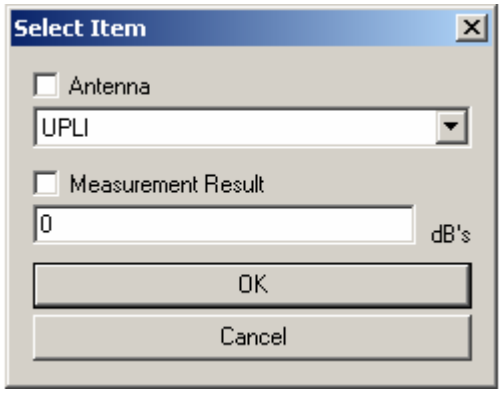

Figure 14: The Select Item interface.

4. Evaluation of the user interfaces developed

The developed interfaces were found to have the following attributes:

- **Provide positive feedback to the user** – The interfaces constantly keep the end user informed of any ongoing processes and the initiation of new ones. This is accomplished by using dynamic buttons which move when pushed and on-screen displays which are constantly updated. Immediate feedback is important to preserve the users' trust in and satisfaction with the application whilst ensuring that the user stays attentive to the process he/she is involved in.

- **Provide good error messages** – From a users point of view an error message is useless if it is not in an understandable format. With this in mind the interfaces report errors in generally used language, containing no jargon, and suggest possible solutions to the errors that occur.

- **Are consistent –** Research has shown [1] that consistency with regards to related functions and general procedures within a software application reduces the time taken for an end user to become familiar with the application thus increasing their confidence. Therefore, all the buttons in the interface are the same shape and colour, and related menu items are grouped together.

- **Use colour advantageously** – The use of colour by all means can both add and detract from the overall quality of an interface. In order to gain the beneficial use of colour within the application, the following was adhered to:

    1. Colour was not used to convey a meaning as different colours are perceived to mean different things by different users.
    2. Colours that clash were not used as this distracts the end user and reduce his/her productivity.

- **Are transparent** – The general presentation of information within the application is concise and contains no irrelevant information. System flow with regards to accomplishing a given task within the application is

intuitive thus allowing users of varying computer skill levels to utilize the application accurately and efficiently.

4. Conclusion

From the introduction of this report, it became apparent of the importance quality interfaces play in the success of an application. It was found that a good quality user interface will increase user efficiency and productivity whilst reducing user error and time spent learning how to use the application. This will undoubtedly increase user acceptance, resulting in a more widely used application. The developed interfaces for a WLAN propagation prediction tool were presented and it was determined that they meet both their functional and non-functional requirements with regards the good interface design.

5. Reference List

[1] Van Vliet H, “Software engineering: Principles and Practice” Second edition, John Wiley & Sons, New York, 2002.

Appendix C:
Application Class Overview

1. Introduction

Contained within this document is a listing and description of the developed WLAN propagation prediction tool's main classes. A class diagram showing the relationship between the various classes is also given. From the presented class diagram, it can be seen that good object orientated programming techniques were carried out throughout the tools development phase resulting in a robust, maintainable and fully upgradeable application.

2. Class Descriptions

The following main classes were used to implement the WLAN propagation prediction tool:

1. **AddMaterialDlg –** A descendent of the CDialog class, this class is responsible for the displaying and functionality of the Add Material interface. In this interface, the materials name, frequency of operation, transmission coefficient and colour is specified. The classes function is to take in the user-entered material specifications, validate them and then add them to the materials database.
2. **CAgent –** A descendent of the CObject class, this class encapsulates the entire attributes of the utilized agent data types. These data types move around the raster containing the user drawn building plan and determine the resulting signal strength distribution. To achieve this, they move a pixel at a time, check for an intersection and using the established simulation parameters determine the resulting signal at their current point using the specified simulation mechanism.
3. **CAntenna –** A descendent of the CObject class, this class encapsulates a specified antenna attributes. These attributes include radiation pattern and gain characteristics.
4. **CAntennaSet –** A descendent of the CRecordSet class, this class provides the functionality to read and write to the antennas database.
5. **CCommand –** A descendent of the CObject class, this class encapsulates the attributes of a command object. Command objects are generated each time a line is drawn and are used loading, saving and manipulation building plan drawings.
6. **ChangeTransCoeffDlg –** A descendent of the CDialog class, this class loads and controls the Change Transmission Coefficient interface. This interface is responsible for changing an entered materials transmission coefficient and simple takes in the materials new transmission coefficient, validates it and updates the materials database.
7. **CHIPApp –** A core application class, his class retrieves all the applications event messages and passes them on to the CHIPView class.
8. **CHIPDoc –** A core application class, this class is responsible for housing and manipulation the applications various data structures. These include all the objects derived from the CObject class. It also receives inputs from the CHIPView class, processors them and passes display information back to the CHIPView class.
9. **CHIPView –** A core application class, this class displays information to the screen and contains the majority of the functionality that acts on the data structures contained within the CHIPDoc class.
10. **Cline –** A descendent of the CObject class, this class encapsulates the attributes of a line object. A line object specifies a drawn line and hence contains the lines start and end points along with the material used to draw it.

11. **CMainFrame –** A core application class, this class holds the applications various visible objects such as its menus, scrollbars and toolbars.
12. **CMaterialSet -** A descendent of the CRecordSet class, this class provides the functionality to read and write to the materials database.
13. **CMeasurementResult –** A descendent of the CObject class, this class encapsulates the attributes of a measurement object. A measurement object is generated each time a user specified measurement is entered on a building floor plan and stores the measurements value and position.
14. **CPointResult –** A descendent of the CObject class, this class encapsulates the attributes of a point result object. Point result objects are generated by the agent objects after them have moved from one pixel to another. They contain the calculated signal strength by the agent as well as the position in raster at which the agent did the calculation.
15. **DrawingSettingsDLG –** A descendent of the CDialog class, this class displays the Drawing Settings interface and takes in the user-entered drawing size.
16. **MaterialDlg –** A descendent of the CDialog class, this class displays the Materials Database interface. In the interface, the contents of the material database is displayed and the functionality.
17. **PlaceExtraDlg –** A descendent of the CDialog class, this class allow the end user to place either an antenna or measurement result on the drawn building floor plan.
18. **SimulationSettingDlg –** A descendent of the CDialog class, this class displays the Simulation Settings interface, takes in the user specified simulation parameters and validates them.

3. Class Diagram

A simplified class diagram showing the relationships that exist between the various classes can be seen in Figure 15.

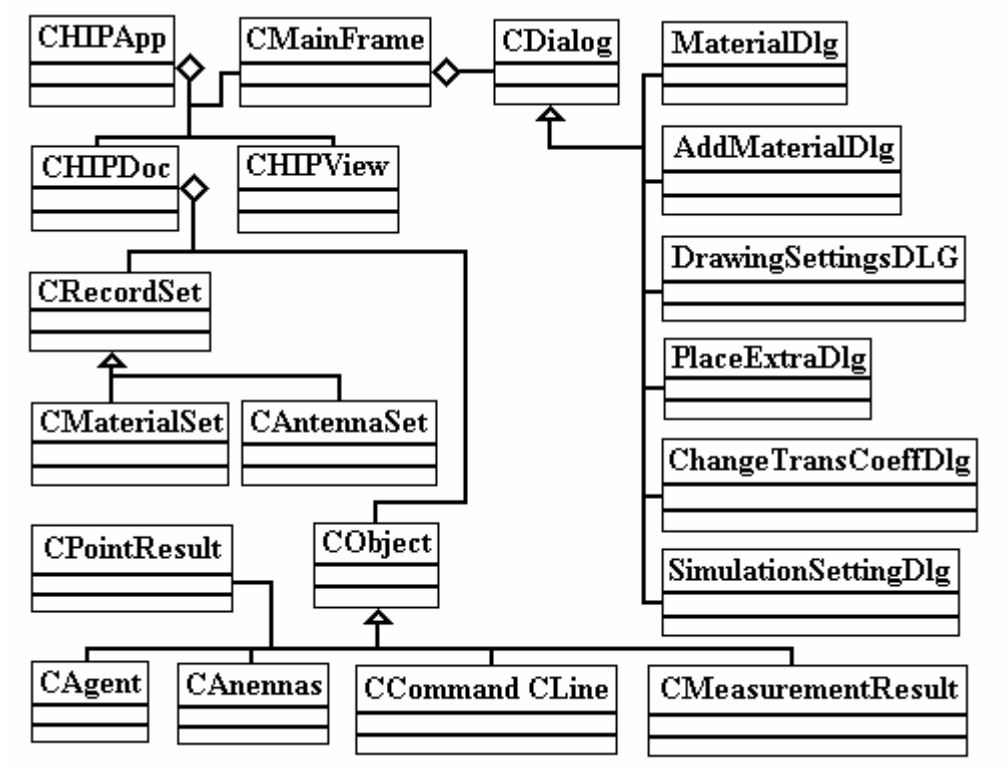

Figure 15: A simplified class diagram for the developed application.

From Figure 1 it can be seen that an object oriented approach was adopted for the design. Cohesion which is a measure of the mutual affinity between the classes is strong and coupling which indicates the dependency between classes/objects is low. This will ultimately lead to a robust, maintainable and upgradeable design as the occurrence of a problem in any given class will not cause the entire application to stop functioning. Maintainability and upgradeability was ensured by both producing well commented code, see Appendix D and by good code reuse.

4. Conclusion

By utilizing an object oriented approach during the development of the prediction tool, a highly modular design resulted. Cohesion and coupling between the various modules was found to be strong and low respectively, resulting in a robust, maintainable and updateable application.

Appendix D:
Application Method Overview

1. Introduction

To ensure maintainability and updatability, well commented code is essential. In light of this, contained within this document are detailed descriptions of the developed WLAN propagation prediction tools various methods by class. Each description describes in detail what each method does along with its various dependencies, objects and variables. The application's classes can be found in appendix G which contains a compact disc of its source code.

2. Application Classes

2.1 AddMaterialDlg

void AddMaterialDlg::AddRecord (int *ID*, CString *Name*, CString *Frequency*)

Description: This method adds a new record to the materials record set.

Dependencies:

AddNew() - Sets up record set in add mode.

Update() - Required to complete AddNew() function.

Requery() - Rebuilds the record set.

GetRValue(~) - Returns (int) the red component of a COLORREF variable.

GetGValue(~) - Returns (int) the Green component of a COLORREF variable.

GetBValue(~) - Returns (int) the Blue component of a COLORREF variable.

atof(~) - Does a CString to double conversion.

Variables and Objects:

CMaterialSet set - Materials record set.

Int ID - New materials ID.

CString Name - New material name.

CString Frequency - New material frequency.

COLORREF Colour - The selected material colour in RGB space.

Bugs:

To do:

void AddMaterialDlg::OnCancel () [protected]

Description: Closes the window, returning the focus to the calling method.

Dependencies:

OnOK() - Simply closes the dialog window.

Variables and Objects:

Bugs:

To do:

void AddMaterialDlg::OnColourSelect () [protected]

Description: This method displays the CColorDialog which is a standard MFC dialog that allows for the selection of a colour. On closing the CColorDialog using its Ok button,

the ColourSelected variable is set to true and the selected colour is stored in Colour variable which is of type COLORREF.

Dependencies:

DoModal() - This method invokes a modal dialog box and returns the dialog-box result when done.

GetColor() - Returns the selected COLORREF variable from a CColorDialog.

Variables and Objects:

BOOL ColourSelected - TRUE if a colour was selected.

COLORREF Colour - The selected material colour in RGB space.

CColorDialog dialog - The colour selection dialog window.

Bugs:

To do:

BOOL AddMaterialDlg::OnInitDialog () [protected, virtual]

Description: Called when the dialog is initialized, this method sets the current selection in the frequency combo box to its first entry.

Dependencies:

SetCurSel(~) - Sets the focus on the specified combo box item.

Variables and Objects:

CComboBox m_Frequency - The frequency selection combo box.

Bugs:

To do:

void AddMaterialDlg::OnOK () [protected, virtual]

Description: This method is called when the Add Material dialog is dismissed with Ok button and does the following:

1. Opens the materials record set.
2. Gets the name and frequency of the specified material and gets them into the required format.
3. CHECK 1: Was a material name specified.
4. CHECK 2: Was a material colour specified.
5. CHECK 3: Was the specified colour white (free space).
6. CHECK 4: Was the specified colour black (antennas).
7. CHECK 5: Was a valid transmission coefficient entered.
8. Having done and passed the above checks, the newly specified material is then added in one of two ways. If the materials record set is empty, the material is simply added using the AddRecord(~) function and the material set closed. Or else if the record set is not empty the following is done:
    1) The ID of the last record in the set is obtained.
    2) CHECK 6: The selected material colour is not already in use.
    3) CHECK 7: The selected material name/frequency combination is not already in use.

If both of the above checks pass, the newly specified material is then added to the record set.

Dependencies:

TrimLeft() - Removes any white space preceding a CString.

Open() - Opens the specified database to allow access to its records.

Close() - Closes the specified database.

GetRecordCount() - Returns the number of records in a record set.

MoveNext() - Moves to the next record in the database.

MoveFirst() - Moves to the first record in the database.

MoveLast () - Moves to the last record of a record set.

UpdateData(~) - By passing in TRUE, the entered variables are copied to their respective member variables.

GetLBText(~) - Returns the CString at the specified index of a list box.

OnOK() - Simply closes the dialog.

GetLength() - Returns the number of characters in a CString.

AfxMessageBox(~) - Simply displays the specified text in a message box.

RGB(~) - Specifies a COLORREF variable.

IsEOF() - TRUE until the end of the record set is reached.

GetRValue(~) - Returns (int) the red component of a COLORREF variable.

GetGValue(~) - Returns (int) the Green component of a COLORREF variable.

GetBValue(~) - Returns (int) the Blue component of a COLORREF variable.
CompareNoCase(~) - Compares two specified CStrings, ignoring case.

AddRecord() - Adds the specified material to the material record set.

Variables and Objects:

CMaterialSet set - Materials record set.

CString Name - Specified material name.

int index - The index of the selected material frequency.

CComboBox m_Frequency - The frequency selection combo box.

CString Frequency - The selected material frequency.

BOOL ColourSelected - TRUE if a material colour selected.

double m_TransCoeff - The specified material transmission - coefficient.

int LastID - The ID of the last material in the material - record set.

MaterialDlg temp - An object of type MaterialDlg used to gain access to its DoubleToCString(~) method.

int Check1 - 1 if material name is already in use.

int Check2 - 1 if material frequency is already in use.

Bugs:

To do:

2.2 CAgent

CAgent::CAgent (CPoint *CurrPoint,* double *dY,* double *dX,* int *Octant,* double *MagnitudeOffset,* int *SourceAnt*)

Description: Being a class constructor, this method is called each time an object of this class is created. This method is simply used to initialize some required variables.

Dependencies:

Variables and Objects:

CPoint m_CurrPoint - The agents current point in the drawing plane.

int m_iSourceAnt - The agents source antenna.

int m_iOctant - The agents octant of propagation.

double m_dY - Variable used in Bresenham's Algorithm.

double m_dX - Variable used in Bresenham's Algorithm.

double m_d - Variable used in Bresenham's Algorithm.

double m_dD - Variable used in Bresenham's Algorithm.

double m_dU - Variable used in Bresenham's Algorithm.

double m_dMagnitudeOffset - The agents initial magnitude offset.

CPoint m_origin - The agents origin.

Bugs:

To do:

2.3 CAntenna

CAntenna::CAntenna (LPCTSTR *Type*, CPoint *Location*, int *ID*, int *Orientation*)

Description: Being a class constructor, this method is called each time an object of this class is created. This method is simply used to initialize some required variables.

Dependencies:

Format(~) - This function writes formatted data to a CString. In this method it is used to get a CString representation of an integer.

Variables and Objects:

CString m_Type - The name of the antenna.

CPoint m_Location - The location of the antenna in the drawing plane.

int m_Orientation - The antennas location.

CString m_csID - The antennas ID.

Bugs:

To do:

void CAntenna::Draw (CDC * *pDC*)

Description: When called, this method draws the antennas representation to the screen at the required co-ordinates through the active device context. The antennas ID, name and a picture are drawn.

Dependencies:

SelectObject(~) - Selects the specified object into a device context.

MoveTo(~) - Moves to the specified point in a device context.

LineTo(~) - Draws a line to the specified point in a device context using the selected drawing object.

SetTextColor(~) - Sets the required colour of the text to be output.

SetTextAlign(~) - Sets the alignment of the text to be output.

TextOut(~) - Outputs (draws through device context) the specified text at the specified point in the raster.

Variables and Objects:

CPen lpen - The newly created drawing object.

CPen* pOldPen - The old drawing object.

CDC* pDC - A pointer to the active device context.

Bugs:

To do:

void CAntenna::Serialize (CArchive & *ar*)

Description: Called when loading and saving a document, this function saves and loads an objects specified variables from an archive (the specified file).

Dependencies:

IsStoring() - TRUE: Archive is storing data.

Variables and Objects:

CString m_Type - The name of the antenna.

CPoint m_Location - The location of the antenna in the drawing plane.

int m_Orientation - The antennas location.

CString m_csID - The antennas ID.

CArchive &ar - The archive (file) being read or written to.

Bugs:

To do:

2.4 CCommand

CCommand::CCommand (LPCTSTR *in*)

Description: Being a class constructor, this method is called each time an object of this class is created. This method is simply used to initialize some required variables.

Dependencies:

Variables and Objects:

CString- m_Command - The user specified or application generated command.

Bugs:

To do:

void CCommand::Serialize (CArchive & *ar*)

Description: Called when loading and saving a document, this function saves and loads an objects specified variables from an archive (the specified file).

Dependencies:

IsStoring() - TRUE: Archive is storing data.

Variables and Objects:

CString m_Command - The user specified or application generated command.

CArchive &ar - The archive (file) being read or written to.

Bugs:

To do:

2.5 ChangeTransCoeffDlg

void ChangeTransCoeffDlg::OnOK () [protected, virtual]

Description: Called when the user closes the dialog using the Ok button. All that this method does is update its member variables with the user inputted ones. It also checks that the new Transmission Coefficient is in the range of 0 to 1.

