# BTech 450

# Final Report


Zhigang Yin

# Abstract

BTech 450 DT project is a one-year project. Given some fairly specific problems, we will design and implement a solution. The problem should be challenging enough to be interesting and motivating.

I consider the project that I have chosen is fairly challenging and interesting. It greatly improved my research and self-learning ability. There is a general goal for the project but no specific instruction telling how to get the problem solved, so there's a lot of space left for me to explore and find all sorts of alternatives to the final solution. By comparing each alternatives, I will then discover not necessarily the best but at least a comparatively good solution.

This end of year report illustrates and summarizes the work I have done in year 2003 (March ~ November) for the Project in Information Technology at Auckland University. Related documentations mentioned in the report are listed as appendices.

# Acknowledgment

Although this is a single-person project, many people gave me great help and guidance on different aspects throughout the entire process of the project.

First of all, I must thank my supervisor, Mr Grant Black. He gave me perfect guidance throughout the project, also provided me with lots of very helpful and updated information that are related to this project.

Secondly, with many thanks to the software engineering group of the ITL company. As I was working as a member in the group, I received help from each of them at various stages.

Thirdly, I want to thank *Dr. S Manoharan* for spending time explaining to me important skills and possible formats for research and report writing.

It was a great experience to be given the chance to work in a real software company, and to feel what a real working environment looks like.

# Table of Contents

# 1. Introduction

## 1.1 Company Introduction

***Information Tools Ltd* (ITL)**

Established in 1989, Information Tools' investigative software and services are now in use in over 1000 sites spanning over 100 countries worldwide.

The company's headquarters are located close to the beach in Milford, on the North Shore of Auckland, New Zealand.

**ITL** specialise in the development of software tools and database services for marketing. The software can be used with most forms of marketing data and includes market research, sales data, media research including GRPs, advertising expenditure, retail audit, Customer databases, in fact virtually any data needed for marketing.

Client organisations comprise major multinational companies, strategic planning groups, Universities, Financial Institutions, Government organisations, the Media, business consultants and Market Research Agencies.

ITL produces various types of analysing software to provide database services to their clients according to their needs.
For example,

- **"ESPRI"** service involves designing and converting data into a form that makes it simple for organisations to investigate that data using the ESPRI program.

- "**HARMONI**" is for organisations that want to integrate data from different sources, both internal and external.

To learn more about their services and products, please visit their website at http://www.infotools.com/

## 1.2 Project Introduction

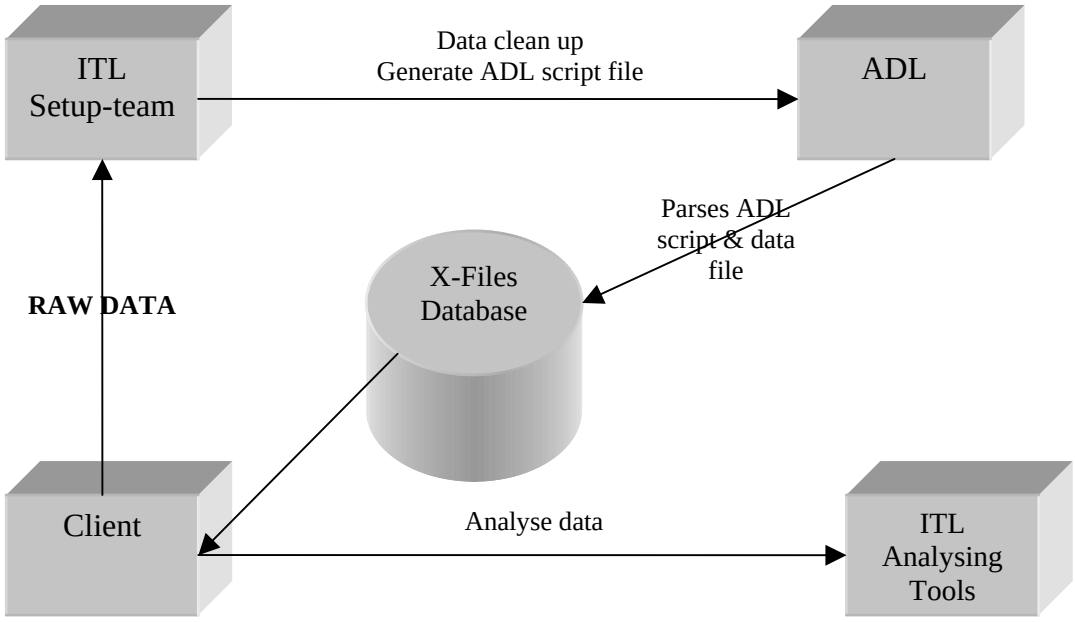
Project title: *Improvements to tools and processes in the creation of Information Tools Ltd (ITL) databases*

Currently around 50% of the company's Auckland office staff (25 or more) is employed to work in a task described as "data setup".

Generally speaking, the setup-team task is that they accept raw data (primarily survey data) in a range of formats and then use *ADL* and supporting programs from the '*iTools*' suite to produce proprietary formatted ITL databases.

The primary tool of the data set-up team is considered to be the ADL program, as at present that is the only program that can create the ITL inverse formatted databases for end-users. Other companies also use the ADL program worldwide, to produce databases for client analysis.

Following diagram shows the workflow of the setup-team task and the role of ITL's main products, (ADL, X-Files Database, and Analysing Tools) in the workflow.




Since the ADL program is a fundamental part of the Information Tools business model, ITL seeks to develop better, more industry standard ways in which the setup task could be done. Through this ITL hopes to improve not only the ADL program itself, but also the production task in which they process data and build databases for customer analysis.

See more detailed description on ADL in Appendix A: *What is ADL?*

# 2. Project Description

## 2.1 Limitations of Existing ADL

There are several issues with the existing ADL language and program:

- ADL scripts are difficult to automate. As ADL scripts are very flexible and the language not well defined, human generated scripts can be very difficult to machine parse. ADL scripts designed to be read and written by setup *people* and not automated tools. This makes it difficult to automate production processes.
- The ADL language is unusual – the closest equivalent language maybe the AWK scripting language, but it is sufficiently different from common programming/scripting languages that finding new employees with ADL or directly related skills is extremely difficult. This means that all new setup staffs require extensive training in order to be productive.
- The language (syntax and semantics) is not consistent, thus ADL has a steep learning curve. For instance, there are multiple ways of creating an Axis[1].

The ADL language and program was largely designed for performing ad-hoc jobs, allowing great flexibility in turning textual data into ITL databases. It was not designed for dealing with (the increasingly common) 'trackers' - large regular data streams that remain largely unchanged in format between waves. The setup team currently works around many of the limitations of ADL by utilizing a range of ad-hoc pre-processing tools such as Grid, MSM2ADL or 'CokeV', which manipulate data into a form that ADL can handle. Post-processing tools such as Shrink/Merge/AddAxes also often modify the databases output by ADL, in order to build complete databases for clients.

## 2.2 Project Requirements

The requirements are from two main sources: the programmers and the Users.

The programmers are standing on the conceptual level, aiming to improve ADL based on their programming experience. Each of the programmers is experienced and can program in more than one programming language, so they can clearly see the advantages of ADL in processing raw data files and in generating databases. At the mean time, by comparing the algorithm and features with other languages, they can see exactly the weakness of current version of ADL.

[1] *The fundamental building block of an Information Tools formatted database. An axis usually represents one question in a questionnaire*.

*--Conceptual requirements:*

**1) Training**
**2) Speed**
**3) Correctness**
**4) Easy to maintain and extend to meet new challenges.**
- Flexibility
- Integration

**5) Retain backwards and forwards compatibility.**
**6) Extra features:**

On the other hand, current ADL users are not necessarily experienced in programming. They might be hired and trained to use only ADL language, but have no idea of what programming languages are like, for example, JAVA, C++, and VB. However, they can still find inconvenient bits while using ADL syntax to generate databases. These are reflected in the feedbacks from users to the programmers.

*--Practical requirements from ADL users:*

- **Error handling**
- **Calculation**
- **Security**
- **Variable types support**
- **Other features**

In the document "*ENHANCED ADL DESIGN DOCUMEN*", all the requirements, both conceptual and practical requirements, are listed and discussed in detail.