Dependencies:

UpdateData(~) - By passing in TRUE, the entered variables are copied to their respective member variables.

OnOK() - Simply closes the dialog.

Variables and Objects:

Double m_TransCoeff - User inputted Transmission Coefficient.

Bugs:

To do:

2.6 CHIPDoc

CHIPDoc::CHIPDoc () [protected]

Description: This is the document classes default constructor. It is used to initialize the required variables as well as determining the initially specified pixel size.

Dependencies:

RGB(~) - Returns the specified COLORREF variable.

pow(~) - Raises a specified double to another specified double.

Variables and Objects:

COLORREF m_Colour - The current drawing colour (white).

CPoint m_ptCurrent - The current drawing point (0m,0m).

int m_InitialAgentCount - The initial agent count (10).

double m_dFirst_Zone_Exp - The path loss exponent of the first Fresnel zone (2).

double m_dOther_Zone_Exp - The path loss exponent of the Fresnel zones other than the first (2.5).

double m_dDiameter_First_Zone - The diameter of the first fresnel zone (10m).

double m_dSimCutOffLevel - The simulation cut-off level (-100 dB).

double m_dAntLoss - The initial antenna loss at 1m (0 db).

BOOL m_bMeasurement - TRUE: Simulation by measurement.

BOOL m_bDataBase - TRUE: Simulation by database coefficients.

CString m_csDrawingSize - The initial drawing size (250m,250m).

double m_dPixelSize - The initial pixel size.

double m_dSimWavelenght - The initial simulation wave length.

double m_dDistanceMoved - The distance moved from the source antenna.

BOOL m_bTracerLinesValid - TRUE: Current tracer lines are valid.

Bugs:

To do:

CAgent * CHIPDoc::AddAgent (CPoint *CurrPoint,* double *dY*, double *dX*, int *Quadrant,* double *MagnitudeOffset,* int *SourceAnt*)

Description: This function is called to add a new agent to the agent object array. In more detail it does the following:

1. Creates a new agent object with the required attributes.
2. Next it checks that there is enough memory and adds the agent if there is.

Note: If there was not enough memory to add the agent, an error message will be displayed and the created agent object removed. This method also returns the newly created agent.

Dependencies:

Add(~) - Adds a new object to a CObjectArray.

AfxMessageBox(~) - Displays the specified text in a simple dialog box.

Delete() - Used to delete an object of type CMemoryException.

Variables and Objects:

CAgent* pAgent - Newly created agent object.

CObjectArray m_oaAgents - Agent object array.

CMemoryException* perr - Memory exception object.

Bugs:

To do:

CAntenna * CHIPDoc::AddAntenna (LPCTSTR *in*, CPoint *pt*, int *ID*, int *ori*)

Description: This function is called to add a new antenna to the antenna object array. In more detail it does the following:

1. Creates a new antenna object with the required attributes.
2. Next it checks that there is enough memory and adds the antenna if there is.
3. When a new antenna is added, the document is marked as dirty, which will prompt the user to save the document on exiting the application.

Note: If there was not enough memory to add the antenna, an error message will be displayed and the created antenna object removed. This method also returns the newly created antenna.

Dependencies:

Add(~) - Adds a new object to a CObjectArray.

SetModifiedFlag() - Marks the document as dirty.

AfxMessageBox(~) - Displays the specified text in a simple dialog box.

Delete() - Used to delete an object of type CMemoryException.

Variables and Objects:

CAntenna* pAntenna - Newly created antenna object.

CObjectArray m_oaAntennas - Antenna object array.

CMemoryException* perr - Memory exception object.

Bugs:

To do:

CCommand * CHIPDoc::AddCommand (LPCTSTR *in*)

Description: This function is called to add a new command to the command object array. In more detail it does the following:

1. Creates a new command object with the required attributes.
2. Next it checks that there is enough memory and adds the command if there is.
3. When a new command is added, the document is marked as dirty, which will prompt the user to save the document on exiting the application.

Note: If there was not enough memory to add the command, an error message will be displayed and the created command object removed. This method also returns the newly created command.

Dependencies:

Add(~) - Adds a new object to a CObjectArray.

SetModifiedFlag() - Marks the document as dirty.

AfxMessageBox(~) - Displays the specified text in a simple dialog box.

Delete() - Used to delete an object of type CMemoryException.

Variables and Objects:

CCommand* pCommand - Newly created command object.

CObjectArray m_oaCommands - Command object array.

CMemoryException* perr - Memory exception object.

Bugs:

To do:

CLine * CHIPDoc::AddLine (CPoint *ptFrom*, CPoint *ptTo*)

Description: This function is called to add a new line to the line object array. In more detail it does the following:

1. Creates a new line object with the required attributes and using these attributes, generates other needed attributes.
2. Next it checks that there is enough memory and adds the line if there is.
3. When a new line is added, the document is marked as dirty, which will prompt the user to save the document on exiting the application.

Note: If there was not enough memory to add the line, an error message will be displayed and the created line object removed. This method also returns the newly created line and invalidates and current tracer lines.

Dependencies:

Add(~) - Adds a new object to a CObjectArray.

SetModifiedFlag() - Marks the document as dirty.

AfxMessageBox(~) - Displays the specified text in a simple dialog box.

Delete() - Used to delete an object of type CMemoryException.

SetupLineVariables() - This method takes the lines start and end points and determines the max/min x/y co-ordinates.

Variables and Objects:

Cline* pLine - Newly created line object.

CObjectArray m_oaLines - Line object array.

CMemoryException* perr - Memory exception object.

BOOL m_bTracerLinesValid - FALSE: Current tracer lines are invalid.

Bugs:

To do:

CMeasurementResult * CHIPDoc::AddMeasurementResult (CPoint *Position*, double *Magnitude*, int *ID*)

Description: This function is called to add a new measurement result to the point result object array. In more detail it does the following:

1. Creates a new measurement result object with the required attributes.
2. Next it checks that there is enough memory and adds the measurement result if there is.
3. When a new measurement result is added, the document is marked as dirty, which will prompt the user to save the document on exiting the application.

Note: If there was not enough memory to add the measurement result, an error message will be displayed and the created measurement result object removed. This method also returns the newly created measurement result.

Dependencies:

Add(~) - Adds a new object to a CObjectArray.

SetModifiedFlag() - Marks the document as dirty.

AfxMessageBox(~) - Displays the specified text in a simple dialog box.

Delete() - Used to delete an object of type CMemoryException.

Variables and Objects:

CMeasurementResult* pMeasurementResult - Newly created measurement result object.
CObjectArray m_oaMeasurementResults - measurement result object array.
CMemoryException* - perr - Memory exception object.

Bugs:

To do:

CPointResult * CHIPDoc::AddPointResult (CPoint *Position*, double *Magnitude*, BOOL *Origional*)

Description: This function is called to add a new point result to the point result object array. In more detail it does the following:

1. Creates a new point result object with the required attributes.
2. Next it checks that there is enough memory and adds the point result if there is.
3. When a new point result is added, the document is marked as dirty, which will prompt the user to save the document on exiting the application.

Note: If there was not enough memory to add the point result, an error message will be displayed and the created point result object removed. This method also returns the newly created point result.

Dependencies:

Add(~) - Adds a new object to a CObjectArray.

SetModifiedFlag() - Marks the document as dirty.

AfxMessageBox(~) - Displays the specified text in a simple dialog box.

Delete() - Used to delete an object of type CMemoryException.

Variables and Objects:

CPointResult* pPointResult - Newly created point result object.

CObjectArray m_oaPointResults - Point result object array.

CMemoryException* perr - Memory exception object.

Bugs:

To do:

void CHIPDoc::ClearBackGround ()

Description: This method clears the DIB section by reloading the backing DIB bitmap.

Dependencies:

Load(~) - Belongs to the DIBSectionLite class and loads a bitmap into the required DIB section.

Variables and Objects:

DIBSectionLite m_dibsection - Oject encapsulating DIB section.

Bugs:

To do:

void CHIPDoc::ClearPointResults ()

Description: When called, this method deletes the CPointResult objects contained within the m_oaPointResults object array. It also removes the pointers to the deleted point result objects.

Dependencies:

RemoveAll() - This method retrieves all the pointers from this array and frees all memory used for pointer storage.

GetSize() - Returns (int) the number of objects in a CObjectArray.

Variables and Objects:

int PRNum - Number of point results.

CObjectArray m_oaPointResults - Applications point result object array.

Bugs:

To do:

void CHIPDoc::DeleteAgent (int *nIndex*)

Description: This method deletes the specified agent (nIndex) from the agent object array.

Dependencies:

RemoveAt(~) - Removes the specified object from a CObjectArray.

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaAgents - Agent object array.

Bugs:

To do:

void CHIPDoc::DeleteAntenna (int *nIndex*)

Description: This method deletes the specified antenna (nIndex) from the antenna object array and invalidates any current tracer lines.

Dependencies:

RemoveAt(~) - Removes the specified object from a CObjectArray.

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaAntenas - Antenna object array.

BOOL m_bTracerLinesValid - FALSE: Current tracer lines are - invalid.

Bugs:

To do:

void CHIPDoc::DeleteCommand (int *nIndex*)

Description: This method deletes the specified command (nIndex) from the command object array.

Dependencies:

RemoveAt(~) - Removes the specified object from a CObjectArray.

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaCommands - Command object array.

Bugs:

To do:

void CHIPDoc::DeleteContents () [virtual]

Description: This method is called each time a document is closed and does the following:

1. Reloads a blank backing DIB.
2. Gets the number of objects in the lines, Commands, Antennas, Measurements and Point Results arrays.

3. Loops through each one of the above mentioned object arrays and deletes their contained objects.
4. Resets each of the above arrays as by deleting their objects, you don't remove the pointers to them.
5. Resets the current drawing colour to white and resets the current drawing position to (0m,0m).
6. Shows the above changes on the documents active view.

Dependencies:

Load(~) - Belongs to the DIBSectionLite class and creates a DIB section from the specified bitmap (in this case a blank one).

GetSize() - Returns (int) the number of objects in a CObjectArray.

RemoveAll() - This method retrieves all the pointers from this array and frees all memory used for pointer storage.

UpdateView() - Updates the documents active view.

Variables and Objects:

int LiNum - Number of lines.

int CoNum - Number of commands.

int AnNum - Number of antennas.

int PRNum - Number of point results.

int MNum - Number of measurements.

CObjectArray m_oaLines - Applications line object array.

CObjectArray m_oaCommands - Applications command object array.

CObjectArray m_oaAntennas - Applications antenna object array.

CObjectArray m_oaMeasurements - Applications measurement object array.

CObjectArray m_oaPointResults - Applications point result object array.

COLORREF m_Colour - The current drawing colour (white).

CPoint m_ptCurrent - The current drawing point (0m,0m).

Bugs:

To do:

void CHIPDoc::DeleteLine (int *nIndex*)

Description: This method deletes the specified line (nIndex) from the line object array and invalidates any current tracer lines.

Dependencies:

RemoveAt(~) - Removes the specified object from a CObjectArray.

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaLines - Line object array.

BOOL m_bTracerLinesValid - FALSE: Current tracer lines are invalid.

Bugs:

To do:

void CHIPDoc::DeleteMeasurement (int *nIndex*)

Description: This method deletes the specified measurement (nIndex) from the measurement object array.

Dependencies:

RemoveAt(~) - Removes the specified object from a CObjectArray.

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaMeasurements - Measurement object array.

Bugs:

To do:

void CHIPDoc::DeletePointResult (int *nIndex*)

Description: This method deletes the specified point result (nIndex) from the point result object array.

Dependencies:

RemoveAt(~) - Removes the specified object from a CObjectArray.

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaPointResults - Point Result object array.

Bugs:

To do:

void CHIPDoc::DisplayAntennas ()

Description: This method redraws the antenna objects in the m_oaAntennas object array on the DIB section (m_dibsection). The way it does this is by creating a dummy memory device context which it passes into the GetMemoryDC(~) method to obtain a device context with the DIB section pre-selected in it. Having done this standard GDI drawing functions can be used through the created device context on the DIB section. To draw the antennas, the following process is followed:

1. Determine the number of antennas in the antenna object array.
2. If their are any antennas, loop through all the antennas and draw them.

On completion, the method releases the obtained device context such that other methods can work with the DIB section.

Dependencies:

GetMemoryDC(~) - Belongs to the DIBSectionLite class and creates a device context with the specified DIB section selected.

ReleaseMemoryDC() - Belongs to the DIBSectionLite class and releases the DIB section from the created device context.

GetAntennaCount() - Belongs to the document class and returns the number of user entered antennas.

Draw(~) - Belongs to the CAntenna class and draws the antenna object on the specified device context.

GetAntenna(~) - Belongs to the document class and returns the specified - antenna object.

Variables and Objects:

int AntCount - Stores the number of entered antennas.

CAntenna* pAntenna - Temp object used to retrieve CAntenna objects.

CDC* pDC - Pointer to device context with DIB section selected.

CDC* dc - Pointer to dummy device context

DIBSectionLite m_dibsection - Oject encapsulating DIB section.

Bugs:

To do:

void CHIPDoc::DisplayLines ()

Description: This method redraws the line objects in the m_oaLines object array on the DIB section (m_dibsection). The way it does this is by creating a dummy memory device context which it passes into the GetMemoryDC(~) method to obtain a device context with the DIB section pre-selected in it. Having done this standard GDI drawing functions can be used through the created device context on the DIB section. To draw the lines, the following process is followed:

1. Determine the number of lines in the line object array.
2. If their are any lines, loop through all the lines and draw them.

On completion, the method releases the obtained device context such that other methods can work with the DIB section.

Dependencies:

GetMemoryDC(~) - Belongs to the DIBSectionLite class and creates a device context with the specified DIB section selected.

ReleaseMemoryDC()- Belongs to the DIBSectionLite class and releases the DIB section from the created device context.

GetLineCount() - Belongs to the document class and returns the number of user entered lines.

Draw(~) - Belongs to the CLine class and draws the line object on the specified device context.

GetLine(~) - Belongs to the document class and returns the specified line object.

Variables and Objects:

int LineCount - Stores the number of entered lines.

CLine* pLine - Temp object used to retrieve CLine objects.

CDC* pDC - Pointer to device context with DIB section - selected.

CDC* dc - Pointer to dummy device context

DIBSectionLite m_dibsection - Object encapsulating DIB section.

Bugs:

To do:

void CHIPDoc::DisplayMeasurements ()

Description: This method redraws the measurements in the m_oaMeasurements object array on the DIB section (m_dibsection). The way it does this is by creating a dummy memory device context which it passes into the GetMemoryDC(~) method to obtain a device context with the DIB section pre-selected in it. Having done this standard GDI drawing functions can be used through the created device context on the DIB section. To draw the measurements, the following process is followed:

1. Determine the number of measurements in the measurement object array.
2. If their are any measurements, loop through all the measurements and draw them.

On completion, the method releases the obtained device context such that other methods can work with the DIB section.

Dependencies:

GetMemoryDC(~) - Belongs to the DIBSectionLite class and creates a device context with the specified DIB section selected.

ReleaseMemoryDC() - Belongs to the DIBSectionLite class and releases the DIB section from the created device context.

GetMeasurementCount()- Belongs to the document class and returns the number of user entered measurements.

Draw(~) - Belongs to the CMeasurment class and draws the measurements on the specified device context.

GetMeasurement(~) - Belongs to the document class and returns the specified measurement object.

Variables and Objects:

int MeasCount - Stores the number of entered measurements.

CMeasurement* pMeasurement - Temp object used to retrieve CMeasurement objects.

CDC* pDC - Pointer to device context with DIB section selected.

CDC* dc - Pointer to dummy device context

DIBSectionLite m_dibsection - Oject encapsulating DIB section.

Bugs:

To do:

CAgent * CHIPDoc::GetAgent (int *nIndex*)

Description: This method returns a pointer to the specified agent (nIndex) in the agent object array.

Dependencies:

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaAgents - Agent object array.

Bugs:

To do:

int CHIPDoc::GetAgentCount ()

Description: Returns (int) the number of agents in the agent object array.

Dependencies:

GetSize() - Returns (int) the number of objects in a CObjectArray.

Variables and Objects:

CObjectArray m_oaAgentss - Agent object array.

Bugs:

To do:

CAntenna * CHIPDoc::GetAntenna (int *nIndex*)

Description: This method returns a pointer to the specified antenna (nIndex) in the antenna object array.

Dependencies:

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaAntennas - Antenna object array.

Bugs:

To do:

int CHIPDoc::GetAntennaCount ()

Description: Returns (int) the number of antennas in the antenna object array.

Dependencies:

GetSize() - Returns (int) the number of objects in a CObjectArray.

Variables and Objects:

CObjectArray m_oaAntennas - Antenna object array.

Bugs:

To do:

CCommand * CHIPDoc::GetCommand (int *nIndex*)

Description: This method returns a pointer to the specified command (nIndex) in the command object array.

Dependencies:

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaCommands - Command object array.

Bugs:

To do:

int CHIPDoc::GetCommandCount ()

Description: Returns (int) the number of commands in the command object array.

Dependencies:

GetSize() - Returns (int) the number of objects in a CObjectArray.

Variables and Objects:

CObjectArray m_oaCommands - Command object array.

Bugs:

To do:

CLine * CHIPDoc::GetLine (int *nIndex*)

Description: This method returns a pointer to the specified line (nIndex) in the line object array.

Dependencies:

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaLines - Line object array.

Bugs:

To do:

int CHIPDoc::GetLineCount ()

Description: Returns (int) the number of lines in the line object array.

Dependencies:

GetSize() - Returns (int) the number of objects in a CObjectArray.

Variables and Objects:

CObjectArray m_oaLines - Line object array.

Bugs:

To do:

CMeasurementResult * CHIPDoc::GetMeasurement (int *nIndex*)

Description: This method returns a pointer to the specified measurement (nIndex) in the measurement object array.

Dependencies:

Variables and Objects:

int nIndex - The position of the object in the CObjectArray.

CObjectArray m_oaMeasurements - Measurement object array.

Bugs:

To do:

int CHIPDoc::GetMeasurementCount ()

Description: Returns (int) the number of measurements in the measurement object array.

Dependencies:

GetSize() - Returns (int) the number of objects in a CObjectArray.

Variables and Objects:

CObjectArray m_oaMeasurements - Measurement object array.

Bugs:

To do:

void CHIPDoc::GetNextPoint ()

Description: This method calculates the agents next point, given only the agents initial point and a line gradient. To do this an adapted Bresenham's Algorithm which is based on the mid point theorem is used. It was chosen as it requires only integer additions and subtractions which are very fast. In using this approach we need to consider each one of the octants separately. Note: We only ever consider the first agent in the agent object array as once it has been terminated, the next agent will take its place.

Dependencies:

GetAgent(~) - Returns the specified CAgent object from the agent object - array.

Variables and Objects:

CAgent* agent - The first agent object of the CAgent object array.

Bugs:

To do:

CPointResult * CHIPDoc::GetPointResult (int *nIndex*)

Description: This method returns a pointer to the specified point result (nIndex) in the point result object array.

Dependencies:

Variables and Objects:

int nIndex - The position of the object in the - CObjectArray.

CObjectArray m_oaPointResults - Point Result object array.

Bugs:

To do:

int CHIPDoc::GetPointResultCount ()

Description: Returns (int) the number of point results in the point result object array.

Dependencies:

GetSize() - Returns (int) the number of objects in a CObjectArray.

Variables and Objects:

CObjectArray m_oaPointResults - Point result object array.

Bugs:

To do:

BOOL CHIPDoc::OnOpenDocument (LPCTSTR *lpszPathName*) [virtual]

Description: Called each time a document is opened, this method displays the document lines, antenna and any made measurements.

Dependencies:

DisplayLines() - Displays the documents lines.

DisplayAntennas() - Displays the documents antenna.

DisplayMeasurements() - Displays the documents measurements.

Variables and Objects:

Bugs:

To do:

void CHIPDoc::OnSettingsDrawingsettings () [protected]

Description: This method is called when the Drawing Settings button in the Settings menu option is pressed. It initially creates and object of type DrawingSettingDlg which it will later use to display the Drawing Settings dialog. It then initializes the Drawing Setting dialog variables and displays the Drawing Setting dialog. On the termination of the Drawing Settings dialog (with OK), the method finally updates the corresponding document class member variables with those altered in the Drawing Setting dialog. Note: The final two lines of this method, calculate and set the application pixel size in meters given the user specified drawing size. The larger dimension of the specified drawing size will be used in this calculation.

Dependencies:

DoModal() - Belongs to the CDialog class and displays the corresponding modal dialog and returns the dialog - result when done.

pow(~) - Raises a specified double to another specified double.

Variables and Objects:

DrawingSettingDlg dlg - Dialog object of type DrawingSettingDlg.

CSize m_csDrawingSize - CSize variable that contains the user specified drawing size. double m_dPixelSize - The calculated pixel size.

Bugs:

To do:

void CHIPDoc::OnSettingsSimulationsettings () [protected]

Description: This method is called when the Simulation Settings button in the Settings menu option is pressed. It initially creates and object of type **SimulationSettingDlg** which it will later use to display the Simulation Settings dialog. It then initializes the Simulation Setting dialog variables and displays the Simulation Setting dialog. On the termination of the Simulation Settings dialog (with OK), the method finally updates the corresponding document class member variables with those altered in the Simulation Setting dialog.

Dependencies:

DoModal() - Belongs to the CDialog class and displays the corresponding modal dialog and returns the dialog result when done.

Variables and Objects:

SimulationSettingDlg dlg - Dialog object of type SimulationSettingDlg.

int m_InitialAgentCount - Both the dialog and document classes have this member variable which specifies the number of initial agents to use.

Bugs:

To do:

void CHIPDoc::OnUpdateDistanceMoved (CCmdUI * *pCmdUI*)

Description: This method is called each time the current position of the mouse pointer changes (OnMouseMove). It gets a CString representation of the mouse pointers displacement whilst the left mouse button is depressed, enables the status bar pane and then changes its text to that of the mouse pointers displacement. Note: This displacement is in meters.

Dependencies:

Format(~) - This function writes formatted data to a CString. In this method it is used to get CString representations of a double.

Enable(~) - This method enables or disables the required user-interface item.

SetText(~) - This function sets the text of the required user-interface item.

Variables and Objects:

CString str - A CString of the mouse pointers displacement.

CCmdUI* pCmdUI - Used only within an ON_UPDATE_COMMAND_UI handler, an object of this type is used to govern the functioning of an interface item.

Bugs:

To do:

void CHIPDoc::OnUpdateIndicatorX (CCmdUI * *pCmdUI*)

Description: This method is called each time the current x-position of the mouse pointer needs updating (OnMouseMove). It gets a CString representation of the mouse pointers current x-position, enables the status bar pane and then changes its text to that of the mouse pointers current x-position.

Dependencies:

Format(~) - This function writes formatted data to a CString. In this method it is used to get CString representations of an integer.

Enable(~) - This method enables or disables the required user-interface item.

SetText(~) - This function sets the text of the required user-interface item.

Variables and Objects:

CString str - A CString of the mouse pointers current x-position.

CCmdUI* pCmdUI - Used only within an ON_UPDATE_COMMAND_UI handler, an object of this type is used to govern the functioning of an interface item.

Bugs:

To do:

void CHIPDoc::OnUpdateIndicatorY (CCmdUI * *pCmdUI*)

Description: This method is called each time the current y-position of the mouse pointer needs updating (OnMouseMove). It gets a CString representation of the mouse pointers current y-position, enables the status bar pane and then changes its text to that of the mouse pointers current y-position.

Dependencies:

Format(~) - This function writes formatted data to a CString. In this method it is used to get CString representations of an integer.

Enable(~) - This method enables or disables the required user-interface item.

SetText(~) - This function sets the text of the required user-interface item.

Variables and Objects:

CString str - A CString of the mouse pointers current y-position.

CCmdUI* pCmdUI - Used only within an ON_UPDATE_COMMAND_UI handler, an object of this type is used to govern the functioning of an interface item.

Bugs:

To do:

void CHIPDoc::OnUpdateSelectedMaterial (CCmdUI * *pCmdUI*)

Description: This method is called each time the selected drawing material is changed. It simple enables the required status bar pane and replaces its text with that of the newly selected materials name.

Dependencies:

Enable(~) - This method enables or disables the required user-interface item.

SetText(~) - This function sets the text of the required user-interface item.

Variables and Objects:

CCmdUI* pCmdUI - Used only within an ON_UPDATE_COMMAND_UI handler, an object of this type is used to govern the functioning of an interface item.

Bugs:

To do:

void CHIPDoc::Serialize (CArchive & *ar*) [virtual]

Description: This method reads or writes the required object arrays as well as the specified drawing size to or from an archive.

Dependencies:

IsStoring() - TRUE if the archive is storing data.

Serialize(~) - Does the actual reading/writing from/to an archive.

UpdateAllViews(~) - Used to reload the command history list box.

Variables and Objects:

CObjectArray m_oaLines - Applications line object array.

CObjectArray m_oaCommands - Applications command object array.

CObjectArray m_oaAntennas - Applications antenna object array.

CObjectArray m_oaMeasurements - Applications measurement object array.

CArchive& ar - Archive being read/written.

Bugs:

To do:

void CHIPDoc::UpdateView ()

Description: When called, this method updates all of the documents views.

Dependencies:

UpdateAllViews(~) - NULL: Update all views.

Variables and Objects:

Bugs:

To do:

2.7 CHIPView

CHIPView::CHIPView () [protected]

Description: This method is used to initialize variables that will be used in this class to the required values.

Dependencies:

NullifyPointerArray() - This method sets the entire point result pointer array to NULL.

Variables and Objects:

CSize m_BackGroundSize - The size of the drawing area in pixels (5000 by - 5000).

double m_ZoomFactor - The initial drawing zoom factor.

double pi - PI.

BOOL m_bBoundary - TRUE: A boundary has been specified.

CPoint m_cpBoundaryTL - Top left corner of specified boundary.

CPoint m_cpBoundaryBR - Bottom right corner of specified boundary.

Bugs:

To do:

BOOL CHIPView::AntMeasPosOk ()

Description: This methods checks that the user specified boundary encompasses all of the entered measurements and antenna. It does this by looping through all of the measurement objects and the antenna object, comparing their positions to that of the specified boundary.

Dependencies:

GetDocument() - Returns a pointer to the active document class.

GetMeasurementCount() - Returns the number of user entered measurements.
GetMeasurement(~) - Returns a pointer to the specified CMeasurementResult object.
AfxMessageBox(~) - Simply displays the specified text in a message box.

GetAntenna(~) - Returns a pointer to the specified CAntenna object.

Variables and Objects:

int m_iNumMeasResults - The number of entered measurements.

CMeasurementResult* pMeasResult - A pointer to a CMeasurementResult object.
CAntenna* pAntenna - A pointer to a CAntenna object.

Bugs:

To do:

BOOL CHIPView::AssessAgent (CAgent * *pAgent*)

Description: This method is called by the OnBasicLog() method. As its input parameter it takes in a pointer to a CAgent object. Using this pointer, it checks that the agents current position is within the raster. If the agent is found to not be within the raster or if it is found to be outside of the specified simulation boundary, it is deleted and TRUE returned.

Dependencies:

GetDocument() – Gets a pointer to the active document.

DeleteAgent() - Belongs to the document class and deletes the agent at the specified index.

Variables and Objects:

CAgent* pAgent - Pointer to object of type CAgent.

Bugs:

To do:

void CHIPView::AveragePointResults ()

Description: This method is used to approximate point results at points where no point results were calculated. It does this by taking a weighted average of the four closest known point results. In more detail, it does the following:

1. Loops through the point result pointer matrix until a null pointer (i.e. no point result calculated).
2. Using this null pointers position as a reference, it then moves in the -ve and +ve x and y directions until one of the following is found: a) An original point result (no an averaged one). b) A wall of and type (you don't want to average over walls as this causes inaccuracy). It does this by considering the corresponding pixel colour and any colour other than black or white will be considered a wall or obstacle. c) The specified simulation boundary is left.
3. Whilst doing this (2.) the number of moves is recorded to each valid point result as it will be needed in the weighted average calculation and on finding a valid point result, the variable m_bValidPR is set to TRUE prompting the following point (4.).
4. Having found a valid point result, the number of moves taken to the point result as well as the point results magnitude are stored in corresponding position in two different arrays. The total number of moves to all the valid point results is also stored. These values will be used later.
5. The final step in the method, calculates the magnitude of the new point result based on a weighted average of the above mentioned obtained valid point results and generates the required point result at the previously null position.

Note: This method is very selective and will use the maximum number of obtainable, valid, point results (i.e. 1 being the minimum and 4 being the maximum).

Dependencies:

GetDocument() - Returns a pointer to the active document class.

GetMemoryDC(~) - Belongs to the DIBSectionLite class and creates a device context with the specified DIB section selected.

ReleaseMemoryDC() - Belongs to the DIBSectionLite class and releases the DIB section from the created device context.

SetUpFeedBackDisplay(~) - Initializes the feedback display with the required heading and range.

TidyUpFeedBackDisplay(~) - Removes the feedback heading and sets the current feedback process control position to 0.

GetPixel(~) - Returns (COLORREF) the colour of the specified pixel.

Format(~) - This method writes formatted string data into a string. In this method, it is used to convert a double variable to a CString.

pow(~) - Raises the specified double to a specified power (in this case -1 or division).
GetParentFrame() - Returns a pointer to the applications Frame class.

Variables and Objects:

CDC* dc - A pointer to a dummy device context.

CDC* pDC - A pointer to a device context with the backing DIB pre-selected.

CPoint m_cpBoundaryTL - The top left corner of the simulation boundary.

CPoint m_cpBoundaryBR - The bottom right corner of the simulation boundary.

int m_dADividend[4] - Stores the number of moves taken in each direction when finding an original point result.

Double m_dMagnitude[4] - Stores the magnitudes of the found point results in each direction.

Double m_dTMagnitude - The calculated average of the surrounding point results.

int m_dTDividend - The total number of moves taken in each direction.

int m - Tracking variable which holds the current x value of the point result being calculated.

int n - Tracking variable which holds the current y value of the point result being calculated.

int o - Tracking variable which holds the number of moves taken to a valid point result.
COLORREF m_crPixel The colour of the current pixel in the tracking algorithm.

BOOL m_bValidPR - TRUE when a valid, original point result is found.

CPoint m_cpPoint - The position of the newly averaged point result.

CPointResult* pResult - The newly averaged point result.

CPointResult* m_pPRMatrix[][] - The point result pointer array.

Bugs:

To do:

void CHIPView::ClearSimResults ()

Description: When this method is called, it clears any simulation results by reloading the DIB section with a blank bitmap and then by redisplaying the required walls, Measurements and antennas on the blank DIB section.

Dependencies:

GetDocument() - Returns a pointer to the active document.

Load(~) - Belongs to the DIBSectionLite class and creates a DIB section from the specified bitmap (in this case a - blank one).

DisplayLines() - From the document class and loops through all the lines in the line object array, redrawing them.

DisplayAntennas() - From the document class and loops through all the antennas in the antenna object array, redrawing them.

DisplayMeasurements() - From the document class and loops through all the measurement results in the measurement results object array, redrawing them.

Variables and Objects:

CDIBSectionLite m_dibsection - Created DIB section on which all application drawing takes place.

Bugs:

To do:

void CHIPView::DisplayPointResults ()

Description: This method is called when the generated point results need drawing. The method initially creates a memory device context with the backing DIB pre-selected in it. It them loops through the point result pointer matrix (m_pPRMatrix) and draws the corresponding point results if their pointers exist. The drawing is done using the Draw(~) method of the point result class which takes as its parameter, the above mentioned device context. The reason behind keeping a 2-D array of pointers to the point result objects it to allow for the direct accessing of point result objects using relative screen co-ordinate addressing. Note: this method also sets up the feedback display in the applications frame with the required heading and range (SetUpFeedBackDisplay(~)) and on completion clears and resets the feedback display (TidyUpFeedBackDisplay()).

Dependencies:

GetDocument() - Gets a pointer to the active document.

Draw(~) - Belongs to the CPointResult class and draws the Point result object on the specified device context.

GetMemoryDC(~) - Belongs to the DIBSectionLite class and creates a device context with the specified DIB section - selected.

ReleaseMemoryDC() - Belongs to the DIBSectionLite class and releases the DIB section from the created device context.

SetUpFeedBackDisplay(~) - Initializes the feedback display with the required heading and range.

TidyUpFeedBackDisplay(~) - Removes the feedback heading and sets the current feedback process control position to 0.

Variables and Objects:

CDC* dc - Pointer to dummy memory device context.

CDC* pDC - Pointer to memory device context.

int i - Used to loop throungh m_pPRMatrix[i][].

int j - Used to loop throungh m_pPRMatrix[][j].

DIBSectionLite m_dibsection - Object encapsulating DIB section.

CPointResult* m_pPRMatrix[][] - Matrix of pointers to CPointResult objects used to both keep track of the generated CPointResult objects and allow them to be easily accessible via the corresponding screen co-ordinates.

Bugs:

To do:

CPoint * CHIPView::ExtractCoOrdinates (LPCTSTR m_In)

Description: This method takes as its input, the user entered command. By considering is structure with regards to the position of certain characters ('[',',', and ']') it extracts the required co-ordinates. It then validates these co-ordinates by initially converting them the pixel values (uses the user specified drawing size) and then by ensuring that they lie within the raster. It returns the validated co-ordinates in pixel values. In more detail:

1. Gets a CString representation of the LPCTSTR input argument.
2. Finds the position if the first bracket, comma and last bracket in the newly created CString. 3. CHECK 1: Ensures that the above extracted characters are in the correct sequence.
3. CHECK 2: Ensures that something exists between the first bracket and the comma as well as the comma and the last bracket.
4. Removes the sections of the CString between the first bracket and the comma and the comma and the second bracket.
5. Converts these values To doubles (will be in meters).
6. Finds out how many pixels the above double values are, given the user specified drawing size.
7. CHECK 3: Ensures that the calculated pixel values are still in the raster.
8. Returns a pointer to a CPoint object containing the required co-ordinates in pixel values.

NOTE: If any of the above checks fail, a NULL pointer is returned.

Dependencies:

Find(~) - Searches a string for the first match of a substring and returns its index (indexed from 0).

Mid(~) - Extracts a substring of specified length and starting index.

atof(~) - Converts a character string to a double-precision floating-point value.
GetDocument() - Gets a pointer to the active document.

pow(~) - Raises the specified double to a specified power (in this case -1 or division).

Variables and Objects:

CString m_csIn - CString representation of input argument.

Int FirstB - Position of first bracket in input argument.

Int Comma - Position of comma in input argument.

Int LastB - Position of last bracket in input argument.

CString m_X - Extracted CString between first bracket and comma.

CString m_Y - Extracted CString between comma and last bracket.

Double m_dX - Double representation of m_X.

Double m_dY - Double representation of m_Y.

Double m_dPixelSize - Pixel size, given user specified drawing size.

Int m_iX - Pixel representation of m_dX.

Int m_iY - Pixel representation of m_dY.

CPoint m_Return - CPoint of above calculated pixel values.

CPoint* cpReturn - Pointer to above CPoint object which gets returned.

Bugs:

To do:

CString CHIPView::ExtractLetters (LPCTSTR in)

Description: Returns the passed in CString without the following characters: 1,2,3,4,5,6,7,8,9,0,[,],(,) and .

Dependencies:

Remove(~) - Removes the specified character from a CString.

Variables and Objects:

CString m_csIn - CString representation of passed in LPCTSTR to allow for CString operations.

Bugs:

To do:

void CHIPView::GeneratePrimaryAgents ()

Description: This method generates a specified number of agents for each antenna placed. Its only required user input is that of the specified number of primary agents set throungh the simulation settings dialog. The detailed **Description** of this methods operation is as follows:

1. Gets the number of user specified antennas.
2. Gets the user specified initial agent count.
3. CHECK 1: Are there any antennas? NO: Exit method YES: Proceed
4. Create an object of type **CAntenna** to gain access to the user specified antenna objects.
5. Loop through the specified antennas and generate the required number of primary agents.