## 2.3 Project Objectives

There are several objectives with the project, these objectives are general instructions that will help me, guide me through the stages and finally achieve the goal of this project.

| | **Objective** | **Tasks** |
|---|---|---|
| 1 | Training | ADL and database setup training |
| 2 | ADL language documentation | Understanding the current structure of language, and the typical use of the language. Code walk-through. Producing a comprehensive document on ADL. |

| | | |
|---|---|---|
| 3 | Designing an enhanced ADL language | Analysing the requirements for the design, and exploring alternatives. Documenting the impact of the possible designs on other existing tools that the designs need to co-operate with. Producing a final design document and a language user-guide. |
| 4 | Implementation plan for the enhanced language | Formulating the implications of implementing the design choice(s). Producing a document describing the implementation proposal. |
| 5 | Program development | Choosing the appropriate programming environment. Program development. Testing and debugging. Code documentation. |
| 6 | Final report and presentation | Prepare presentation and report |

## 2.4 Status of completion

Along with each objectives of the project, there is certain form of measurement of completion. These measurements/ milestones are listed in the following table. Some of them were finished in the first semester, and the rest were done during the semester break and the second semester of this year.

| | **Objective** | **Milestone** | **Period** | **Status** |
|---|---|---|---|---|
| 1 | Training | Ability to read and write ADL scripts | March | Completed |
| 2 | ADL language documentation | A comprehensive document describing ADL | April | Completed |
| 3 | Designing an enhanced ADL language | A design document and a user-guide for the enhanced language | May | Completed |
| 4 | Implementation plan for the enhanced language | An implementation specification | June | Completed |
| 5 | Program development | Application program and code documentation | July ~ September | Completed |
| 6 | Final report and presentation | Report and presentation | October | Completed |

# 3. Research & Learning

## 3.1 ADL & Analysis Applications

### 3.1.1 ADL

Before start working on the project, it is necessary to familiarize myself with the ITL working environment, including the workflow, the analysing software, the ITL Database format and most importantly, ADL.

I was given 40 hours or more on training of how to write ADL script files and "compiling" the script file with ADL programme, also on how to read and extract useful information by using the analysing tools.

I have been provided a CD containing all the information I possibly need for the project: Analysis Application, latest version of ADL programme, ADL manual (softcopy and hardcopy), sample data files, and the source code of the ADL program.

Even though the training session was finished, I must still spend at least 2 hours a week on writing ADL script files and using analysis application, as I've noticed the importance of continuous practice for programming language. There are two components required for running ADL: data file and the ADL script file.

The ADL script files have "ADL" extension, and generally have the same file name as the data file for simplicity and clarity; however the data file and ADL file can have different names and will be parsed by the ADL programme with no problem. For example, if the data matrix below is stored in a file called "test.dat", the ADL script usually is named "test.adl" in order to read in data from the data file. Theoretically, the data file can have any extension, as long as its content is in form of data matrix.

During the course of this project, data files are basically test data made up by myself. They usually contain small digit matrix typically look like:

1, 1, 2, 3, 4
2, 0, 1, 5, 3
2, 1, 5, 1, 6
0, 0, 0, 1, 4

For the above DAT file my ADL script might be like:

```
Filetype delimited ","
["Gender" where = 2
"Female"    =1
"Male"      =0
]
```

This script file will then looks at the data matrix, and detect that this file is "," delimited. Then it will generate an axis named "Gender" with 2 ELEMENTS namely "Female" and "Male".
"Where = 2" tells ADL to look at the second column of the matrix and count the number of "1"s and "0"s. So the result will be 2 counts on each sex.

The data matrix can be delimited by anything as long as the delimiter does not exist in the actual data. For instance,

```
1, 1, *, 3, 4
2, 0, /, 5, 3
2, 1, ,, 1, 6
0, 0, @, 1, 4
```

The above data matrix has a column (column 3) that contains a list of symbols rather than digits. The data is fine, but as "," exists as a data value, the delimiter of the data file can no longer be comma. It has to be changed to some other symbol, or otherwise completely removed, i.e. NO delimiter for this particular file, ADL will then treat each character as a column.

As an experiment, I save the matrix:

```
1, 1, 2, 3, 4
2, 0, 1, 5, 3
2, 1, 5, 1, 6
0, 0, 0, 1, 4
```

In a file named "test.dat". Also write the ADL script file interpreting this data according to the questionnaire requirements. Say, we want to find the count of female and male in the data file knowing the gender information is stored in column 2 of the data matrix:

```
Filetype delimited ","
["Gender" where = 2
"Female"    =1
"Male"      =0
]
```

Name the above piece of script "test.adl".

Run the ADL programme against these DATA ADL file pair:

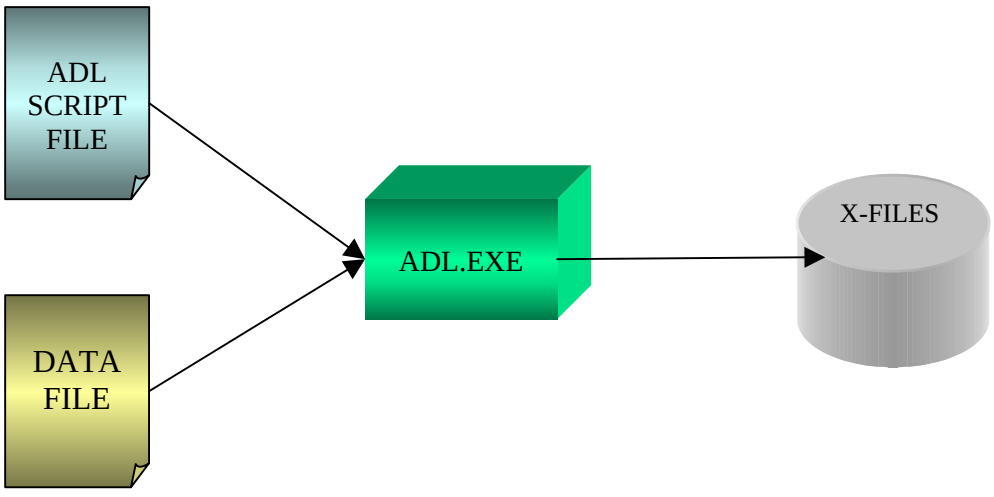
```
C:\WINDOWS>cd..

C:\>adl C:\MYDOCU~1\BTECH\TEST\TEST.ADL C:\MYDOCU~1\BTECH\TEST\TEST.DAT
ADL 3.3b Mar 14 2003
Performing database initialization
Processing your specs
Preprocessing C:\MYDOCU~1\BTECH\TEST\TEST.ADL


Scanning TEST.ADL
Starting the data conversion process
Reading raw data file
\ 0 Cases Processed
4 cases read,  4 cases output
Elapsed time 00.00.00
Processing completed.

C:\>_
```

The result of running ADL against the data file is illustrated below:



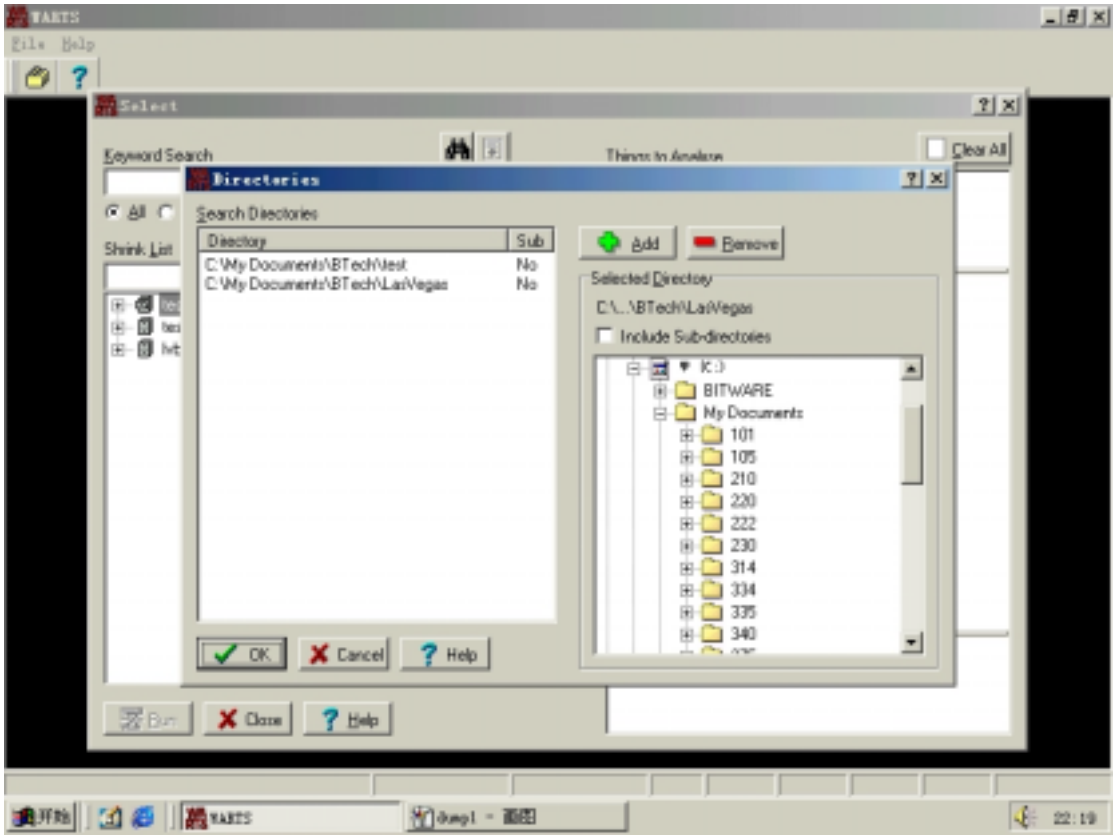

As I am using ADL, I can find features of the language that I didn't notice or even finding problems of the language. The better I understand the language, the better it is for the future work of this project.

A good example is that, while using ADL, I found some features that is supported by ADL but not stated in the ADL manual. This will result in an update of the manual in later version.

### 3.1.2 Analysis Application

ITL has produced various brands of Analysis applications. Each has slightly different features. For instance, some have more graph types supported; some have slightly different calculation and other supportive functionalities provided. Different brands suits different company's specific requirements. The following screen shots are from one of the analysis applications, called "WARTS".

**Shot 1:** Directory selection dialog box. User selects the directory that contains valid X-Files database, i.e. .xbf[2], .xef[3], .xdf[4] files.

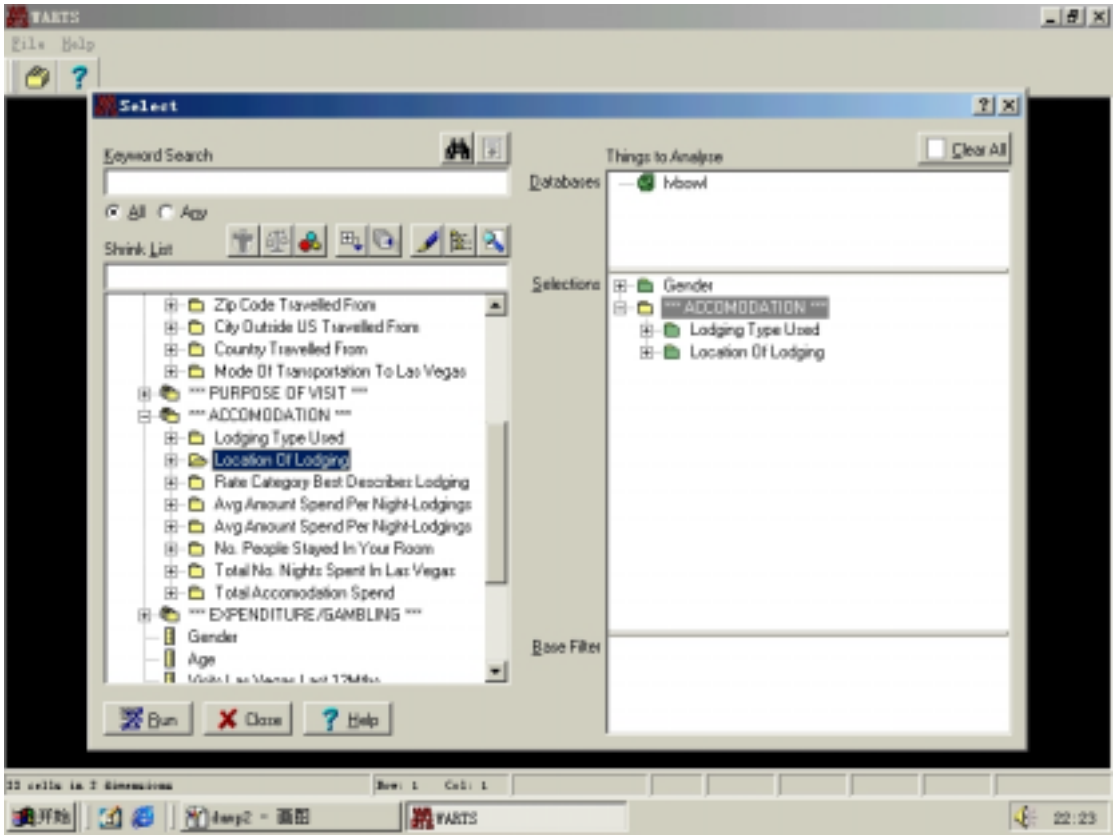



**Shot 2:** Axis selection dialog box. A tree style database is displayed in the left text area if selected directory was valid. Double click on desired axis and measures, selected ones will be copied over to the right middle text area. Click on "RUN" button, selected items will be tabulated.

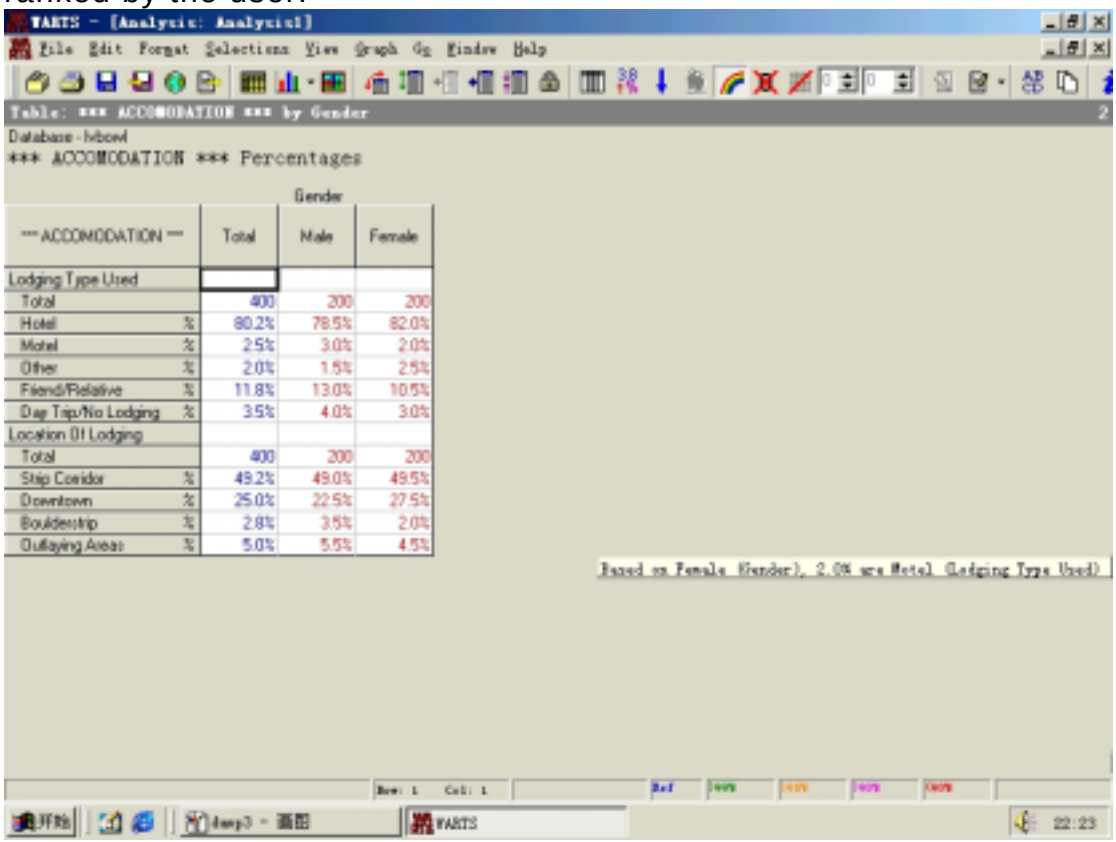
[2] *A base file, with extension xbf, storing questions*