NOTE: The line drawing algorithm is octant dependent so each primary agent will be generated for a specific octant i.e. 1-8. On the creation of an agent object, its origin, source antenna, magnitude offset and gradient information is also specified. The octancy dependency results in 8 times the number of user specified primary agents being drawn.

Dependencies:

GetDocument() - Returns a pointer to the active document class.

GetAntennaCount() - Returns (int) the number of user specified antennas.

GetAntenna(~) - Returns a pointer to the specified antenna in the antenna object array found in the document class.

AddAgent(~) - Adds the specified agent to the agent object array of the document class.

Variables and Objects:

int AntCount - The number of user specified antennas.

int m_InitialAgentCount - The number of user specified primary agents.

CAntenna *AntTemp - Pointer to object of type CAntennas to allow access to the user specified antennas.

int AntPos - A for loop stepping variable used to iterate through the entered antennas.

Bugs:

To do:

void CHIPView::GetSimulationConstants ()

Description: This method, when called, updates the views simulation constants variables with the corresponding variables in the document class. This method is called before any of the simulation mechanisms are carried out.

Dependencies:

GetDocument() - Returns a pointer to the active document class.

Variables and Objects:

BOOL m_bMeasurement - TRUE: Intersection by measurement results.

BOOL m_bDataBase - TRUE: Intersection by material coefficients found in database.
Double m_dPixelSize - The user specified pixel size (i.e. depends on specified floor plan size).

Double m_dSimCutOffLevel - The specified simulation cut-off level.

Double m_dFirst_Zone_Exp - The specified first zone path loss exponent.

Double m_dOther_Zone_Exp - The other than first zone path loss exponent.

Double m_dDiameter_First_Zone - The specified diameter of the first zone.

Double m_dAntLoss - The specified antenna loss at 1m.

Bugs:

To do:

void CHIPView::InhibitAgentActivity (CAgent * pAgent)

Description: Prevents an agent from generating a point result whilst it is within 1m from its source antenna. This is done as the utilized propagation equations were normalized at 1m to gain antenna independence. It does this by calculating the distance between the agents current point and it origin and compares it to unity. NOTE: The first line of code gives the agent a dummy magnitude value (not a possible value) such that the AssessAgent method can be used as it will cause an assert if the state of the agent magnitude is undefined.

Dependencies:

GetDocument() - Gets a pointer to the active document class.

GetNextPoint() - Calculates the agents next point and moves the agent.

AssessAgent(~) - Checks to see whether the agent is still in the raster and deletes it if it is not.

pow(~) - Belongs to the math's class and raises a specified number to another specified number.

Variables and Objects:

double m_dPixelSize - The user specified pixel size (determined from the user specified drawing size).

double m_dDist - The calculated distance the agent has moved.

CAgent* pAgent - Methods input argument.

Bugs:

To do:

void CHIPView::IntersectionByMeasurement (CAgent * pAgent, CDC * pDC, CMaterialSet * pSet)

Description: This method handles an intersection when the from measurement simulation mechanism is used. In more detail:

1. Gets the colour of the intersected wall.
2. Loops through the applications materials database to determine the transmission coefficient of the intersected wall.
3. Loops through the measurement result object array to determine if a measurement result exists for the intersected material.
4. If no measurement result was found, the materials name is added to the agents intersection history where as if a measurement result is found, the agents magnitude is reduced accordingly.

Note: If the angle of arrival boolean variable is set and a and a measurement result is found, then the following is carried out:

1. A check is carried out to determine if the wall is vertical or horizontal.
2. Depending on the outcome of 1., two different approaches are used. One for octants 1, 8, 4 and 5 and the other for octants 2, 3, 6 and 7. In each of the above approaches the angle of arrival is calculated and used to alter the walls transmission coefficient (Coeff/cos(angle of arrival)).

Dependencies:

GetPixel(~) - Returns the colour of the specified pixel in a device context.

MoveFirst() - Moves to the first record of a record set. IsEOF() TRUE until the end of the record set is reached.

GetRValue(~) - Returns (int) the red component of a COLORREF variable.

GetGValue(~) - Returns (int) the Green component of a COLORREF variable.

GetBValue(~) - Returns (int) the Blue component of a COLORREF variable.

MoveNext() - Moves to the next record in a record set.

GetDocument() - Returns a pointer to the active document.

log10(~) - Returns the log base 10 of the specified number.

DeleteAgent(~) - Deletes an agent at the specified index from the agent object array.

pow(~) - Raises a specified double to the power of another specified double.

GetMeasurementCount() - Belongs to the document class and returns the number of user entered measurements.

GetMeasurement(~) - Belongs to the document class and returns the specified measurement object.

Format(~) - This function writes formatted data to a CString.

Empty() - Empties associated CString object.

Variables and Objects:

COLORREF m_colour - Variable which stores the intersected wallscolour.

CAgent* pAgent - Input argument (the intersecting agent).

CMeasurementResult* pMR - Pointer to measurement object.

CDC* pDC - Input argument (a DC with backing DIB selected).

CMaterialSet* pSet - Input argument (the materials record set).

BOOL m_bNotEqual - TRUE if intersected wall found in database.

BOOL M_bMeasResult - TRUE if valid measurement result is found.

BOOL m_bAngleOfArrival - TRUE if angle of arrival is required in calculation.

double m_dOffset - The dB representation if walls transmission coefficient.

Bugs:

To do:

void CHIPView::IntersectionBySimulation (CAgent * pAgent, CDC * pDC, CMaterialSet * pSet)

Description: This method handles an intersection when the from database simulation mechanism is used. In more detail:

1. Gets the colour of the intersected wall.
2. Loops through the applications materials database to determine the transmission coefficient of the intersected wall. 3. Calculates the dB value of the transmission coefficient and add it to the agents magnitude offset.

Note: If the angle of arrival boolean variable is set, then the following is carried out:

1. A check is carried out to determine if the wall is vertical or horizontal.
2. Depending on the outcome of 1., two different approaches are used. One for octants 1, 8, 4 and 5 and the other for octants 2, 3, 6 and 7. In each of the above approaches the angle of arrival is calculated and used to alter the walls transmission coefficient (Coeff/cos(angle of arrival)).

Dependencies:

GetPixel(~) - Returns the colour of the specified pixel in a device context.

MoveFirst() - Moves to the first record of a record set.

IsEOF() - TRUE until the end of the record set is reached.

GetRValue(~) - Returns (int) the red component of a COLORREF variable.

GetGValue(~) - Returns (int) the Green component of a COLORREF variable.

GetBValue(~) - Returns (int) the Blue component of a COLORREF variable.

MoveNext() - Moves to the next record in a record set.

GetDocument() - Returns a pointer to the active document.

log10(~) - Returns the log base 10 of the specified number.

DeleteAgent(~) - Deletes an agent at the specified index from the agent object array.

pow(~) - Raises a specified double to the power of another specified double.

Variables and Objects:

COLORREF m_colour - Variable which stores the intersected walls colour.

CAgent* pAgent - Input argument (the intersecting agent).

CDC* pDC - Input argument (a DC with backing DIB selected).

CMaterialSet* pSet - Input argument (the materials record set).

BOOL m_bNotEqual - TRUE if intersected wall found in database.

double m_dOffset - The dB representation if walls transmission coefficient.

BOOL m_bAngleOfArrival - TRUE if angle of arrival is required in calculation.

Bugs:

To do:

void CHIPView::IsIntersection (CAgent * pAgent, CDC * pDC, CPoint PrevPoint, CMaterialSet * pSet)

Description: This method takes as its input parameters the current agent, a DC loaded with the backing DIB, the agents previous point and the materials database record set. It does this to test for intersections between agents and walls by doing the following:

1. Gets the colour of the agents current point.
2. Checks to see if the colour is not black or white (white and black represent free space and antennas respectively.)
3. Checks if the agents previous point was white i.e. the intersection has just occurred.
4. If as a result of the above checks it is determined that an intersection has occurred, the simulation mechanism is then examined and the appropriate method called.

Dependencies:

IntersectionBySimulation(~) - Method that handles an intersection when the simulation mechanism uses predefined reflection and transmission coefficients.

IntersectionByMeasurement(~)- Method that handles an intersection when the simulation mechanism determines the required reflection and transmission coefficients from measurement.

RGB(~) - Specifies a COLORREF variable.

GetPixel(~) - Returns the colour of the specified pixel in a device context.

Variables and Objects:

CAgent *pAgent - A pointer to the current agent object.

CDC *pDC - A pointer to a device context with the backing DIB pre-selected.

CPoint PrevPoint - The current agents previous point.

CMaterialSet* pSet - A pointer to the materials record set.

BOOL m_bDataBase - TRUE if simulation uses predefined coefficients.

BOOL m_bMeasurement - TRUE if simulation generates coefficients from measurements.

Bugs:

void CHIPView::IsNewPointResult (CAgent * pAgent, CDC * pDC)

Description: This method calculates a new point result if no point result currently exists at the agents current position. It does this as follows:

1. CHECK 1: Does a point result exist at the agent current position and is the agents
2. Calculates the distance the agent has moved based on the value of m_bFresnelZone. TRUE indicates that the Fresnel zones should be used.
3. Using this distance, it calculates the agents free loss (also included is losses due to any previous intersections).
4. Using 2-4, the existing point result is altered to reflect changes due to the impinging agent
5. A record is also kept of all the impinging agents intersections at the point result and the agents magnitude is updated.

Dependencies:

GetDocument() - Gets a pointer to the active document class.

pow(~) - Belongs to the math's class and raises a specified number to another specified number.

AddPointResult(~)- Adds a specified point result to the point result object array of the document class.

log10(~) - Returns the log base 10 of the specified number.

GetPixel(~) - Returns the colour (RGB) of the specified pixel.

Variables and Objects:

double m_dPixelSize - The user specified pixel size (determined from the user specified drawing size).

CAgent* pAgent - Methods input argument.

double m_dDistance - Agents current displacement.

double m_dMagnitude - Agents magnitude due to its displacement.

double m_dMagnitudeNew - The new calculated point result magnitude value.

CPointResult* pResult - A pointer to an object of type CPointResult used to alter an existing point result.

double pi - The number 3.14 etc.

BOOL m_bFresnelZone - TRUE: Use Fresnel Zones.

Bugs:

To do:

void CHIPView::LineTo (LPCTSTR in)

Description: This method is responsible for drawing a line in response to a user LineTo command. It does this as follows:

1. Extracts the required co-ordinates using the ExtractCoOrdinates method which returns a pointer to a CPoint object, containing the "to" co-ordinates in pixel values. This method also validates the input command.
2. Gets the current or "from" co-ordinate from the document class.
3. Adds the required line to the line object array of the document class if a material has been selected.
4. Changes the "from" co-ordinate to the "to" co-ordinate in the document class if a material has been selected.
5. Adds the required command to the command object array of the document class if a material has selected. 6. Draws the line and finally updates the command list in the frame class.

Dependencies:

ExtractCoOrdinates(~) - Gets a pointer to a CPoint object containing the extracted pixel value representation of the user inputted command.

GetDocument() - Gets a pointer to the active document.

AddLine(~) - Adds a line object to the line object array of the document class.

AddCommand(~) - Adds a command to the command object array of the document class.

DisplayLines(~) - Belongs to the document class and redraws all the lines in the line object array on the backing DIB.

ReDrawCommands(~) - Reloads all the command objects in the command object array in the command history list in the application frame.

RGB(~) - Returns the specified COLORREF variable.

Variables and Objects:

CPoint* m_cpCoOrd - Pointer to CPoint object containing required "to" pixel values.

CPoint ptFrom - "from" CPoint object.

CPoint ptTo - CPoint object containing "to" pixel values.

CLine* pLine - Pointer to newly added CLine object.

CCommand* pCommand - Pointer to newly added CComand object.

Bugs:

To do:

BOOL CHIPView::MeasurementPositionsOk ()

Description: This method checks that all the entered measurement results were placed over valid simulation results.

Dependencies:

GetDocument() - Returns a pointer to the active document.

GetMeasurement(~) - Returns the specified CMeasurementResult object from the measurement object array of the document class.

GetMeasurementCount() - Returns the number of entered measurement results.

Variables and Objects:

int m_iNumMeasResults - The number of entered measurement results.

CMeasurementResult* pMeasResult - A pointer to a measurement result object used to step through them.

Bugs:

To do:

void CHIPView::MoveAgents ()

Description: This method is one of the applications core methods and does the following:

1. Generates the specified number of primary agents.
2. Sets-up the feedback display with the required text.
3. Opens the materials record set.
4. Gets a pointer to a device context with the required backing DIB pre-selected.
5. Loops through all the agents in the agent object array and moves them. On each agent move, its previous point is stored (used in the intersection testing) and its current position and magnitude validated to ensure that it has not reached the specified cut-off magnitude or left the specified simulation boundary.
6. During the agent moving, the agents activity is inhibited whilst it is within 1m of the antenna (magnitude calculations were normalized at 1m).
7. On the completion of this method, the feedback display is reset, the DIB section released and the material record set closed.

Dependencies:

GeneratePrimaryAgents() - Generates the user specified primary agents.

SetUpFeedBackDisplay(~) - Initializes the feedback display with the required heading and range.

TidyUpFeedBackDisplay(~) - Removes the feedback heading and sets the current feedback process control position to 0.

Open() - Opens the specified database to allow access to its records.

Close() - Closes the database.

GetDocument() - Returns a pointer to the active document class.

GetMemoryDC(~) - Belongs to the DIBSectionLite class and creates a device context with the specified DIB section - selected.

ReleaseMemoryDC() - Belongs to the DIBSectionLite class and releases the DIB section from the created device context.

GetAgentCount() - Returns the number of agents in the agent object array.

GetAgent(~) - Returns the specified CAgent object.

InhibitAgentActivity(~) - Prevents the agents from creating point results when they are within 1m from their source antenna.

IsNewPointResult(~) - Generates a point result from an agents move.

GetNextPoint() - Returns the agents next position (based on a line drawing algorithm).

GetParentFrame() - Returns a pointer to the applications Frame class.

SetPos(~) - Belongs to the CProcessControl class and sets the current process control position to that specified.

IsIntersection(~) - Handles the case in which an agent has interacted with a wall.

SetPos(~) - Belongs to the CProcessControl class and sets the current process control position to that specified.

Variables and Objects:

int i - Tracks the number of agents deleted and is used in the feedback display process bar.

CMaterialSet Set - Material database record set object.

CMaterialSet* pSet - Pointer to above object.

CDC* dc - A dummy device context pointer.

CDC* pDC - A pointer to a device context with the required backing DIB pre-selected.

CAgent* pAgent - A CAgent object used to contain the various agents used within this method.

Bugs:

To do:

void CHIPView::MoveTo (LPCTSTR in)

Description: This method is responsible for moving the drawing cursor in response to a user MoveTo command. It does this as follows:

1. Extracts the required co-ordinates using the ExtractCoOrdinates method which returns a pointer to a CPoint object, containing the "to" co-ordinates in pixel values. This method also validates the input command.
2. Updates the current position variable in the document class.
3. Finally adds the required command to the command object array of the document class.

Dependencies:

ExtractCoOrdinates(~) - Gets a pointer to a CPoint object containing the extracted pixel value representation of the user inputted command.

GetDocument() - Gets a pointer to the active document.

ReDrawCommands(~) - Reloads all the command objects in the command object array in the command history list in the application frame.

Variables and Objects:

CPoint* m_cpCoOrd - Pointer to CPoint object containing required "to" pixel values.

CPoint ptTo - CPoint object containing "to" pixel values. CCommand* pCommand Pointer to newly added CComand object.

Bugs:

To do:

void CHIPView::NewCommand (LPCTSTR in)

Description: This is called from the frame class after a user command has been entered (the frame class waits for a ']' character). It then gets a CString representation of the above to allow for string operations on the entered command. Having done this it extracts the text part of the entered command and calls the respective method.

Dependencies:

GetDocument() - Gets a pointer to the active document.

Find(~) - Finds the index if the specified character in a CString.

Left(~) - Extracts the portion of CString left of the specified index.

MoveTo(~) - Moves the drawing cursor to the specified point.

LineTo(~) - Draws a line from the current drawing cursor point to a specified point.

Variables and Objects:

CString m_In - CString representation of the user command.

Bugs:

To do:

void CHIPView::NullifyPointerArray ()

Description: Sets each pointer in the m_pPRMatrix[5000][5000] which tracks the generated point result objects to NULL.

Dependencies:

Variables and Objects:

int i - Goes from 0 to 4999 and used to step through rows of m_pPRMatrix.

int j - Goes from 0 to 4999 and used to step through columns of m_pPRMatrix.

Bugs:

To do:

void CHIPView::OnAdvancedLog () [protected]

Description: This method carries out the Advanced log simulation mechanism which accounts of free space loss, intersections, angle of arrival and Fresnel Zones. It does this as follows:

1. Gets the required simulation constants and sets up the required simulation environment.
2. Checks that the specified antenna lies within the specified simulation boundary and returns if it does not.
3. Sets the required simulation governing variables to their required values (m_bAngleOfArrival=TRUE and m_bFresnelZone=TRUE).
4. It then does the required moving of the generated agents governed by the variables set in 3.
5. Having done this, the deposited point results are averaged. This involves initially setting up the required feedback and on completion, resetting the feedback display.
6. The method terminates with code to redraw the generated and averaged point results, the specified antenna and any drawn lines.

Dependencies:

SetUpFeedBackDisplay(~) - Initializes the feedback display with the required heading and range.

TidyUpFeedBackDisplay(~) - Removes the feedback heading and sets the current feedback process control position to 0.

AveragePointResults() - Generates point results at regions where they were not calculated by taking the weighted average of up to four of the closest simulated point results.

GetSimulationConstants() - Gets the requires simulation constants.

SetupSimulationEnviroment() - Sets up the simulation environment.

AntMeasPosOk() - Returns TRUE if the specified antenna is within the specified simulation boundary.

DisplayPointResults() - Displays all the point results in the point result object array.

GetDocument() - Returns a pointer to the active document class.

DisplayLines() - From the document class and loops through all the lines in the line object array, - redrawing them.

DisplayAntennas() - From the document class and loops through all the antennas in the antenna object array, redrawing them.

ReDraw() - Invalidates and redraws the view to reflect changes.

MoveAgents() - Moves the specified number of agents around the raster. Their movement is governed by the specified simulation mechanism and the specified simulation constants.

Variables and Objects:

BOOL m_bAngleOfArrival - TRUE: Use angle of arrival.

BOOL m_bFresnelZone - TRUE: Use Fresnel Zones.

Bugs:

To do:

void CHIPView::OnBasicLog () [protected]

Description: This method carries out the Basic log simulation mechanism which accounts of free space loss and intersections. It does this as follows:

1. Gets the required simulation constants and sets up the required simulation environment.
2. Checks that the specified antenna lies within the specified simulation boundary and returns if it does not.
3. Sets the required simulation governing variables to their required values (m_bAngleOfArrival=FALSE and m_bFresnelZone=FALSE).
4. It then does the required moving of the generated agents governed by the variables set in 3.
5. Having done this, the deposited point results are averaged. This involves initially setting up the required feedback and on completion, resetting the feedback display.
6. The method terminates with code to redraw the generated and averaged point results, the specified antenna and any drawn lines.

Dependencies:

SetUpFeedBackDisplay(~) - Initializes the feedback display with the required heading and range.

TidyUpFeedBackDisplay(~) - Removes the feedback heading and sets the current feedback process control position to 0.

AveragePointResults() - Generates point results at regions where they were not calculated by taking the weighted average of up to four of the closest simulated point results.

GetSimulationConstants() - Gets the requires simulation constants.

SetupSimulationEnviroment() - Sets up the simulation environment.

AntMeasPosOk() - Returns TRUE if the specified antenna is within the specified simulation boundary.

DisplayPointResults() - Displays all the point results in the point result object array.

GetDocument() - Returns a pointer to the active document class.

DisplayLines() - From the document class and loops through all the lines in the line object array, redrawing them.

DisplayAntennas() - From the document class and loops through all the antennas in the antenna object array, redrawing them.

ReDraw() - Invalidates and redraws the view to reflect changes.

MoveAgents() - Moves the specified number of agents around the raster. Their movement is governed by the specified simulation mechanism and the specified simulation constants.

Variables and Objects:

BOOL m_bAngleOfArrival - TRUE: Use angle of arrival.

BOOL m_bFresnelZone - TRUE: Use Fresnel Zones.

Bugs:

To do:

void CHIPView::OnDraw (CDC * pDC) [virtual]

Description: This method is called each time the applications view requires redrawing. It initially gets a device context with the backing DIB pre-selected into it. Following this it maps the backing DIB onto the screen (taking into account the specified zoom factor) and releases the backing DIB such that it can be selected into another device context when required.

Dependencies:

GetMemoryDC(~) - Belongs to the DIBSectionLite class and creates a device context with the specified DIB section selected.

ReleaseMemoryDC()- Belongs to the DIBSectionLite class and releases the DIB section from the created device context.

StretchBlt(~) - Does the mapping from one device context to another, taking into account any size differences (used for zooming in/out).

GetDocument() - Returns a pointer to the active document class.

Variables and Objects:

CDC* pDC - Pointer to device context with DIB section selected.

CDC* dc - Pointer to dummy device context

DIBSectionLite m_dibsection - Object encapsulating DIB section.

Bugs:

To do:

BOOL CHIPView::OnEraseBkgnd (CDC * pDC) [protected]

Description: The framework calls this member function when the CWnd object background needs erasing. Changed to return FALSE to ensure flicker free drawing.

Dependencies:

Variables and Objects:

Bugs:

To do:

void CHIPView::OnHScroll (UINT nSBCode, UINT nPos, CScrollBar * pScrollBar) [protected]

Description: Called every time the horizontal scroll bar is moved. This method passes the current scroll position to the UpdateRulerInfo method to exact changes in the horizontal ruler display. It also invalidates the window forcing a redraw which reflects the scrolled to portion of the display.

Dependencies:

Invalidate(~) - Marks the display as "dirty" forcing a redraw.

GetScrollPosition(~) - Returns the current scroll position (int).

UpdateRulerInfo(~) - Updates the rulers to reflect a scroll change.

Variables and Objects:

UINT nSBCode - Specifies a scroll-bar code that indicates a scrolling request by the user i.e. SB_BOTTOM = Scroll to bottom.

UINT nPos - Contains the current scroll-box position. N.B. dependant on which nSBCode is specified.

CScrollBar* pScrollBar - Pointer to object of type CScrollBar which encapsulates the applications horizontal scroll bar.

Bugs:

To do:

void CHIPView::OnInitialUpdate () [protected, virtual]

Description: This method is called by the framework after the view is first attached to the document, but before the view is initially displayed. It has been altered to show the applications required rulers and initialize certain variables.

Dependencies:

NullifyPointerArray() - This method sets the entire point result pointer array to NULL.

ShowRulers(~) - When TRUE is passed in, the applications rulers are shown.

GetParentFrame() - Returns a pointer to the applications Frame class.

Variables and Objects:

CSize m_BackGroundSize - The size of the drawing area in pixels (5000 by - 5000).

double m_ZoomFactor - The initial drawing zoom factor.

double pi - PI.

BOOL m_bBoundary - TRUE: A boundary has been specified.

CPoint m_cpBoundaryTL - Top left corner of specified boundary.

CPoint m_cpBoundaryBR - Bottom right corner of specified boundary.

Bugs:

To do:

void CHIPView::OnIntermediateLog () [protected]

Description: This method carries out the Intermediate log simulation mechanism which accounts of free space loss, intersections, angle of arrival and Fresnel Zones. It does this as follows:

1. Gets the required simulation constants and sets up the required simulation environment.

2. Checks that the specified antenna lies within the specified simulation boundary and returns if it does not.
3. Sets the required simulation governing variables to their required values (m_bAngleOfArrival=TRUE and m_bFresnelZone=FALSE).
4. It then does the required moving of the generated agents governed by the variables set in 3.
5. Having done this, the deposited point results are averaged. This involves initially setting up the required feedback and on completion, resetting the feedback display.
6. The method terminates with code to redraw the generated and averaged point results, the specified antenna and any drawn lines.

Dependencies:

SetUpFeedBackDisplay(~) - Initializes the feedback display with the required heading and range.

TidyUpFeedBackDisplay(~) - Removes the feedback heading and sets the current feedback process control position to 0.

AveragePointResults() - Generates point results at regions where they were not calculated by taking the weighted average of up to four of the closest simulated point results.

GetSimulationConstants() - Gets the requires simulation constants.

SetupSimulationEnviroment() - Sets up the simulation environment.

AntMeasPosOk() - Returns TRUE if the specified antenna is within the specified simulation boundary.

DisplayPointResults()- Displays all the point results in the point result object array.

GetDocument() - Returns a pointer to the active document class.

DisplayLines() - From the document class and loops through all the lines in the line object array, - redrawing them.

DisplayAntennas() - From the document class and loops through all the antennas in the antenna object array, redrawing them.

ReDraw() - Invalidates and redraws the view to reflect changes.

MoveAgents() - Moves the specified number of agents around the raster. Their movement is governed by the specified simulation mechanism and the specified simulation constants.

Variables and Objects:

BOOL m_bAngleOfArrival - TRUE: Use angle of arrival.

BOOL m_bFresnelZone - TRUE: Use Fresnel Zones.

Bugs:

To do:

void CHIPView::OnLButtonDown (UINT nFlags, CPoint point) [protected]

Description: This method is called each time the left mouse button is depressed and responds in either of two ways:

A. If the shift button is depressed.
    1. Invalidates any current tracer lines
    2. If it is the first time this combination has been depressed, a device context is obtained with the backing DIB pre-selected onto which a dot is drawn (at the current mouse position) to indicate a possible corner of the simulation boundary to be specified.

3. If it is the second time this combination is used, once again a device context is obtained with the backing DIB pre-selected onto which a rectangle is drawn having opposite corners corresponding to the selected point in 2 and this latest selected point. Having done this, the user specified lines, antennas and measurement results are redrawn. Following this the final stage in this case determines out of the two specified points, which one constitutes the top left and bottom right corners of the simulation boundary rectangle.

B. If no key depressed The mouse cursor is captured and the starting point of a possible line (which is a function of both the current point and the current scroll bar positions) is set.

Dependencies:

GetKeyState(~) - Returns the state of the specified system key. If less than zero, indicates that key is depressed.

GetDocument() - Returns a pointer to the active document class.

GetMemoryDC(~) - Belongs to the DIBSectionLite class and creates a device context with the specified DIB section - selected.

ReleaseMemoryDC() - Belongs to the DIBSectionLite class and releases the DIB section from the created device context.

SelectObject(~) - Selects the specified object into a device context.

GetScrollPosition() - Returns (int) the scroll position of the view.

CreatePen() - Creates an object of type CPen which is used to draw on the screen through a device context.

AfxMessageBox(~) - Simply displays the specified text in a message box.

DisplayLines() - Displays the documents lines.

DisplayAntennas() - Displays the documents antenna.

DisplayMeasurements() - Displays the documents measurements.

ReDraw() - Invalidates and redraws to display to show any changes resulting from the entered command.

SetCapture() - Captures the mouse such that no other applications - can use it.

Variables and Objects:

BOOL m_bTracerLinesValid - FALSE: Current tracer lines are invalid.

BOOL m_bBoundaryTL - TRUE if no boundary.

CDC* dc - A dummy device context pointer.

CDC* pDC - A pointer to a device context with the required backing DIB pre-selected.

CDIBSectionLite m_dibsection - Created DIB section on which all application drawing takes place.

CPen penBlack - New CPen drawing object.

CPen* pOldPen - Old CPen drawing object.

CPoint m_cpBoundaryBR - Bottom right corner of specified boundary.

CPoint m_cpFrom - The starting point of a line.

Bugs:

To do:

void CHIPView::OnLButtonUp (UINT nFlags, CPoint point) [protected]

Description: This method is called each time the left mouse button is released. It initially checks that a material has been selected and that the shift key is not depressed and returns if it is i.e. if a boundary is being specified. It then releases the mouse such that it can be

used by another application and following this it retrieves the current mouse position which is a function of the windows current scroll position. Using the mouse's current position as well as its previous position (obtained when the left mouse button was depressed) it generates the required MoveTo and LineTo commands of the specified line. Towards the end of the method, the generated commands and line objects are added to their respective object arrays. The method concludes displaying the entered line and corresponding commands.

Dependencies:

GetDocument() - Returns a pointer to the active document class.

RGB(~) - Returns the specified COLORREF variable.

GetCapture() - Returns the capture state of the mouse cursor.

ReleaseCapture() - Releases the mouse cursor such that other applications can use it.

GetScrollPosition() - Returns (int) the scroll position of the view.

Format(~) - This function writes formatted data to a CString.

AddCommand(~) - Adds a command to the command object array of the document class.

AddLine(~) - Adds a line to the line object array of the document - class.

DisplayLines() - Displays the lines contained in the line object array.

ReDrawCommands() - Displays the commands contained in the command object array in the command history list box.

ReDraw() - Invalidates and redraws the view to reflect changes.

Variables and Objects:

CPoint m_cpTo - The ending point of the specified line.

CString m_csMoveTo - The required moveto command (starting point of line).

CString m_csLineTo - The required lineto command (end point if line).

CCommand* pCommand - The generated command object.

CLine* pLine - The generated line object.

Bugs:

To do:

void CHIPView::OnMaterials () [protected]

Description: This method is called in response to the Databases->Materials menu item being selected. It displays the Materials Menu dialog and on closing the dialog with Ok button, clears and reloads the materials list box in the applications frame.

Dependencies:

DoModal() - Loads a modal dialog box.

GetParentFrame() - Returns a pointer to the applications frame class.

ClearMaterialList() - Clears the materials list box in the applications frame.

LoadMaterialList() - Reloads the materials list box in the applications frame showing any changes.

Variables and Objects:

MaterialDlg dlg - An object of type MaterialDlg used to encapsulate the Materials Menu dialog.

Bugs:

To do:

void CHIPView::OnMouseMove (UINT nFlags, CPoint point) [protected]

Description: Called every time the mouse is moved, this function sets the variables m_X and m_Y in the document class to the current x and y mouse position taking into account the relative positions of the vertical and horizontal scroll sliders. It also updates both the vertical and horizontal ruler bars to reflect any changes governed by both the vertical and horizontal scroll sliders. All the feedback is in meters.

Dependencies:

GetDocument() - Gets a pointer to the current document.

GetScrollPosition() - Returns the current scroll position (int).

UpdateRulerInfo(~) - Updates the rulers to reflect a scroll change.

Variables and Objects:

int m_X - Current mouse position (x) in document class.

int m_Y - Current mouse position (y) in document class.

CPoint point - Current mouse position.

Double m_dPixelSize - The specified pixel size.

Bugs:

To do:

void CHIPView::OnRButtonDblClk (UINT nFlags, CPoint point) [protected]

Description: This method is called each time the right mouse button is double clicked. Its primary function is to load the Place Extra dialog box and handle its events. In more detail, it does the following:

1. Sends the user specified simulation cut-off level as well as simulated signal magnitude at the selected point to the dialog before it is displayed.
2. On closing the dialog with its ok bottom it checks the state of the m_bMeasurement and m_bAntenna boolean variables.
3. If m_bMeasurement is true or a measurement was specified, the method gets the current mouse cursor and view scroll positions. Having done this, it then ensures that the current tracer lines are valid and that the specified measurement result position lies on a tracer line and that the respective point result has a single material intersection history. It then adds the required measurement result to the measurement result object array in the document class and also the required command in a similar way. After which it reloads the command history list box.
4. If m_bAntenna is true of an antenna was specified, an antenna will be added if no current antenna exists and the required commands added to the command history list box.

Note: This method will only allow one antenna per simulation and on the addition of either a measurement or antenna the objects in the corresponding object array in the document class will be redrawn. The addition of either a measurement or antenna will also invalidate a previously specified simulation boundary.

Dependencies:

GetDocument() - Returns a pointer to the active document class.

DoModal() - Displays a modal dialog box.

AddMeasurementResult(~)- Adds a measurement result to the measurement result object array in the document class.

AddCommand(~) - Adds a command to the command object array in the document class.

ReDrawCommands() - Displays the commands contained in the command object array in the command history list box.

DisplayMeasurements() - Displays the measurement result object in the measurement result object array.

AddAntenna(~) - Adds an antenna to the antenna object array in the document class.

AfxMessageBox(~) - Simply displays the specified text in a message box.

DisplayAntennas() - Displays the antenna objects in the antenna object array in the document class.

ReDraw() - Invalidates and redraws the view to reflect changes.

GetScrollPosition() - Returns (int) the scroll position of the view.

Replace(~) - Replaces the specified character in a CString with another. NULL simply removes the specified character. Returns the number of characters replaced.

Variables and Objects:

PlaceExtraDlg dlg - An object of type PlaceExtraDlg which is displayed.

CPoint Position - The current mouse cursor position.

CMeasurementResult* pMR - The users specified measurement result.

CCommand* pCommand - The required command object.

CAntenna* pAntenna - The required antenna object.

BOOL m_bBoundaryTL - TRUE if no boundary.

Bugs:

To do:

void CHIPView::OnRButtonDown (UINT nFlags, CPoint point) [protected]

Description: This method is called each time the right button is depressed and simply writes the point results magnitude (at the current mouse cursors position) to the screen. In more detail, it does the following:

1. Checks that a point result exists at the mouse cursors current point and that the shift key is depressed.
2. Gets the required point results magnitude and formats the required string.
3. Gets a pointer to a DC with the backing DIB pre-selected.
4. Does the displaying of the magnitude and redraws the view.

Dependencies:

GetScrollPosition() - Returns (int) the scroll position of the view.

Format(~) - This function writes formatted data to a CString.

GetDocument() - Returns a pointer to the active document.

ReDraw() - Invalidates and redraws the view to reflect changes.

SetTextColor(~) - Sets the DC's text colour.

SetTextAlign(~) - Sets the DC's text alignment.

TextOut(~) - Writes the specified text to the DC.

GetKeyState(~) - Returns the state of the specified system key. If less than zero, indicates that key is depressed.

Variables and Objects:

CString m_csMagnitude - The CString representation of the required point results magnitude.

CDC* dc - A dummy device context.

CDC* pDC - The needed device context with backing DIB pre-selected.

Bugs:

To do:

void CHIPView::OnShowValidLines () [protected]

Description: This method is run before any measurement results can be entered. The reasons behind this are to prevent the user from entering a measurement result over an averaged point result. The basic operation of this method is as follows:

1. Gets the simulation constants.
2. Sets up the specified simulation environment.
3. Ensures that both angle of arrival as well as Fresnel zones will be used to establish the tracer lines. This will ensure maximum accuracy. It also sets the variable m_bMeasurement to TRUE thus ensuring that any predefined material coefficients will not be used (i.e. we want to calculate new ones).
4. It then moves the specified number of agents around the raster and on completion, displays their produced point results.
5. The final line of the method validates the generated tracer lines, allowing for measurement results to be entered.

Note: Any changes to the drawing will invalidate the tracer lines. This method also checks to see whether the specified antenna lies within the specified simulation boundary.

Dependencies:

GetSimulationConstants() - Gets the requires simulation constants.

SetupSimulationEnviroment() - Sets up the simulation environment.

AntMeasPosOk() - Returns TRUE if the specified antenna is within the specified simulation boundary.

DisplayPointResults() - Displays all the point results in the point result object array.

GetDocument() - Returns a pointer to the active document class.

DisplayLines() - From the document class and loops through all the lines in the line object array, redrawing them.

DisplayAntennas() - From the document class and loops through all the antennas in the antenna object array, redrawing them.

DisplayMeasurements() - From the document class and loops through all the measurement results in the measurement results object array, redrawing them.

ReDraw() - Invalidates and redraws the view to reflect changes.

MoveAgents() - Moves the specified number of agents around the raster. Their movement is governed by the specified simulation mechanism and the specified simulation constants.

Variables and Objects:

BOOL m_bAngleOfArrival - TRUE: Use angle of arrival.

BOOL m_bFresnelZone - TRUE: Use Fresnel Zones.

BOOL m_bMeasurement - TRUE: Use measurements.