[3] *An element file, with extension xef, storing pre-coded answers*

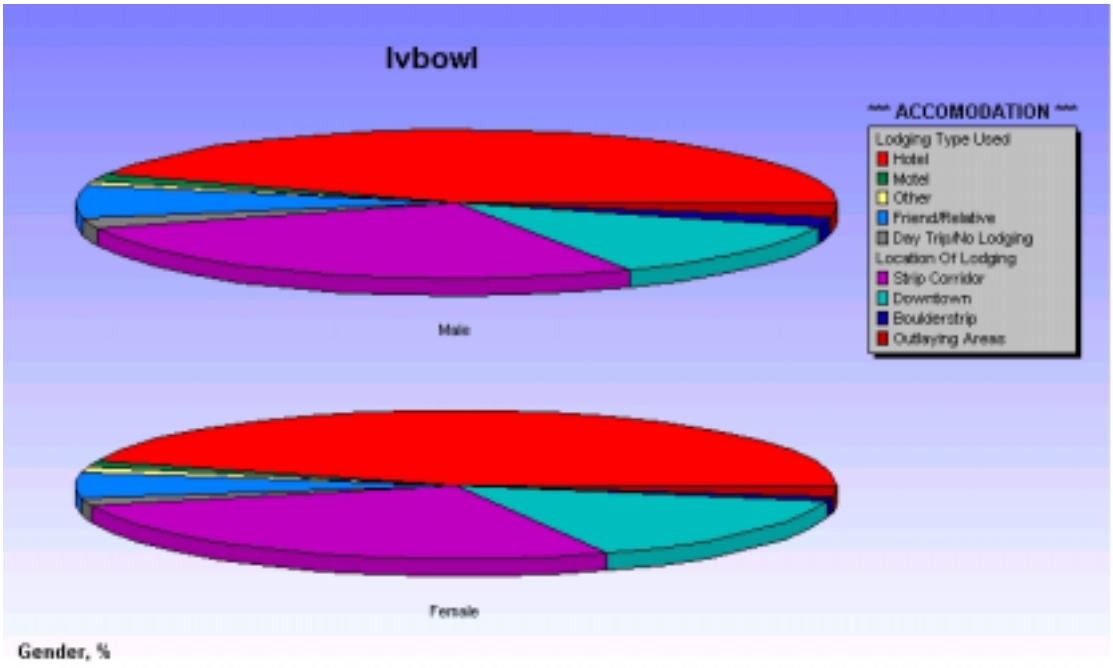
[4] *A data file with extension xdf, stores the data used in the database analysis*

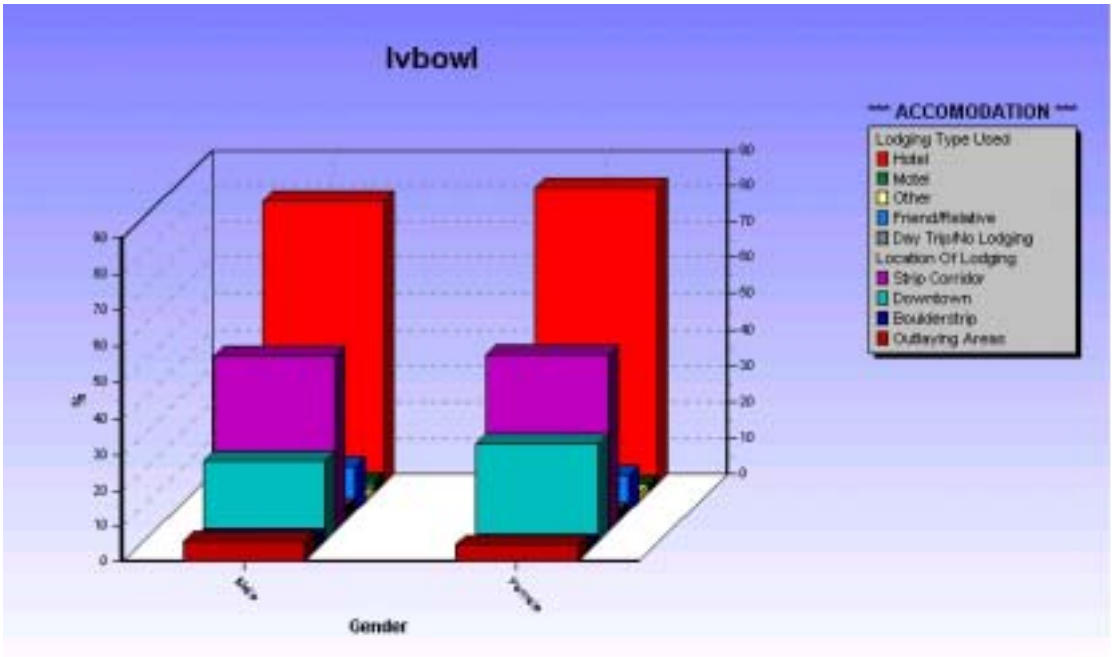



**Shot 3:** A successfully generated table according to the selected items above. Table's contents can be dynamically selected, filtered, ranked by the user.



**Shot 4:** 2 different graphs generated automatically on the same table. There are many more types of graphs to choose from.




The practice on the analysis application might seem not too relative to the purpose of project or to ADL. However, this is what the ITL's clients are most interested in, this is the ultimate purpose of the whole setup process and complex ADL script file generation process.

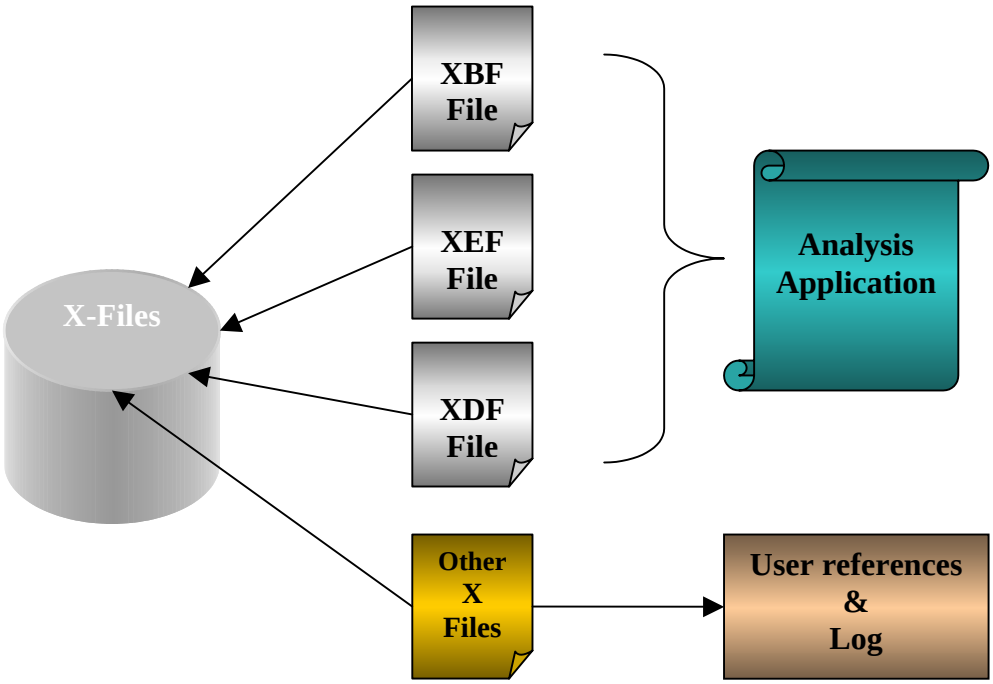
The analysis tools is useful for understanding some of the real-world databases, and how each component of the database can be combined so that meaningful information can be extracted from the tables and graphs.