BOOL m_bDataBase - TRUE: Use coefficients in database.

Bugs:

To do:

void CHIPView::OnSimulateSaveresults () [protected]

Description: Saves the current screen contents to a user specified DIB such that the simulation results can be distributed and viewed on any picture editor. NOTE: The CFileDialog is initialized such that only *.bmp file extensions can be used and the overwriting of files will be prompted.

Dependencies:

DoModal() - Invokes the modal CFileDialog box.

Save(~) - Belongs to the CDIBSectionLite class and saves the contents of the applications backing DIB to a specified bitmap file.

Variables and Objects:

CString m_csPath - The path of the specified bitmap file.

Bugs:

To do:

void CHIPView::OnUpdate (CView * pSender, LPARAM lHint, CObject * pHint) [protected, virtual]

Description: Called by the framework after the view's document has been modified. It has been altered to change both the vertical and horizontal scroll bars to reflect any zoom functions as well as update the required ruler information.

Dependencies:

SetScrollSizes(~) - Adjusts the scrolling characteristics of two scroll bars when the zoom factor is changed.

UpdateRulersInfo(~) - Adjusts the applications rulers when either the zoom factor has changed or the drawing size is altered.

Invalidate(~) - Marks the view for a redraw on the next UpdateWindow call.

UpdateWindow() - Redraws the view if required.

ReDrawCommands() - Redisplays the command history in the command history list box in the applications frame.

GetScrollPosition() - Returns (int) the current scroll position of both the vertical and horizontal scroll bars.

Variables and Objects:

float m_ZoomFactor - The current view zoom factor.

CSize m_BackGroundSize - The size of the backing DIB (5000x5000 pixels).

Bugs:

To do:

void CHIPView::OnVScroll (UINT nSBCode, UINT nPos, CScrollBar * pScrollBar) [protected]

Description: Called every time the vertical scroll bar is moved. This method passes the current scroll position to the UpdateRulerInfo method to exact changes in the vertical ruler display. It also invalidates the window forcing a redraw which reflects the scrolled to portion of the display.

Dependencies:

Invalidate(~) - Marks the display as "dirty" forcing a redraw.

GetScrollPosition(~) - Returns the current scroll position (int).

UpdateRulerInfo(~) - Updates the rulers to reflect a scroll change.

Variables and Objects:

UINT nSBCode - Specifies a scroll-bar code that indicates a scrolling request by the user i.e. SB_BOTTOM = Scroll to bottom.

UINT nPos - Contains the current scroll-box position. N.B. dependant on which nSBCode is specified.

CScrollBar* pScrollBar - Pointer to object of type CScrollBar which encapsulates the applications vertical scroll bar.

Bugs:

To do:

void CHIPView::OnZoomIn () [protected]

Description: This method is called in response to the Zoom->Zoom In menu item being selected. It initially shows the applications drawing rulers. It then multiplies the current drawing zoom factor by 2 and updates the applications rulers to reflect the required scale.

Dependencies:

GetParentFrame() - Returns a pointer to the applications frame class.

ShowRulers(~) - TRUE: show rulers

OnUpdate(~) - Called by the frame after the view has been altered (this does the scaling of the rulers see method).

Variables and Objects:

int m_ZoomFactor - The required zoom factor used in the OnDraw method by a DC StretchBlt function.

Bugs:

To do:

void CHIPView::OnZoomNormal () [protected]

Description: This method is called in response to the Zoom->Zoom Normal menu item being selected. It initially shows the applications drawing rulers. It then sets the current drawing zoom factor to one and updates the applications rulers to reflect the required scale.

Dependencies:

GetParentFrame() - Returns a pointer to the applications frame class.

ShowRulers(~) - TRUE: show rulers

OnUpdate(~) - Called by the frame after the view has been altered (this does the scaling of the rulers see method).

Variables and Objects:

int m_ZoomFactor - The required zoom factor used in the OnDraw method by a DC StretchBlt function.

Bugs:

To do:

void CHIPView::OnZoomOut () [protected]

Description: This method is called in response to the Zoom->Zoom Out menu item being selected. It initially checks that the current zoom factor is greater the 0.25 (i.e. you don't want to zoom out to much). It also ensures that the applications rulers are not shown if the zoom factor is less than one. If the above checks are passed, it divides the current drawing factor by 2 and updates the applications rulers to reflect the required scale.

Dependencies:

GetParentFrame() - Returns a pointer to the applications frame class.

ShowRulers(~) - TRUE: show rulers

OnUpdate(~) - Called by the frame after the view has been altered (this does the scaling of the rulers see method).

Variables and Objects:

int m_ZoomFactor - The required zoom factor used in the OnDraw method by a DC StretchBlt function.

Bugs:

To do:

void CHIPView::ReDraw ()

Description: Invalidates the view and forces a redraw.

Dependencies:

Invalidate(~) - Marks the view as "dirty" such that on the next update it will be redrawn.

UpdateWindow() - Initiates an update.

Variables and Objects:

Bugs:

To do:

void CHIPView::ReDrawCommands ()

Description: Called whenever the elements of the command history are altered i.e. whenever elements are added or removed.

Dependencies:

GetParentFrame() - Gets a pointer to a frame window if successful, otherwise NULL.

ClearHistoryList() - Removes all the elements in the command history list. LoadHistoryList() Reloads the command history list with any alterations.

Variables and Objects:

Bugs:

To do:

void CHIPView::ResolveMeasurementResult (CMeasurementResult * pMResult)

Description: This method takes a user specified measurement result and using the intersection history of the corresponding point result, determines the transmission coefficient of any intersected object. In more detail it does the following:

1. Before we begin it is important to realize that the corresponding point results intersection history will be in the following form (num)Name where num is the angle of arrival in radians and Name is the name of the intersected material.
2. With this in mind, the first stage of this method extracts the angle of arrival and gets a double representation of it.
3. Having done this, the material name is then extracted.
4. Having the above, the method then calculates the transmission coefficient of the intersected material by getting the difference between the simulated value at the specified point and the corresponding measured value. This difference is attributed to the material intersection and by using the cosine of the angle of intersection, the materials transmission coefficient is calculated.
5. The required variables of the passed in measurement result are then changed to contain the above calculated values.

Dependencies:

Replace(~) - Replaces the specified character in a CString with another. NULL simply removes the specified character.

atof(~) - Converts a CString to a double.

ExtractLetters(~) - Returns the letters contained within a CString.

SpanExcluding(~) - Extracts all the characters of a CString that are within the specified set.

cos(~) - Returns the cosine of the specified angle in radians.

Variables and Objects:

CString m_csAngle - CString of extracted angle.

double m_dAngle - Double of extracted angle.

CMeasurementResult* pMResult - The passed in measurement result object.

CString m_csMaterialName - CString of extracted material name.

double m_dDifference - The difference between the measured value at a point and the simulated value.

double m_dCoefficient - The calculated material transmission coefficient.

Bugs:

To do:

void CHIPView::SetUpFeedBackDisplay (CString heading, int range)

Description: This method sets up the feedback display which belongs to the Frame class. The feedback display consists of a read only edit box (m_eSimSection) and a process control bar (m_pcSimProcess). The setup process is as follows:

1. Select the first line of the edit box.
2. Change its text to the required text (heading).
3. Set the range of the process control to the required range (range).
4. Set the current process control position to 0 or the start position.

Dependencies:

SetSel(~) - Belongs to the CEdit class and sets the current edit focus on the specified line. N.B. -1 specifies entire line.

ReplaceSel(~) - Belongs to the CEdit class and changes the selected text to the specified text.

SetRange(~) - Belongs to the CProcessControl class the sets the range of the associated process control object.

SetPos(~) - Belongs to the CProcessControl class and sets the current process control position to that specified.

GetParentFrame() - Returns a pointer to the applications Frame class.

Variables and Objects:

CEdit m_eSimSection - CEdit control associated with the CEdit box in the applications frame.

CProcessControl m_pcSimProcess - CProcessControl control associated with the CProcessControl bar in the applications frame.

CString heading - Feedback heading (method argument).

int range - Process control range (method argument).

Bugs:

To do:

void CHIPView::SetupSimulationEnviroment ()

Description: This method sets up the simulation environment by doing the following:

1. Clears the backing DIB and redisplays the specified lines and antennas.
2. Nullifies the point result pointer array as new point results are to be generated.
3. Removes any existing point results for the same reason.

Dependencies:

Load(~) - Loads a DIB into the utilized DIB section.

AfxMessageBox(~) - Displays a simple message box with specified text.

GetDocument() - Returns a pointer to the active document class.

DisplayLines() - From the document class and loops through all the lines in the line object array, redrawing them.

DisplayAntennas() - From the document class and loops through all the antennas in the antenna object array, redrawing them.

NullifyPointerArray() - Sets all the pointers in the point result pointer array to NULL.

ClearPointResults() - From the document class and deletes all the point results in the point result object array.

Variables and Objects:

Bugs:

To do:

void CHIPView::TidyUpFeedBackDisplay ()

Description: This method tidies up the feedback display by removing any specified text and by resetting the process control position back to zero.

Dependencies:

SetSel(~) - Belongs to the CEdit class and sets the current edit focus on the specified line. N.B. -1 specifies entire line.

ReplaceSel(~) - Belongs to the CEdit class and changes the selected text to the specified text.

SetPos(~) - Belongs to the CProcessControl class and sets the current process control position to that specified.

GetParentFrame() - Returns a pointer to the applications Frame class.

Variables and Objects:

CEdit m_eSimSection - CEdit control associated with the CEdit box in the applications frame.

CProcessControl m_pcSimProcess - CProcessControl control associated with the CProcessControl bar in the applications frame.

Bugs:

To do:

void CHIPView::UpdateRulersInfo (int nMessage, CPoint ScrollPos, CPoint Pos = CPoint(0, 0))

Description: This method is used to set the attributes of a stRULER_INFO object which when passed back to the MainFrame class will govern the appearance and functioning of both the vertical and horizontal rulers.

Dependencies:

GetDocument() - Returns a pointer to the applications document class.

GetParentFrame() - Returns a pointer to the applications frame class.

UpdateRulersInfo(~) - Adjusts the applications rulers when either the zoom factor has changed or the drawing size is altered.

Variables and Objects:

stRULER_INFO pRulerInfo - Object containing the applications rulers information.

Double m_dPixelSize - The specified pixel size.

Bugs:

To do:

2.8 Cline

CLine::CLine (CPoint ptFrom, CPoint ptTo, COLORREF Colour)

Description: Being a class constructor, this method is called each time an object of this class is created. This method simply initialized the required object variables.

Dependencies:

Variables and Objects:

CPoint m_ptFrom - The starting point of the line.

CPoint m_ptTo - The end point of the line.

COLORREF m_Colour - The colour (or material) of the line.

Bugs:

To do:

void CLine::Draw (CDC * pDC)

Description: This method is responsible for drawing a CLine object. It does this by initially creating and selecting a pen into the active device context of the required colour. It then draws the line by moving to the lines starting point and then by drawing a line to the lines end point. The method ends by reselecting the old drawing pen into the active device context.

Dependencies:

SelectObject(~) - Selects the specified object into a device context.

MoveTo(~) - Moves to the specified point in a device context.

LineTo(~) - Draws a line to the specified point in a device context using the selected drawing object.

Variables and Objects:

CPen lpen - The newly created drawing object.

CPen* pOldPen - The old drawing object.

CDC* pDC - A pointer to the active device context.

Bugs:

To do:

void CLine::Serialize (CArchive & ar)

Description: Called when loading and saving a document, this function saves and loads an object specified variables from an archive (the specified file).

Dependencies:

IsStoring() - TRUE: Archive is storing data.

Variables and Objects:

CPoint m_ptFrom - The lines starting point.

CPoint m_ptTo - The lines ending point.

int m_MaxY - The lines maximum y-coordinate.

int m_MaxX - The lines maximum x-coordinate.

int m_MinY - The lines minimum y-coordinate.

int m_MinX - The lines minimum x-coordinate.

COLORREF m_Colour - The lines colour (or material).

Bugs:

To do:

void CLine::SetupLineVariables ()

Description: This method determines the maximum and minimum x and y coordinates of a lines starting and ending points.

Dependencies:

Variables and Objects:

CPoint m_ptFrom - The lines starting point.

CPoint m_ptTo - The lines ending point.

int m_MaxY - The lines maximum y-coordinate.

int m_MaxX - The lines maximum x-coordinate.

int m_MinY - The lines minimum y-coordinate.

int m_MinX - The lines minimum x-coordinate.

Bugs:

To do:

2.9 CMainFrame

void CMainFrame::ClearHistoryList ()

Description: When called this method deletes the entire contents of the command history list box.

Dependencies:

DeleteAllItems() - Deletes the contents of a CListCtrl.

Variables and Objects:

CListCtrl m_CommandHistory - The command history list box control object.

Bugs:

To do:

void CMainFrame::ClearInputField ()

Description: Clears the input (for drawing commands) edit box by selecting the previously entered text and then deleting it.

Dependencies:

SetSel(~) - Selects the specified line of a CEdit control (-1 for entire line).

Clear() - Deletes the current line of a CEdit control.

Variables and Objects:

CEdit m_Input - The input edit box control object.

Bugs:

To do:

void CMainFrame::ClearMaterialList ()

Description: When called this method deletes the entire contents of the materials list box.

Dependencies:

DeleteAllItems() - Deletes the contents of a CListCtrl.

Variables and Objects:

CListCtrl m_Materials - The materials list box control object.

Bugs:

To do:

BOOL CMainFrame::CreateAntennasListBox ()

Description: This methods adds a list control to the applications tool bar. It does this by initially configuring the list controls place holder (position and size) and then creating the required list control. The method concludes by inserting the required columns into the created list control and loading the list control with the required list items (antennas database).

Dependencies:

SetButtonInfo(~) - This function sets the separator's width in pixels.

GetItemRect(~) - Gets the co-ordinates of a separator you want to replace with a control.

Create(~) - Creates the required child list control.

InsertColumn(~) - Inserts the specified column into a list control.

Variables and Objects:

int nWidth - The specified width of the list control.

int nHeight - The specified heigth of the list control.

CToolBar m_wndFeedBackBar - The tool bar object into which the list control is to be inserted.

CRect rect - A rect object holding the final co-ordinates and size of the required list control.

CEdit m_eSimProcess - The actual created list control object.

Bugs:

To do:

void CMainFrame::CreateEditInput ()

Description: This method adds an edit box to the applications Input tool bar. It does this by initially configuring the edit boxes place holder (position and size) and then creating the required edit box. NOTE: This edit box takes as its input, user drawing and placement commands. It's purpose is to implement a CAD like drawing interface for end users that are more comfortable with text based drawing.

Dependencies:

SetButtonInfo(~) - This function sets the separator's width in pixels.

GetItemRect(~) - Gets the coordinates of a separator you want to replace with a control.

Create(~) - Creates the required child edit box.

Variables and Objects:

int nWidth - The specified width of the edit box.

int nHeight - The specified heigth of the edit box.

CToolBar m_wndInputBar - The tool bar object into which the edit box is to be inserted.

CRect rect - A rect object holding the final co-ordinates and size of the required edit box.

CEdit m_Input - The actual created edit box object.

Bugs:

To do:

BOOL CMainFrame::CreateHistoryListBox ()

Description: This methods adds a list control to the applications tool bar. It does this by initially configuring the list controls place holder (position and size) and then creating the required list control. The method concludes by inserting the required columns into the created list control.

Dependencies:

SetButtonInfo(~) - This function sets the separator's width in pixels.

GetItemRect(~) - Gets the co-ordinates of a separator you want to replace with a control.

Create(~) - Creates the required child list control.

InsertColumn(~) - Inserts the specified column into a list control.

Variables and Objects:

int nWidth - The specified width of the list control.

int nHeight - The specified heigth of the list control.

CToolBar m_wndFeedBackBar - The tool bar object into which the list control is to be inserted.

CRect rect - A rect object holding the final co-ordinates and size of the required list control.

CEdit m_eSimProcess - The actual created list control object.

Bugs:

To do:

BOOL CMainFrame::CreateMaterialsListBox ()

Description: This methods adds a list control to the applications tool bar. It does this by initially configuring the list controls place holder (position and size) and then creating the required list control. The method concludes by inserting the required columns into the created list control and loading the list control with the required list items (materials database).

Dependencies:

SetButtonInfo(~) - This function sets the separator's width in pixels.

GetItemRect(~) - Gets the co-ordinates of a separator you want to replace with a control.

Create(~) - Creates the required child list control.

InsertColumn(~) - Inserts the specified column into a list control.

Variables and Objects:

int nWidth - The specified width of the list control.

int nHeight - The specified heigth of the list control.

CToolBar m_wndFeedBackBar - The tool bar object into which the list control is to be inserted.

CRect rect - A rect object holding the final co-ordinates and size of the required list control.

CEdit m_eSimProcess - The actual created list control object.

Bugs:

To do:

void CMainFrame::CreateProcessPCFeedBack ()

Description: This method adds a process control to the applications tool bar. It does this by initially configuring the process controls place holder (position and size) and then creating the required process control.

Dependencies:

SetButtonInfo(~) - This function sets the separator's width in pixels.

GetItemRect(~) - Gets the coordinates of a separator you want to replace with a control.

Create(~) - Creates a smooth child progress control.

Variables and Objects:

int nWidth - The specified width of the process control.

int nHeight - The specified heigth of the process control.

CToolBar m_wndToolBar - The tool bar object into which the process control is to be inserted.

CRect rect - A rect object holding the final co-ordinates and size of the required process control.

CProcessCtrl m_pcSimProcess - The actual created process control object.

Bugs:

To do:

void CMainFrame::CreateProcessTextFeedBack ()

Description: This method adds an edit box to the applications tool bar. It does this by initially configuring the edit boxes place holder (position and size) and then creating the required edit box. NOTE: The edit box is set to read-only as it is only required to report the different stages of simulation.

Dependencies:

SetButtonInfo(~) - This function sets the separator's width in pixels.

GetItemRect(~) - Gets the coordinates of a separator you want to replace with a control.

Create(~) - Creates the required child edit box.

SetReadOnly(~) - TRUE: Associated edit box will be read-only.

Variables and Objects:

int nWidth - The specified width of the edit box.

int nHeight - The specified heigth of the edit box.

CToolBar m_wndToolBar - The tool bar object into which the edit box is to be inserted.

CRect rect - A rect object holding the final co-ordinates and size of the required edit box.

CEdit m_eSimProcess - The actual created edit box object.

Bugs:

To do:

void CMainFrame::LoadAntennasList ()

Description: When called, this method loads the antenna list box with the antennas found in the antenna record set. In more detail, it does the following:

1. Opens the antenna record set.
2. Gets the number of antennas in the antenna record set and returns if zero.
3. Loops through the antenna records in the antenna record set (last to first) and adds them (name and ID) to the antennas list box.

Dependencies:

Format(~) - This function writes formatted data to a CString.

InsertItem(~) - Inserts an item into the list view control.

SetItem(~) - Sets some or all of a list view item’s attributes.

Open() - Opens the specified database to allow access to its records.

Close() - Closes the database.

GetRecordCount() - Returns the number of records in a record set.

MovePrev() - Moves to the previous record in the database.

MoveLast() - Moves to the last record in the database.

IsBOF() - TRUE until the beginning of the record set is reached.

Variables and Objects:

int SetRecordNo - The number of antennas in the antenna record set.

LVITEM lvi - A list box item variable.

CString strItem - CString of antennas ID.

CAntennaSet set - The antennas record set.

Bugs:

To do:

void CMainFrame::LoadHistoryList ()

Description: When called, this method loads the command history list box with the commands found in the command object array. In more detail, it does the following:

1. Gets a pointer to the active document.
2. Gets the number of commands in the command object array and returns if zero.
3. Loops through the commands in the command object array (last to first) and adds them (name and position) to the command history list box.
4. Lastly it ensures that the last command entered is always visible.

Dependencies:

GetActiveDocument() - Returns a pointer to the active document.

GetCommandCount() - Returns (int) the number of commands in the command object array.

GetCommand(~) - Returns the specified command object from the command object array.
Format(~) - This function writes formatted data to a CString.

InsertItem(~) - Inserts an item into the list view control.

SetItem(~) - Sets some or all of a list view item's attributes.

GetItemCount() - Returns the number of items in a list control.

EnsureVisible(~) - Ensures that the specified item in a list control is visible.

Variables and Objects:

CHIPDoc* pDoc - A pointer to the active document class.

int CommandNumber - The number of commands in the command object array.

CCommand* pCommand - A pointer to a CCommand object used to retrieve command information.

LVITEM lvi - A list box item variable.

CString strItem - CString of commands position in command object array.

int nCount - The number of inserted list box items

Bugs:

To do:

void CMainFrame::LoadMaterialsList ()

Description: When called, this method loads the materials list box with the materials found in the materials record set. In more detail, it does the following:

1. Opens the materials record set.
2. Gets the number of materials in the materials record set and returns if zero.
3. Loops through the material records in the material record set (last to first) and adds them (name and ID) to the materials list box.

Dependencies:

Format(~) - This function writes formatted data to a CString.

InsertItem(~) - Inserts an item into the list view control.

SetItem(~) - Sets some or all of a list view item's attributes.

Open() - Opens the specified database to allow access to its records.

Close() - Closes the database.

GetRecordCount() - Returns the number of records in a record set.

MovePrev() - Moves to the previous record in the database.

MoveLast() - Moves to the last record in the database.

IsBOF() - TRUE until the beginning of the record set is reached.

Variables and Objects:

int SetRecordNo - The number of materials in the material record set.

LVITEM lvi - A list box item variable.

CString strItem - CString of materials ID.

CMaterialSet set - The materials record set.

Bugs:

To do:

BOOL CMainFrame::OnCreateClient (LPCREATESTRUCT lpcs, CCreateContext * pContext) [protected, virtual]

Description: This method is called each time the client area of the applications frame is created. It was changed to display both the required vertical and horizontal rulers.

Dependencies:

CreateRulers(~) - Creates the required rulers.

Variables and Objects:

Bugs:

To do:

void CMainFrame::OnDisplayBar () [protected]

Description: This method toggles the visibility of the Display tool bar by doing the following:

1. Gets the state of the Display tool bar.
2. If it is visible, it hides it and vise versa.
3. Reshuffles the frame layout.

Dependencies:

GetStyle() - This method gets the styles currently applied to a toolbar control.

ShowControlBar(~) - Method used to both show and hide a tool bar.

RecalcLayout() - Usually called by frame work the reshuffle the fame window layout to maximize the available space when showing/hiding controls or resizing the entire frame.

Variables and Objects:

BOOL bVisiable - TRUE: Tool bar visible.

Bugs:

To do:

void CMainFrame::OnEnSelect ()

Description: This method is called each time the contents of the input edit field in the applications frame is changed. It does the following:

1. Gets a pointer to the active view.
2. Retrieves the CString from the input edit field.

3. Checks that the terminating character ']' has not yet been entered.
4. Makes all string characters lowercase.
5. Removes any white space before string.
6. If the terminating character has been entered, the formatted input CString is passed to the NewCommand method in the view class and the edit input field is cleared before the view is redrawn.

Dependencies:

GetActiveView() - Returns a pointer to the active view.

LineLength(~) - Retrieves a character index for a given line number.

GetBuffer(~) - This method retrieves a pointer to the internal character buffer for the CString object.

GetLine(~) - Retrieves the specified line of text from an edit control.

Find(~) - Returns the index of the specified character in a CString.

MakeLower() - Makes all the characters in the associated CString lower case.

TrimLeft() - Removes any white space from the beginning of the associated CString.

NewCommand(~) - Handles the input of a new command.

ClearInputField() - Clears the input edit field.

ReDraw() - Invalidates and redraws to display to show any changes resulting from the entered command.

Variables and Objects:

CString m_strInput - The current user inputted CString.

CHIPView* pView - A Pointer to the active view.

CEdit m_Input - The input edit control object.

int iExit - -1: Terminating character not found in CString.

Bugs:

To do:

void CMainFrame::OnInputBar () [protected]

Description: This method toggles the visibility of the Input tool bar by doing the following:

1. Gets the state of the Input tool bar.
2. If it is visible, it hides it and vise versa.
3. Reshuffles the frame layout.

Dependencies:

GetStyle() - This method gets the styles currently applied to a toolbar control.

ShowControlBar(~) - Method used to both show and hide a tool bar.

RecalcLayout() - Usually called by frame work the reshuffle the fame window layout to maximize the available space when showing/hiding controls or resizing the entire frame.

Variables and Objects:

BOOL bVisiable - TRUE: Tool bar visible.

Bugs:

To do:

void CMainFrame::OnMaterialSelect ()

Description: This method is called each time the selection in the materials list box (in the applications frame) is changed. It initially gets a pointer to the active document class and a double representation of the selected list item (POSITION->CString-> Double). It then

opens the materials record set and returns if it is empty. If it is not empty, it loops through the record set until the position of the selected material is reached. It then updates the current selected material indicator in the application status bar and sets the current drawing colour in the document class to that of the selected material.

Dependencies:

GetActiveDocument() - Returns a pointer to the active document.

GetFirstSelectedItemPosition() - Returns (POSITION) the position of the selected list box item.

Format(~) - This function writes formatted data to a CString. In this method it is used to get a CString representation of a POSITION variable.

atof(~) - Does a CString To double conversion.

Open() - Opens the specified database to allow access to its records.

Close() - Closes the database.

GetRecordCount() - Returns the number of records in a record set.

MoveNext() - Moves to the next record in the database.

MoveFirst() - Moves to the first record in the database.

RGB(~) - Returns the specified COLORREF variable.

Variables and Objects:

CHIPDoc* pDoc - Pointer to active document class.

POSITION ListPosition - Value used to denote position of record in list view control.

CString positionCS - CString representation of ListPosition.

double positionD - Double representation of ListPosition.

CMaterialSet set - Object of type MaterialSet (container class for database records).

Bugs:

To do:

void CMainFrame::OnRemoveCommand ()

Description: This method is responsible for removing a command from the command history list box and its associated action. It does this as follows:

1. Gets a pointer to the active document class.
2. Gets a pointer to the active view class.
3. Gets the number of commands in the command history list box and returns if zero.
4. Gets the position of the selected command and obtains a CString and then double representation of it.
5. It then prompts if the user is sure they want to delete the command and if they press Ok, the deletion process begins.
6. The next two steps involve looping through the commands in the command object array and determining how many "antenna" and "lineto" commands have been specified up to and including the specified command awaiting deletion. This is used to determine the index of the line/antenna object to delete if the specified command resulted in a line/antenna being drawn.
7. Determines if the selected command resulted in a line/ antenna being drawn and deletes the required line/ antenna from the respective object array.
8. Deletes the selected command from the command object array and clears/reloads the command history list box to show change.
9. The final stage involves clearing the backing DIB and redrawing the required information on the DIB.

Dependencies:

GetActiveDocument() - Returns a pointer to the active document.

GetActiveView() - Returns a pointer to the active view.

GetCommandCount() - Returns (int) the number of commands in the command object array.

GetFirstSelectedItemPosition() - Returns (POSITION) the position if the selected list box item.

Format(~) - This function writes formatted data to a CString. In this method it is used to get a CString representation of a POSITION variable.

atof(~) - Does a CString To double conversion.

MessageBox(~) - Displays the specified text in a simple message box.

GetCommand(~) - Returns the specified command object from the command object array.

Find(~) - Returns the index of the specified character in a CString.

Left(~) - Returns the CString to the left of the specified index in another CString.

DeleteLine(~) - Deletes the specified line object from the line object array.

DeleteCommand(~) - Deletes the specified command object from the command object array.

DeleteAntenna(~) - Deletes the specified antenna object from the antenna object array.

ClearHistoryList() - Deletes the contents of the command history list box.

LoadHistoryList() - Reloads the command history list box with the command objects found in the command object array.

ClearBackGround() - Selects a blank DIB into the active device context.

DisplayLines() - Draws the lines in the line object array on the backing DIB.

DisplayAntennas() - Draws the antennas in the antenna object array on the backing DIB.

NullifyPointerArray() - Sets all the pointers in the point result pointer array to NULL.

ReDraw() - Invalidates and redraws to display to show any changes resulting from the deletion.

Variables and Objects:

CHIPDoc* pDoc - Pointer to active document class.

CHIPView* pView - Pointer to active view class.

int ComCount - The number of commands in the command object array.

POSITION ListPosition - Value used to denote position of record in list view control.

CString positionCS - CString representation of ListPosition.

double positionD - Double representation of ListPosition.

int LinePos - The number of line related commands before and including the selected command.

int AntPos - The number of antenna related commands before and including the selected command.

Bugs:

To do:

void CMainFrame::OnUpdateDisplayBar (CCmdUI * pCmdUI) [protected]

Description: This method is primarily responsible for checking the Display bar menu item when it is visible and un-checking the Display bar menu item when it is not.

Dependencies:

GetStyle() - This method gets the styles currently applied to a toolbar control.

SetCheck(~) - Used to set a user-interface item to the required check state.

Variables and Objects:

Bugs:

To do:

void CMainFrame::OnUpdateInputBar (CCmdUI * pCmdUI) [protected]

Description: This method is primarily responsible for checking the Input bar menu item when it is visible and un-checking the Input bar menu item when it is not.

Dependencies:

GetStyle() - This method gets the styles currently applied to a toolbar control.

SetCheck(~) - Used to set a user-interface item to the required check state.

Variables and Objects:

Bugs:

To do:

void CMainFrame::ShowRulers (BOOL bShow)

Description: This method shows and hides the applications rulers.

Dependencies:

ShowRulers(~) - When TRUE is passed in, the applications rulers are shown.

Variables and Objects:

Bugs:

To do:

void CMainFrame::UpdateRulersInfo (stRULER_INFO stRulerInfo)

Description: This method updates the current ruler information.

Dependencies:

UpdateRulersInfo(~) - Updates the applications rules with the information contained in the stRULER_INFO object.

Variables and Objects:

Bugs:

To do:

2.10 CMeasurementResult

CMeasurementResult::CMeasurementResult (CPoint Position, double Magnitude, int ID)

Description: Called as a constructor when an object of this type is created. This method simply initializes the objects required variables.

Dependencies:

Format(~) - This function writes formatted data to a CString. In this method it is used to get CString representations of both integer and double variables.

Variables and Objects:

CPoint m_cpPosition - The position of the specified measurement result in the raster.

Double m_dMagnitude - The magnitude of the specified measurement result.

CString m_csID - The CString of the measurement results ID.

CString m_csMagnitude - The CString of the measurement results specified magnitude.

BOOL m_bResolved - TRUE: Measurement result has been resolved.

Bugs:

To do:

void CMeasurementResult::Draw (CDC * pDC)

Description: This method displays a measurement result on the screen. The measurement result is displayed (at its specified position) as text which contains both its ID and its specified magnitude. This method only displays unresolved measurement results.

Dependencies:

SetTextColor(~) - Sets the required colour of the text to be output.

SetTextAlign(~) - Sets the alignment of the text to be output.

TextOut(~) - Outputs (draws on device context) the specified text at the specified point in the raster.

Variables and Objects:

CDC* pDC - A pointer to the active device context.

BOOL m_bResolved - TRUE: Measurement result has been resolved.

Bugs:

To do:

void CMeasurementResult::Serialize (CArchive & ar)

Description: Called when loading and saving a document, this function saves and loads an object specified variables from an archive (the specified file).

Dependencies:

IsStoring() - TRUE: Archive is storing data.

Variables and Objects:

CPoint m_cpPosition - The position of the specified measurement result in the raster.

Double m_dMagnitude - The magnitude of the specified measurement result.

CString m_csID - The CString of the measurement results ID.

CString m_csMagnitude - The CString of the measurement results specified magnitude.

BOOL m_bResolved - TRUE: Measurement result has been resolved.

Bugs:

To do:

2.11 CPointResult

CPointResult::CPointResult (CPoint Position, double m_dmagnitude, BOOL m_borigional)

Description: Called as a constructor when an object of this type is created. This method simply initializes the objects required variables.

Dependencies:

Variables and Objects:

CPoint m_Position - The position of the point result in the raster.

Double m_dMagnitude - The magnitude of the point result.

BOOL m_bOrigional - TRUE: The point result was created by simulation and not averaging.

Bugs:

To do:

void CPointResult::Draw (CDC * pDC)

Description: When this function is called, it changes the on-screen pixel colour at the point results position to a colour related to its magnitude.

Dependencies:

SetPixel(~) - Sets the specified pixel of a device context to the specified colour.

GetPointResultColour() - Returns a colour (COLORREF) that corresponds to the point results magnitude.

Variables and Objects:

CDC* pDC - A pointer to the active device context.

Bugs:

To do:

COLORREF CPointResult::GetPointResultColor ()

Description: This method returns the colour representation of a point results magnitude. It does this by using the point results magnitude in a linear COLORREF scale which goes from pink to red to blue to white.

Dependencies:

RGB(~) - Specifies a COLORREF variable.

Variables and Objects:

Double m_dMagnitude - A positive representation of the point results magnitude.

Bugs:

To do:

2.12 DrawingSettingsDLG

void DrawingSettingsDLG::OnOK () [protected, virtual]

Description: Called when the user closes the Drawing Setting dialog using the Ok button. All that this method does is update its member variables with the user inputted ones.

Dependencies:

UpdateData(~) - By passing in TRUE, the entered variables are copied to their respective member variables.

OnOK() - Simply closes the dialog.

Variables and Objects:

Bugs:

To do:

2.13 MaterialDlg

CString MaterialDlg::DoubleToCString (double in)

Description: This method takes as its argument a double value and returns a CString representation.

Dependencies:

Format(~) - This method writes formatted string data into a string. In this method, it is used to convert a double variable to a CString.

Variables and Objects:

CString strFinalVal - CString variable to hold conversion.

Double in - The methods input argument.

Bugs:

To do:

void MaterialDlg::LoadList ()

Description: This method loads the materials list box with the required information from the materials database. It does this by initially opening the materials database and checking that it contains data. After it has done this, it moves to the last record in the materials record set and steps through them (starts at the end as list box is loaded bottom up) populating the required column of the list box with the required information from the materials database. The following information is extracted and displayed:

1. Material ID.
2. Material Name.
3. Material Reflection Coefficient.
4. Material Transmission Coefficient.
5. Material Frequency.

Dependencies:

Format(~) - This method writes formatted string data into a string.

Open() - Opens the specified database to allow access to its records.

MoveLast() - Moves to the last record of the database.

MovePrev() - Moves to the previous record in the database.

Close() - Closes the database.

AfxMessageBox(~) - Simply displays the specified text in a message box.

IsBOF() - TRUE until the beginning of the record set is reached.

GetRecordCount() - Returns the number of records in a record set.

InsertItem(~) - Inserts an item into the list view control.

SetItem(~) - Sets some or all of a list view item's attributes.

DoubleToCString(~) - Returns the CString of the specified double.

Variables and Objects:

CMaterialSet set - Object of type MaterialSet (container class for database records).

LVITEM lvi - List Box item object.

CString strItem - Used to hold material attributes before they are inserted into the materials list box.

Bugs:

To do:

void MaterialDlg::OnAddMaterial () [protected]

Description: This method displays the Add Material dialog window and on closing the dialog, clears and reloads the materials list box.

Dependencies:

DoModal() - Displays a modal dialog box.

DeleteAllItems() - This function deletes all the items from a list view control.

LoadList() - This function reloads the list view control with the newly edited database records.

Variables and Objects:

AddMaterialDlg dialog - Add Material dialog object used to display the Add Material dialog.

Bugs:

To do:

void MaterialDlg::OnChangeCoeff () [protected]

Description: This method is called in response to the user clicking the Change Transmission Coeff. in the Materials Menu. This method does the following:

1. Get the position of the selected item in the list.
2. Convert this position to a CString and then a double.
3. Check that an item in the list was selected.
4. Open the materials database.
5. Get the ID of the last record in the database.
6. Move to the selected record.
7. Open the Change Transmission Coefficient dialog and when dismissed with Ok, get the new Transmission coefficient, generate the required reflection coefficient, change the existing coefficients, close the database and tidy up any variables used.