## 3.2 X-Files

"X-Files" is the generic name for the ITL database; it consists of a series of files:

- The base file (*.xbf): stores the questions
- The element file (*.xef): stores pre-coded answers.
- The data file (*.xdf): stores the data
- The link file (*.xlk): contains linking information for value axes and other special commands required by the database
- The Info file (*.ifo): A text file description of the database
- The *.adt file: used by ADL to point error messages at the correct line number
- The debug file (*.dbg): contains debug information and auto output
- The log file (*.log): A log of the ADL run
- The User file (*.usr): contains top line results for checking
- The *.xbu file: contains storage structure for user created axis/question labels
- The *.xeu file: contains storage structure for user created element / answer labels
- The *.xdu file: contains storage structure for user created data

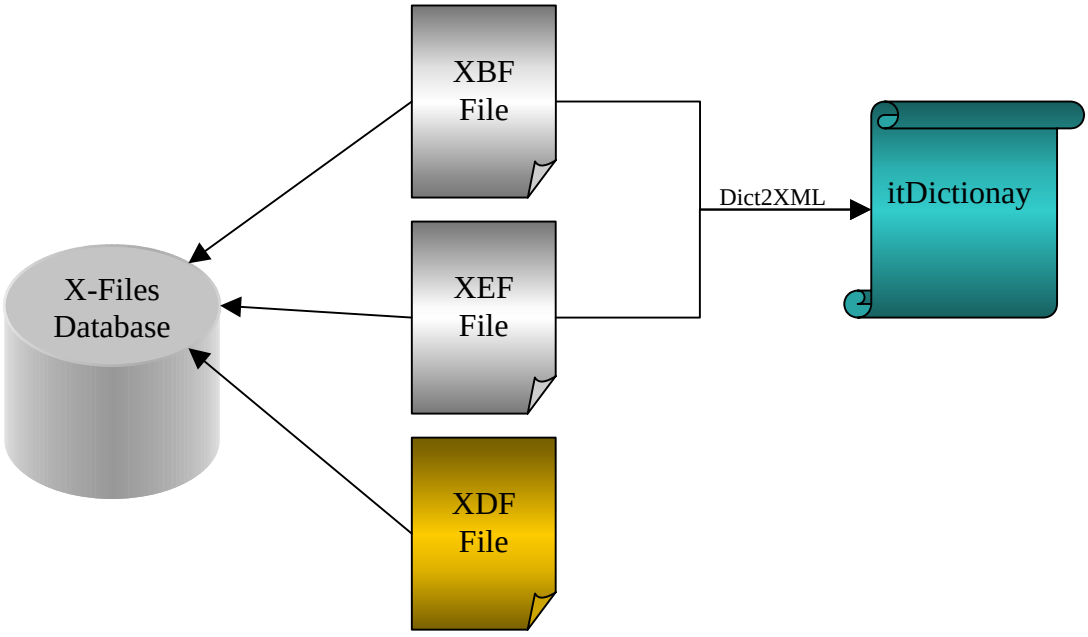
But only the XBF, XEF and XDF files are used by the analysis applications for data analysis. The rest are neither required by the analysis applications, nor supplied to the clients as part of the finished X-files database. The other files serve the purpose of user references, debug and log. These files are user specific, and are created only in their appropriate user directory, as and when they create their user created axes. The following diagram illustrates the structure and usage of all X-files.



Among the three files used by analysis applications, the base file and the element file define the DICTIONARY.
Both the base file and the element file consist of two major components:

- File Header
- Item

Following diagram visually describes the structure of the X-Files database and how the XML "ITDictionary" was generated from XBF & XEF files:




The structures of each type of the X-Files are discussed in detail in the presentation document "*X Files*"(by *Dr. S Manoharan)*.

## 3.3 Research on other project related topics

### 3.3.1 Unicode In data set-up

The X-files database originally supports ASCII characters only, and then the character set support was extended to allow any character set. Demand for having Asian character sets like Chinese, Japanese, Korean, Thai, etc. arises as ITL's products and services are introduced to Asian countries.

My supervisor Grant started a research and documentation on support for Unicode characters. I volunteered to do some of the simple

experiments on my home machine, for I have a Chinese Edition Windows platform installed on it. But his was limited to simplified Chinese characters only.

For experiment purpose, I carried out a series of tests:

1. Build simple Chinese character data files
2. Writing ADL script files processing the Chinese character data files using Chinese characters as the Axis names and element names, as well as the ADL script file name.
3. Running these files against ADL programme and check for processing error or exception
4. Viewing the Chinese character labels in the analysis application in both tables and charts, and make screen shots.

The outcome was encouraging: both ADL programme and the analysis application could handle the Chinese characters properly. Some of the test results and screen shots became helpful in completing the company documentation.

### 3.3.2 XML (Extensible Mark-up Language)

There is a tool in the “iTools” suite that is called “Dict2XML” written by *Dr. S Manoharan*. Its function is to generate a XML file from the X-File dictionary, i.e. xef file and xbf file.

To refresh the XML knowledge I already have and to gain better understanding of XML concepts, I went through the on-line tutorial materials about XML on *http://www.w3schools.com*. Some on-line forum discussing topics on XML (*http://slashdot.org/comments.pl?sid=57483&cid=0&pid=0&startat=&threshold=1&mode=thread&commentsort=3&op=Change*)

### 3.3.3 COMPSCI 330 S1 C --- Language Implementation

This stage-three computer science course at University of Auckland helps to understand on how computer languages are specified, what they mean, and how they are implemented, is fundamental to computer science.

Unfortunately I did not study the course in advance of starting the project. Lack of knowledge on the aspects this course covers might lead to huge difficulties while proceeding with the project work.

Therefore I downloaded the lecture handouts and some of the reading materials provided on the course web page and studied thoroughly. These materials gave me a slightly different view of programming languages, and I found it helpful at the design stage of the new language.

### 3.3.4 Triple – S (***SSS***)

As the project proceeds, I was introduced to some scripting languages and software that have basically the same functions as ADL language and programme, e.g. triple-s; Quantum; Snap, etc. Each language or software has its strength and weakness.

Triple-S is probably one of the popular ones. It also drew my attention because it is using XML as the file format for the script file. This happens to have similar sides with my own design (For details of my design, please refer to *4.2 Enhanced ADL Design*)

The triple-s was first published in 1994 and evolved through the last 20 years:
**Triple-s version 1.0:** The original standard that came out in 1994. It describes variables and data for single, multiple, quantity, character and logical response variables. Includes sample survey files.

**Triple-s version 1.1:** The revised standard became available in 1998. It incorporates many new features; also the syntax was improved at some stage.

**Triple-s XML version 1.1:** This is the original version that was published in February 2000. It uses XML syntax, it includes an on-line DTD and sample survey files that are available for use/reuse.

**Triple-s XML version 1.2:** The latest version available for download.

# 4. Documentation & Design

## 4.1 ADL Language Documentation

As described in the project objectives, my second task is to produce documentation for the ADL language.

ADL language is similar to many other programming languages, it has two main parts: semantics and the syntax. Since the semantics are clearly defined in the ADL manual, my documentation will focus on the syntax part. After some research and discussion with my supervisor, as well as confirmation of University of Auckland BTech coordinator, Dr. S Manoharan, I've decided to use the ***BNF (Backus-Naur Form notation)*** for the documentation.

There were many resources on the web about BNF documentation, the one I studied on was
"*The BNF Web Club Language SQL, ADA, JAVA, MODULA2, PL/SQL…*"
----http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html

There was complete documentation for various programming languages, as well as database languages that I found extremely helpful for my own documentation.

On top of the traditional BNF, I "invented" a few "meta-symbols" in order to have clearer description of the ADL language. The list of meta-symbols used in the documentation can be found in "***BNF_ADL documentation***".

The documentation was written originally in plain text file, after general completion, I modified it to a HTML file and added hyper-links within the document. This gives better navigation help to the user, simplifies the reading process.

However, there are still errors and statements that are ambiguous. ITL intends to keep the documentation for lifetime use, so it should and surely will be kept on changing and correcting through the following years. As long as there is modification to the ADL language, there will be change to the documentation. For instance, if the enhanced ADL language was completed and approved after this project, the documentation will need to be updated.