Dependencies:

GetFirstSelectedItemPosition() - This method gets the position of the first selected item in the list view control.

atof(~) - Convert specified character string to a double-precision floating-point number.

Format(~) - This method writes formatted string data into a string. In this method, it is used to convert a POSITION variable to a CString.

Open() - Opens the specified database to allow access to its records.

MoveFirst() - Moves to the first record of the database.

MoveNext() - Moves to the next record in the database.

DoModal() - This method invokes a modal dialog box and returns the dialog-box result when done.

Edit() - This function allows for changes to the current record.

Update() - Required to complete Edit() function.

Requery() - Rebuilds the record set.

Close() - Closes the database.

DeleteAllItems() - This function deletes all the items from the list view control.

LoadList() - This function reloads the list view control with the newly edited database records.

AfxMessageBox(~) - Simply displays the specified text in a - message box.

Variables and Objects:

POSITION ListPosition - Value used to denote position of record in list view control.

CString positionCS - CString representation of ListPosition.

double positionD - Double representation of ListPosition.

CMaterialSet set - Object of type MaterialSet (container class for database records).

ChangeTransCoeffDlg dlg - Object of type ChangeTransCoeffDlg used to interact with the Change Transmission Coefficient dialog.

Bugs:

To do:

BOOL MaterialDlg::OnInitDialog () [protected, virtual]

Description: This method is called each time the Materials Dialog is initialized. It basically sets up the required columns in the materials list box and loads the materials list box with the required data from the materials database.

Dependencies:

GetClientRect(~) - Returns (CRect) a rectangle which comprises the materials list box.

Width() - Returns (int) the width of a CRect object.

InsertColumn(~) - Inserts the specified column into a list box control.

LoadList() - Loads the list control with the data from the materials database.

Variables and Objects:

CRect rect - Material list control rectangle.

int nColInterval - The width of each added column in the material list.

Bugs:

To do:

void MaterialDlg::OnRemoveMaterial () [protected]

Description: Allows for the deletion of a material from the database. This is done by obtaining the user selected material ID from the list control. Having done this, the corresponding material and its data is removed from the material database. The subsequent materials in the material database will then get their ID values reduced by 1. The user is prompted if he/she is sure that they want to remove the material. This method also updates the list control to show any changes after a deletion. In detail:

1. Gets the position of the selected material
2. Gets a CString and then double representation of the selected items position.
3. CHECK 1: Was a material selected.
4. Opens the materials record set and confirms the deletion of a material.
5. Loops through the record set until it finds the selected record. When it finds the record, it deletes the record and then runs through the remainder of the records and decrements their ID values.
6. The method ends by setting the position variable to 0 or no material selected (else a double click will remove two materials), clears the material list box and then reloads it.

Dependencies:

GetFirstSelectedItemPosition() - This method gets the position of the first selected item in the list view control.

atof(~) - Convert specified character string to a double-precision floating-point number.

Format(~) - This method writes formatted string data into a string. In this method, it is used to convert a POSITION variable to a CString.

Open() - Opens the specified database to allow access to its records.

MoveFirst() - Moves to the first record of the database.

MoveNext() - Moves to the next record in the database.

DoModal() - This method invokes a modal dialog box and returns the dialog-box result when done.

Edit() - This function allows for changes to the current record.

Delete() - Deletes the current record of a record set.

Update() - Required to complete Edit() function.

Requery() - Rebuilds the record set.

Close() - Closes the database.

DeleteAllItems() - This function deletes all the items from the list view control.

LoadList() - This function reloads the list view control with the newly edited database records.

AfxMessageBox(~) - Simply displays the specified text in a message box.

IsEOF() - TRUE until the end of the record set is reached.

Variables and Objects:

POSITION ListPosition - Value used to denote position of record in list view control.

CString positionCS - CString representation of ListPosition.

double positionD - Double representation of ListPosition.

CMaterialSet set - Object of type MaterialSet (container class for database records).

Bugs:

To do:

2.14 PlaceExtraDlg

void PlaceExtraDlg::OnAntenna () [protected]

Description: This method is called each time the Antenna check box is checked and simple un-checks the Measurement check box.

Dependencies:

SetCheck(~) - Belongs to the CButton class and sets the state of the check box i.e. 0 = unchecked, 1 = checked, 2 = undetermined.

Variables and Objects:

Bugs:

To do:

BOOL PlaceExtraDlg::OnInitDialog () [protected, virtual]

Description: This method is called each time the Place Extra dialog is initialized. It initially creates an object of type **CAntennaSet** to gain access to the Antennas database. It then moves to the first antenna record and then loops through all the antenna records, populating the antenna list box (m_Antenna_List) with the names of all the antennas currently in the applications Antennas Database.

Dependencies:

Open() - Opens the specified database to allow access to its records.

MoveFirst() - Moves to the first record of the database.

MoveNext() - Moves to the next record in the database.

Close() - Closes the database.

IsEOF() - TRUE until the end of the record set is reached.

SetCurSel(~) - Sets the focus on the specified combo box item.

AddString(~) - Adds the specified string to combo box.

Variables and Objects:

CAntennaSet set - Antenna database record set object.

BOOL m_bMeasurementOK - TRUE: Valid measurement specified.

BOOL m_bAntennaOK - TRUE: Valid antenna specified.

Bugs:

To do:

void PlaceExtraDlg::OnMeasurement () [protected]

Description: This method is called each time the Measurement check box is checked and simple un-checks the Antenna check box.

Dependencies:

SetCheck(~) - Belongs to the CButton class and sets the state of the check box i.e. 0 = unchecked, 1 = checked, 2 = undetermined.

Variables and Objects:

Bugs:

To do:

void PlaceExtraDlg::OnOK () [protected, virtual]

Description: Called each time the Place Extra dialog is dismissed with Ok button. This method initially checks to see which check box was selected. If the Antenna check box was selected, the selected antenna name is found and the dialog closed. If the Measurement check box was checked, the specified measurement result is validated top ensure that it lies within the

Dependencies:

GetCheck() - Belongs to the CButton class and returns the state of button i.e. 0 = unchecked, 1 = checked, 2 = undetermined.

GetCurSel() - Returns the zero-based index of the currently selected item, if any, in a single selection list box.

GetLBText(~) - Return the CString at the specified index of a list box.

UpdateData(~) - By passing in TRUE, the entered variables are copied to their respective member variables.

OnOK() - Simply closes the dialog.

AfxMessageBox(~) - Simply displays the specified text in a message box.

Variables and Objects:

BOOL m_bAntennaOK - TRUE: Antenna correctly selected.

BOOL m_bMeasurementOK - TRUE: Measurement correctly selected and entered.

Double m_dLowerMeasLimit - The simulated magnitude at the selected point.

Double m_dSimCutOffLevel - The specified simulation cut-off magnitude level.

Bugs:

To do:

2.15 SimulationSettingDlg

void SimulationSettingDlg::OnFromDatabase () [protected]

Description: Called when the user checks the from database check box. All this method does is uncheck the measurement box and set its corresponding variable to FALSE.

Dependencies:

SetCheck() - Belongs to the CButton class and sets the state of the check box i.e. 0 = unchecked, 1 = checked, 2 = undetermined.

Variables and Objects:

BOOL m_bMeasurement - TRUE: Use measurements in simulation mechanism.

Bugs:

To do:

BOOL SimulationSettingDlg::OnInitDialog () [protected, virtual]

Description: This method is called each time the Simulation Settings dialog is initialized. It initially checks the state of the two simulation mechanism governing variables and checks the respective check boxes. It also sets the focus of the frequency selection combo box to its first entry.

Dependencies:

SetCheck(~) - Belongs to the CButton class and sets the state of the check box i.e. 0 = unchecked, 1 = checked, 2 = undetermined.

SetCurSel(~) - Sets the focus on the specified combo box item.

Variables and Objects:

BOOL m_bDataBase - TRUE: Use database values in simulation mechanism.

BOOL m_bMeasurement - TRUE: Use measurements in simulation mechanism.

Bugs:

To do:

void SimulationSettingDlg::OnMeasurement () [protected]

Description: Called when the user checks the measurement check box. All this method does is uncheck the From Database box and sets its corresponding variable to FALSE;

Dependencies:

SetCheck() - Belongs to the CButton class and sets the state of the check box i.e. 0 = unchecked, 1 = checked, 2 = undetermined.

Variables and Objects:

BOOL m_bDataBase - TRUE: Use database values in simulation mechanism.

Bugs:

To do:

void SimulationSettingDlg::OnOK () [protected, virtual]

Description: Called when the user closes the Simulation Setting dialog using the Ok button. All that this method does is update its member variables with the user inputted ones. It also checks which check button (Calculation mechanism) is checked and sets the m_bMeasurement or m_bDataBase variables as required. This method also ensures that the entered simulation cut-off level is between 0 and 250 as well as ensuring that entered antenna loss at 1m is less than zero.

Dependencies:

UpdateData(~) - By passing in TRUE, the entered variables are copied to their respective member variables.

GetCheck() - Belongs to the CButton class and returns the state of button i.e. 0 = unchecked, 1 = checked, 2 = undetermined.

OnOK() - Simply closes the dialog.

GetLBText(~) - Return the CString at the specified index of a list box.

Variables and Objects:

BOOL m_bMeasurement - TRUE: Measurement selected. FALSE: From Database selected.

Bugs:

To do:

3. Conclusion

From Section 2 it is evident that the applications code is well documented. This will ultimately simplify any maintenance, updating or expanding the application might require by persons unfamiliar with it.

Appendix E:
Measurement Data and Example Output

1. Introduction

In this Document, the measured results for the test conducted to determine an objects attenuation factor's dependence on the incoming wave's angle of arrival are presented. Sample simulation outputs are also shown and discussed.

2. Object attenuation factor as a function of the angle of arrival

To model the naturally occurring phenomenon in which obliquely incident waves transmit less power than waves that occurs at more normal incidences on objects, the following experiment was conducted:

1. Two antennas with know parameters were set-up a set distance apart in an anechoic chamber.
2. The path loss between the transmitting and receiving antenna was measured.
3. A large piece of tinted glass was then placed between the two antennas at various angles, and for each angle the corresponding path loss measured, see Table 4.

Table 4: Measured loss due to a piece of tinted glass at various angles between two antennas.

Angle of incidence with respect to the normal (degrees)	Loss due to tinted glass (dB)
0	-14.8
5	-15.1
10	-14.7
15	-14.1
20	-15.4
25	-13.6
30	-15.8
35	-18.6
40	-19.0
45	-20.7
50	-22.1
55	-24.0
60	-28.3
65	-34.0
70	-39.4
75	-51.0
80	-69.3

These values were compared to the approximation (Loss at normal incidence)/cos(θ)) where θ is the angle of incident with respect to the normal and found to be similar, see Figure 2. The measurement results deviated more from the approximation at large incidence angles as due to the limited size of the tinted glass and the resulting edge diffraction effects.

3. Example Output

An example application output is shown in Figure 16 (EM Lab at the University of the Witwatersrand) which was generated using on-site measurement data. This is evident by the fact that signals are present inside the shielded chamber which result from the application down-playing the effects of high attenuation objects due to the presence of the indirect signal path contributions in the taken measurements.

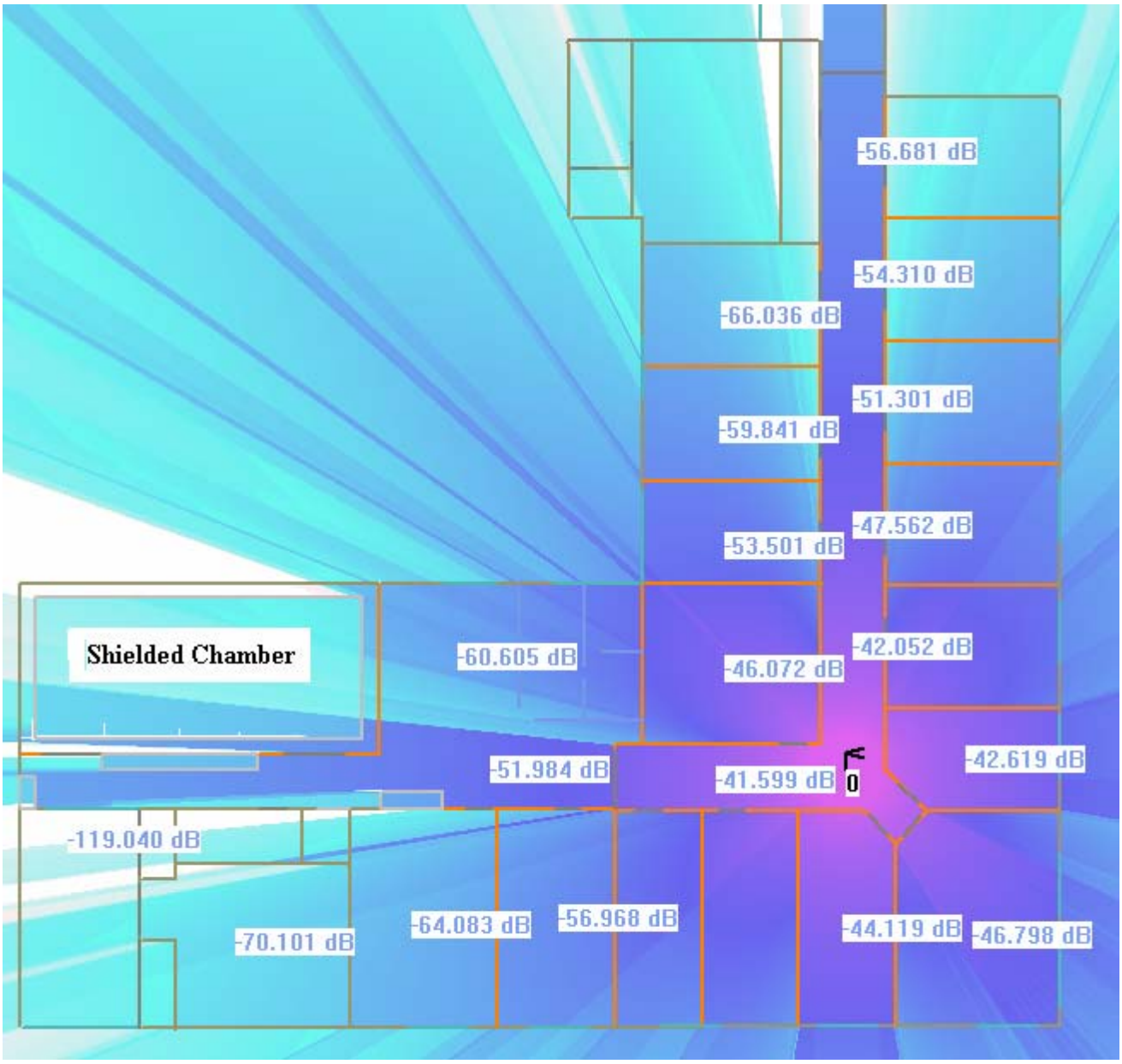

Figure 16: An Example application output.

Figure 17 contains another example application output which was generated for a small office building using the advanced log distance model and on-site measurement data. Of interest is the overall low attenuation rate resulting from the use of low attenuation construction materials as well as building layout. In such buildings the placement of the required access point is of no consequence.

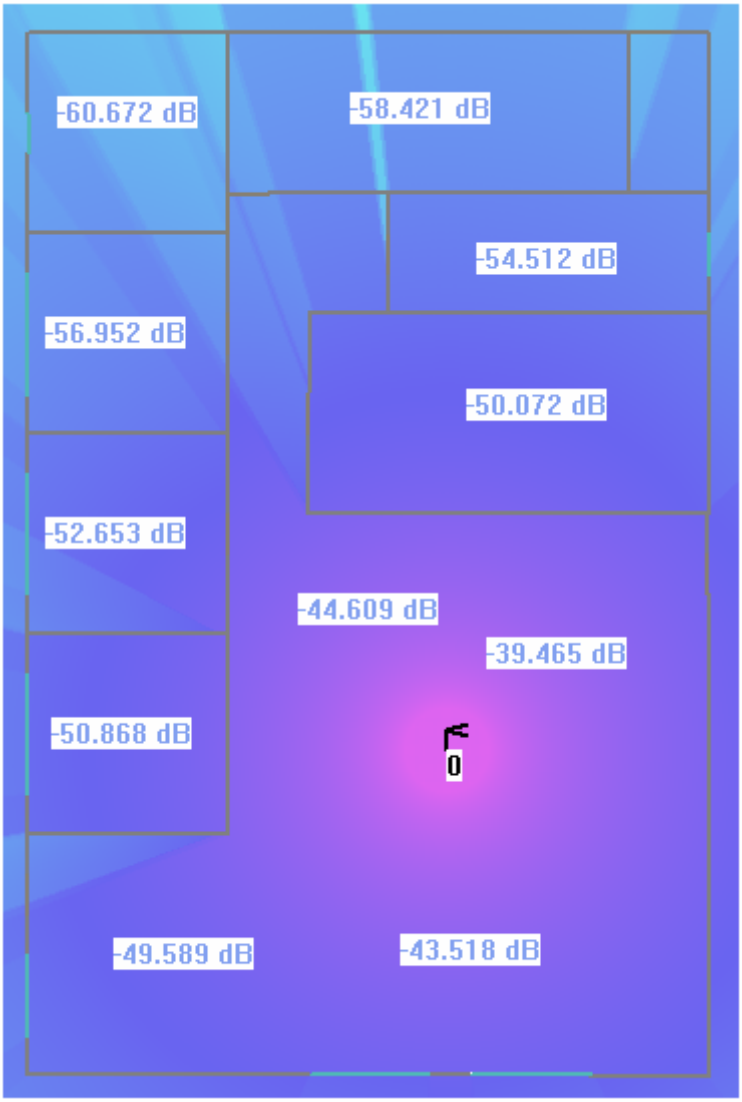

Figure 17: An Example application output

4. Conclusion

Tests results for the verification of an objects attenuation factor's dependence on the incoming wave's angle of arrival were presented and discussed. Example application outputs when using on-site measurements to determine building material attenuation factors were also shown.

Appendix F:
Source Code

Contained within the attached compact disc is the entire code listing for the developed WLAN propagation prediction tool, as well as an executable and the associated files (example materials database etc.) required to run it.