## 4.2 Enhanced ADL language design

After the completion of *BNF_ADL documentation*, I have gained even better understanding on the current ADL language and observed many of the strengths and weaknesses of the language. Next is to design a model that is enhancing the current language, so that ADL can be more productive.

My design was carried out according to the requirements from both conceptual level and the practical level as listed in section 2.1 and further discussed in the "*Enhanced ADL design document*".

Three initial models were raised, each using a different approach:

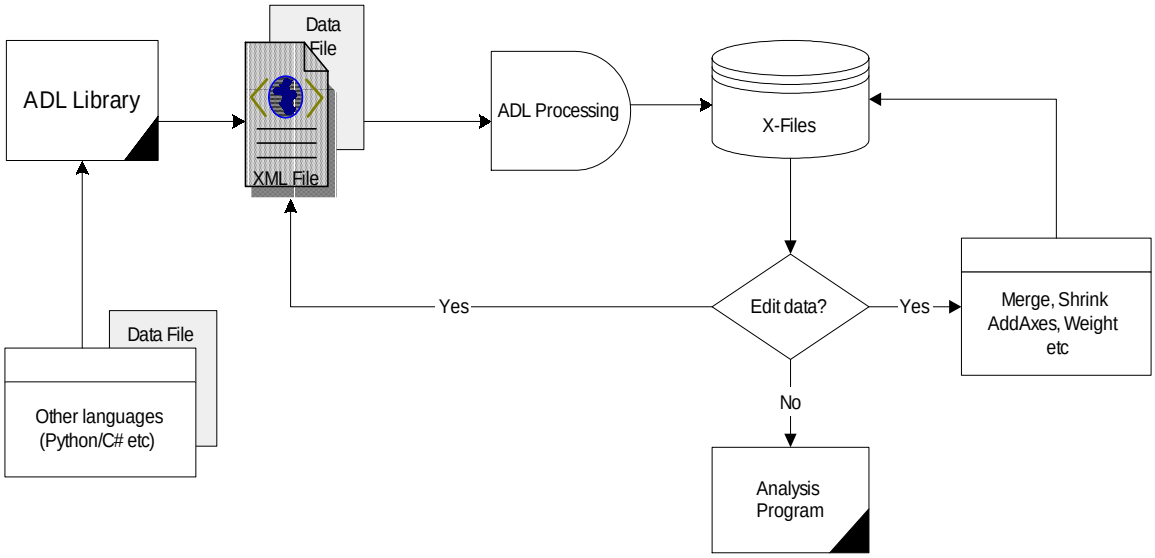
- ***Model One***: "Super ADL" --- Original ADL language with extra features
- ***Model Two***: ADL script file generated using another programming language
- ***Model Three***: XML format of X-Files Database & XML to X-Files engine

**Pros** and **Cons** of each alternative are discussed in the design document. A test table was set up with regards to the requirements; each alternative was rated according to these requirements.

By then, I was clear that each model has its own advantages and disadvantages.
The more advantages my final solution can have the better. However, this is limited by both the time and the complexity.

My decision for now is Model 2, i.e. to rewrite ADL in another programming language, using API for the programming language (XML handlers, libraries), and output a XML file as an intermediate format. A convert engine will then take this XML file and the original .DAT file as input in order to generate the ITL Database in X-Files format.

This is shown in the following diagram:


ADL Library
Data File
XML File
ADL Processing
X-Files
Edit data?
Yes
Yes
Merge, Shrink
AddAxes, Weight
etc
No
Analysis
Program
Data File
Other languages
(Python/C# etc)

# 5. Software Implementation

Once the design is completed, detailed implementation plan was set and all the software implementation follow the order and the time allocated to each stage. For details, please refer to
*"Enhanced ADL –Implementation Plan"*

## 5.1 XML → X-Files converter

### 5.1.1 Introduction

One of the applications in the ITOOLS application suite is "DICT2XML". This application takes the .xbf and .xef files from an X-file database and reformats them to an XML file, which is defined as the "Dictionary".

Because to set-up X-file databases efficiently is the goal of the set-up process, in order to have deeper and clearer understanding of the X-file database format, I started this "reverse" application that reverses the "DICT2XML" generated XML back to the "DICTIONARY", i.e. *.xbf & *.xef files.

### 5.1.2 Programme Walkthrough

1. Run the application xml2x.exe executable
2. The console application starts up and asks for an input XML file. Type in the path and file name of the XML to be converted. Press enter, the conversion process starts.
3. Once the conversion is completed, the programme will notify the user. By now, if the conversion was successful, there should be two new files with the same name and extension of ".xef" and ".xbf" stored in the same directory as the input file.

### 5.1.3 Pros & Cons

- **Pros:**
  Full cycle of file format conversion between X-file database files and XML file. It provides better understanding of the X-file database files. This was accomplished with reference to the binary file structure of *.xbf and *.xef files that was stated in the document "*X files*" (*Dr. S Manoharan*).

- **Cons:**
  Cannot complete the conversion straight from a raw data file to *.xdf file. This was due to lack of understanding and lack of documentation of the base file of X-file file set.

The resulting files from the application are binary files that cannot be interpreted by normal text editor or the ITL analysis tools. During programming and debugging process, a binary viewer named “FRHED.EXE” was used for comparing the binary values of the output files from my application and the ADL generated x-files

### 5.1.4 Future Ideas

Produce a detailed X-file documentation. Maybe not necessary normal ADL language and Analysis application users, but will be very helpful to future development of ADL language and programme, as well as the improvement of X-file database.

## 5.2 Simple DATA File Processor

### 5.2.1 Introduction

Having finished the XML → X-Files converter, I have gained some experience on file type conversion, especially in Binary file input and output process. Major difficulty was at the Data file conversion part, due to the lack of understanding of the binary arrangement of the *.xdf.

Then the idea of building a new data file processor that is parallel to ADL programme emerged. Although the new processor will not have as many features and functionalities as the ADL programme, I expect to explore the possibility of a new method of processing the same data, and want to compare the speed of the new processor with the ADL programme in particular.

This simple data processor is a console application written in C# language. Takes in a data file with any extension and contains a comma separated data matrix, output the expected result on the screen.

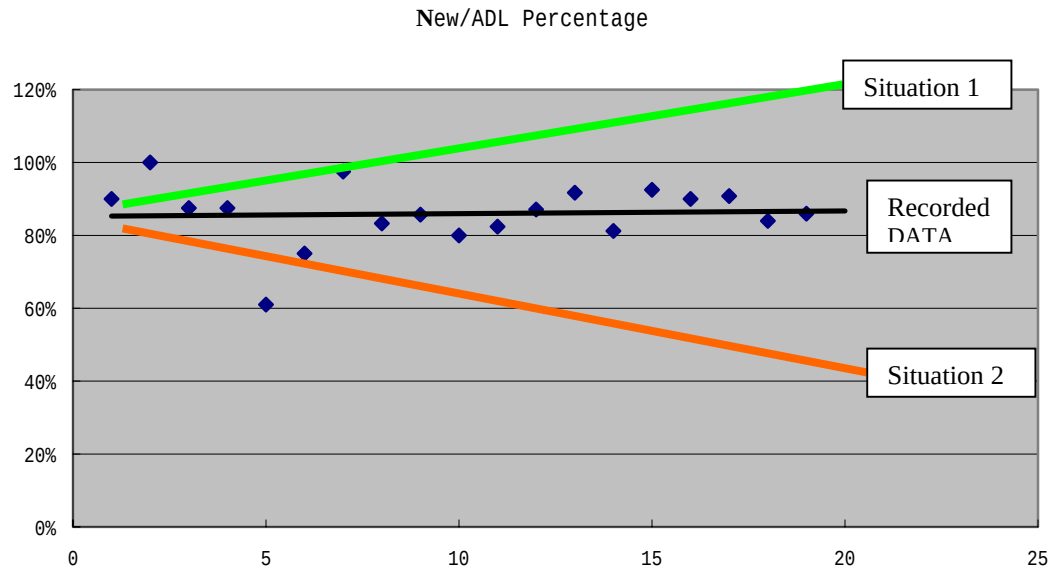
### 5.2.2 Programme Walkthrough

1. Start the console application, a command line will ask for two separate user input values: Number of axes to generate; Number of elements for each axis
2. Reads in a specified file that contains a comma delimited data matrix.
3. Reads the data matrix row-by-row, increment the count of matching value accordingly. Similar idea as ADL programme.
4. The start and finish times are recorded and the time difference are printed on the screen

### 5.2.3 Pros & Cons

I randomly generated a 200Column*10000Row comma separated Data Matrix, and then ran ADL and the new processor against the data on the same machine. Time measurements were taken and tabulated as follows:

| Axis # | Element # | ADL processing Time (Sec) | New processing Time (Sec) | Percentage New/ ADL |
|---|---|---|---|---|
| 1 Axis | 2 | 3 | 2.7 | 90% |
| | 10 | 3 | 3 | 100% |
| | 20 | 4 | 3.5 | 87.5% |
| 10 Axes | 2 | 4 | 3.5 | 87.5% |
| | 10 | 7 | 4.3 | 61% |
| | 20 | 12 | 9 | 75% |
| 20 Axes | 2 | 4 | 3.9 | 97.5% |
| | 10 | 12 | 10 | 83.3% |
| | 20 | 21 | 18 | 85.7% |
| 30 Axes | 2 | 5 | 4 | 80% |
| | 10 | 17 | 14 | 82.4% |
| | 20 | 31 | 27 | 87.1% |
| 40 Axes | 2 | 6 | 5.5 | 91.7% |
| | 10 | 22 | 18 | 81.2% |
| | 20 | 40 | 37 | 92.5% |
| 50 Axes | 2 | 8 | 7.2 | 90% |
| | 10 | 26 | 23.6 | 90.8% |
| | 20 | 50 | 42 | 84% |



From the time values table and the trend line drawn from the new/ADL (%) chart we can see the new processing time stays at a fairly constant rate of 86% comparing to the original ADL processing time.

While the number of axes and elements grows greater and greater, if the trend line drawn happened to be in situation 1 (Green Line), the new processor's processing time tends to be slower and slower comparing to the original ADL programme; If the trend line drawn looked like in situation 2 (Red Line), the new processor's processing time tends to be faster and faster comparing to the original programme.

It would be worthwhile to continue with the new processor experiment if the trend line happened to be in situation 2.

Considering the recorded time for ADL processing includes the time for generating 6 files: .xbf, .xdf, .xef, .adt, .log, .usr, whereas the time for the new processor does not include any file generation.

I proved that ADL has its advantage in data processing speed. Therefore, my main application will be concentrating on the improvement of the ADL language input part and leave the data processing part untouched. This solution will make the most use of the ADL's

### 5.2.4 Future Ideas

Although the testing programme did not improve on the data processing speed, future projects could be set up to find ways to increase the data efficiency of ADL or substitute the original ADL by a completely new data processing algorithm.

## 5.3 XML → ADL converter

### 5.3.1 Introduction

Since I have chosen to use XML files to store the newly designed ADL script structure, it is essential to design a clear and simple XML formatted ADL structure.

The *BNF formatted ADL* documentation plus the research work I did during the documentation became great help when I moved on to design the XML formatted ADL structure.

In addition to the research on BNF format, the research on *triple-s* provided good background knowledge as the latest triple-s format is in the XML format.

The final structure contains two major parts:

```
- <ADL>
  + <Header>
  + <Body>
  </ADL>
```

- Header: consists of all file level properties

```
- <Header>
    <Debug />
    <fileTypeDelimited />
    <Filter />
    <ifo />
    <Include />
    <Link />
    <maxAxisLabel />
    <maxCases />
    <maxElementLabel />
    <maxLen />
    <maxRec />
    <minCell />
    <minTable />
    <Other />
    <specialChar />
    <Suppress />
  </header>
```

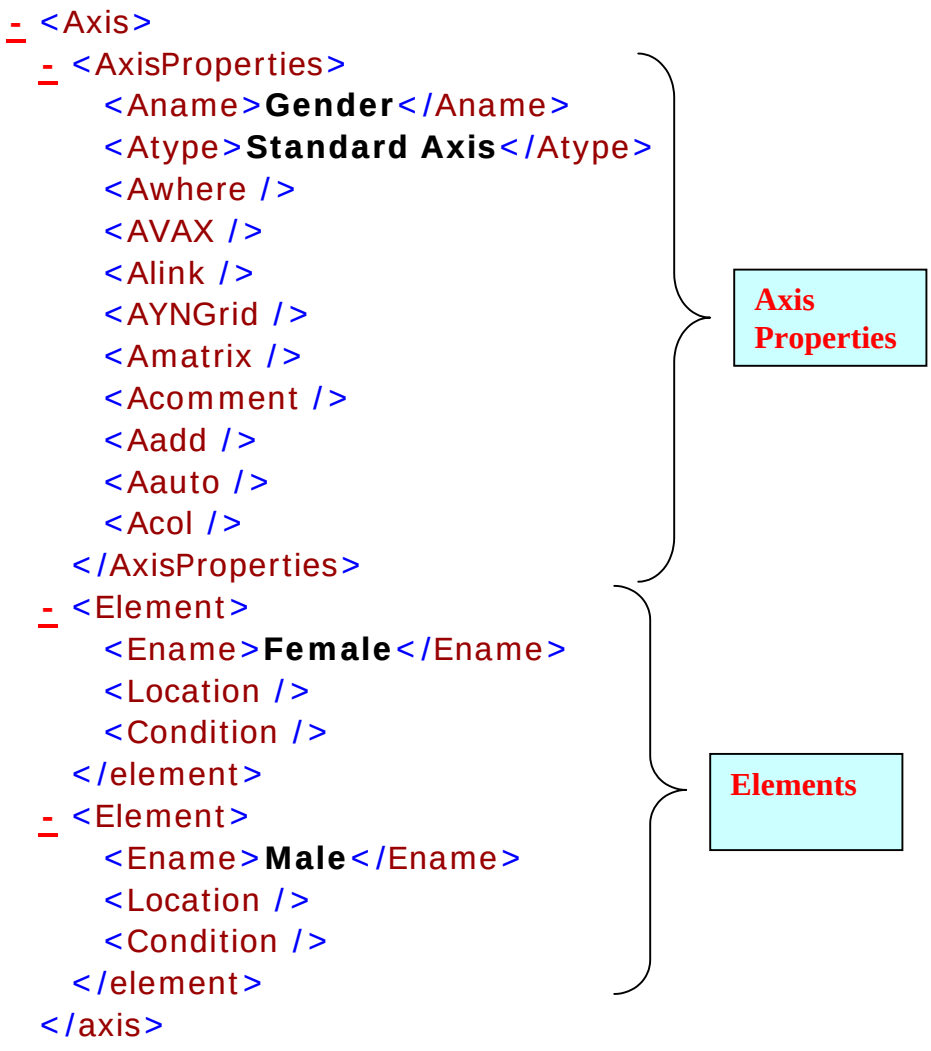
- Body: consists two sub-parts:

```
- <Body>
  + <Macros>
  + <Axis>
  </body>
```

  - Macros: may contain zero or many macro objects with a macro name and detail pairs.

```
  - <Macros>
      <Mname>mac1</Mname>
      <Mdetail />
      <Mname>mac2</Mname>
      <Mdetail />
    </macros>
```

- Axes: may contain zero or many axes object with axis level properties and zero or many elements. Axis may be of various types.




## 5.3.2 Programme Walkthrough

This programme is a GUI (Graphical User Interface) type of application. It has two major functions:

- XML to ADL file conversion
  1. User starts the programme (screen-shot: *Figure 1*), enters the file path (if the file resides in different directory from the convert tool) and name in the top text box, or alternatively uses the browse button to select file from an open-file dialog box (as shown in *Figure 2*).

2. Press the top convert button to start conversion. Once conversion is completed, a message will be printed in the status bar that is at the bottom of the application window.
3. If no error occurred, an ADL file with the same file name as the input file and ".adl" extension should be generated now in the same directory as the input XML.

- X-Files generation
  1. User enters the file path and name for ADL file and Data file in the bottom two text boxes accordingly, or alternatively uses the browse buttons to select file from an open-file dialog box.
  2. Press the bottom convert button to start conversion. This is in fact an interface for the ADL programme. The input files are input files for the ADL programme, and the convert button runs the ADL programme.
  3. Once conversion is completed, a notice will be printed in the status bar. Output X-files are generated in the same directory as the input ADL file (not the data file).


**Figure 1: Convert Tool**

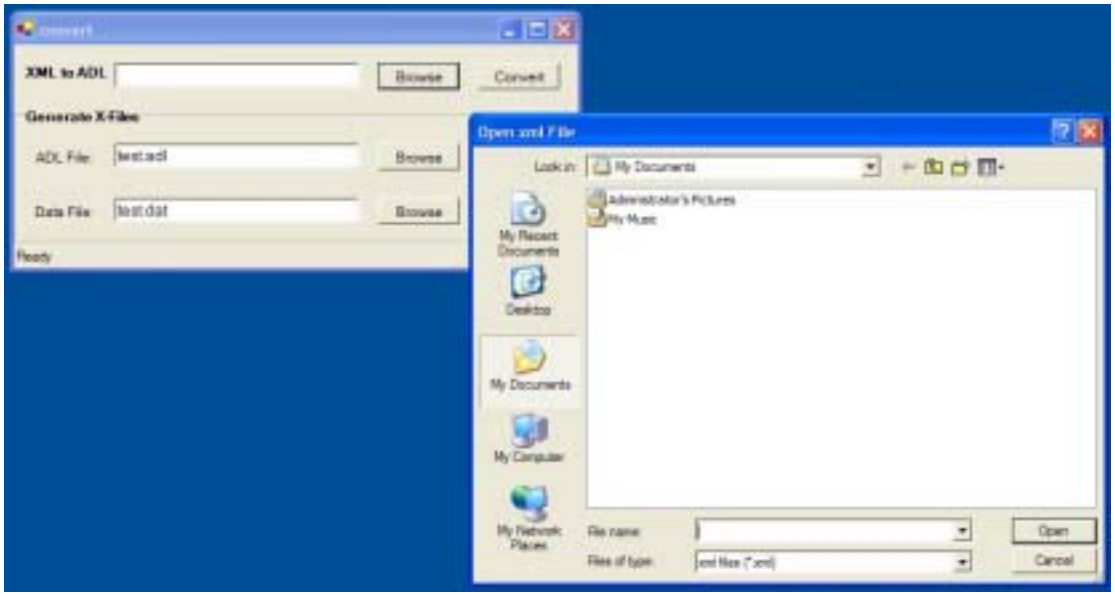



**Figure 2: Open file Dialog**

### 5.3.3 Pros & Cons

**XML to ADL**

- **Pros:**
  Since I have decided to keep original ADL programme as the data processing engine, there must be a "Bridge" between the newly designed XML formatted ADL script and the traditional ADL language formatted file in order for the ADL programme to parse the script. The first major function of this application functions as the "Bridge" needed.
- **Cons:**
  Could potentially slow down the "data set-up" process because of the extra step in between, i.e. (XML→) ADL → X-files

**Generate X-files**

- **Pros:**
  The second function of this application provides a graphical interface based on the traditional command line. May save some typing time, or avoid ADL running exception due to typing mistake.
- **Cons:**
  Because the traditional ADL programme requires a keyboard "enter" to close its console, after the new interface finished the conversion, there is the ADL programme window left open showing the processing information. Some might consider this left-behind window unnecessary.

### 5.3.4 Future Ideas

To skip the extra conversion step, or combine the two conversion steps either by creating a new ADL format that could be parsed directly by the current ADL program, or by producing a new ADL programme that parses XML formatted ADL script file.

## 5.4 Macro Generator

### 5.4.1 Introduction

This application allows users to build / modify macros and store the macros as a list in an XML file.
According to the definition of macros in the ADL manual, a macro defines a list of elements for later use. This will save user some typing time. However, through experiments, I discovered that a macro could theoretically contain anything, e.g. axis, comment, variable declaration, etc.

### 5.4.2 Programme Walkthrough

1. Starts the application. Click on the main menu item “File” to create a new macro file or select an existing macro list file to edit.

2. Create a new Macro object by entering the name in “Macro Name” text box, then “Add” button to add to the macro list (displayed in the list box “Macro list”)

3. Select an item from the “Macro list” list box; the macro detail is displayed in the text box “Macro Detail”. Modify the macro detail as needed and press “OK” button to save the change, else use “Cancel” button to clear the macro detail

4. If user needs to change a Macro’s name, simply click on the macro name in the macro list. The current macro name is displayed in the “Macro Name” text box, type in the new name and press “update” button to save the change.

5. When editing is finished, used the File → save/save as menu to save the Macro list to an XML file.

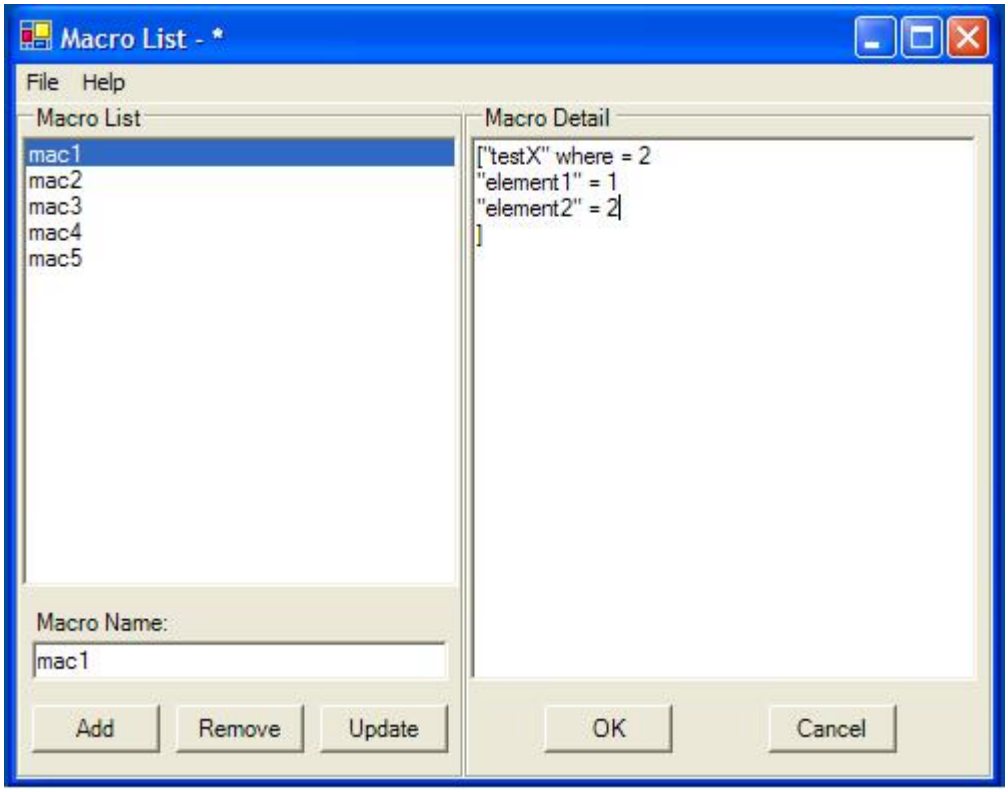



**Figure 3: Macro List Generator**

### 5.4.3 Pros & Cons

- **Pros:**
  1. In the traditional ADL language, macros could be scattered all over the file, once the file becomes large, it might be extremely hard to find a particular macro. This application solves the problem by ordering them and placing them in one place. Makes it easier to find a particular macro by its name.
  2. Since the macros are stored separately from the rest of the ADL scripts, it could be use/reused by any ADL file.

- **Cons:**
  1. May exist too loosely from the rest of the ADL script contents, becomes hard to use.
  2. If a macro was written for a particular file only, it will not be able to be reused by other files unless the macro details are changed.

### 5.4.4 Future Ideas

To build context free macros that can be used/reused by any ADL script file. A Macro functions more like a method as in a programming language (e.g. java, C++, etc.). The Macro List XML file will act as a library file (e.g. macro.dll)

## 5.5 ADL_CS Main Application

### 5.5.1 Introduction

ADL_CS is the ADL script builder for the new XML formatted ADL file. If a user is provided an XML formatted ADL skeleton file, he could modify the file by adding axes and elements to the file. But writing in XML tags is probably as difficult as writing in ADL language, and even harder to make mistakes.

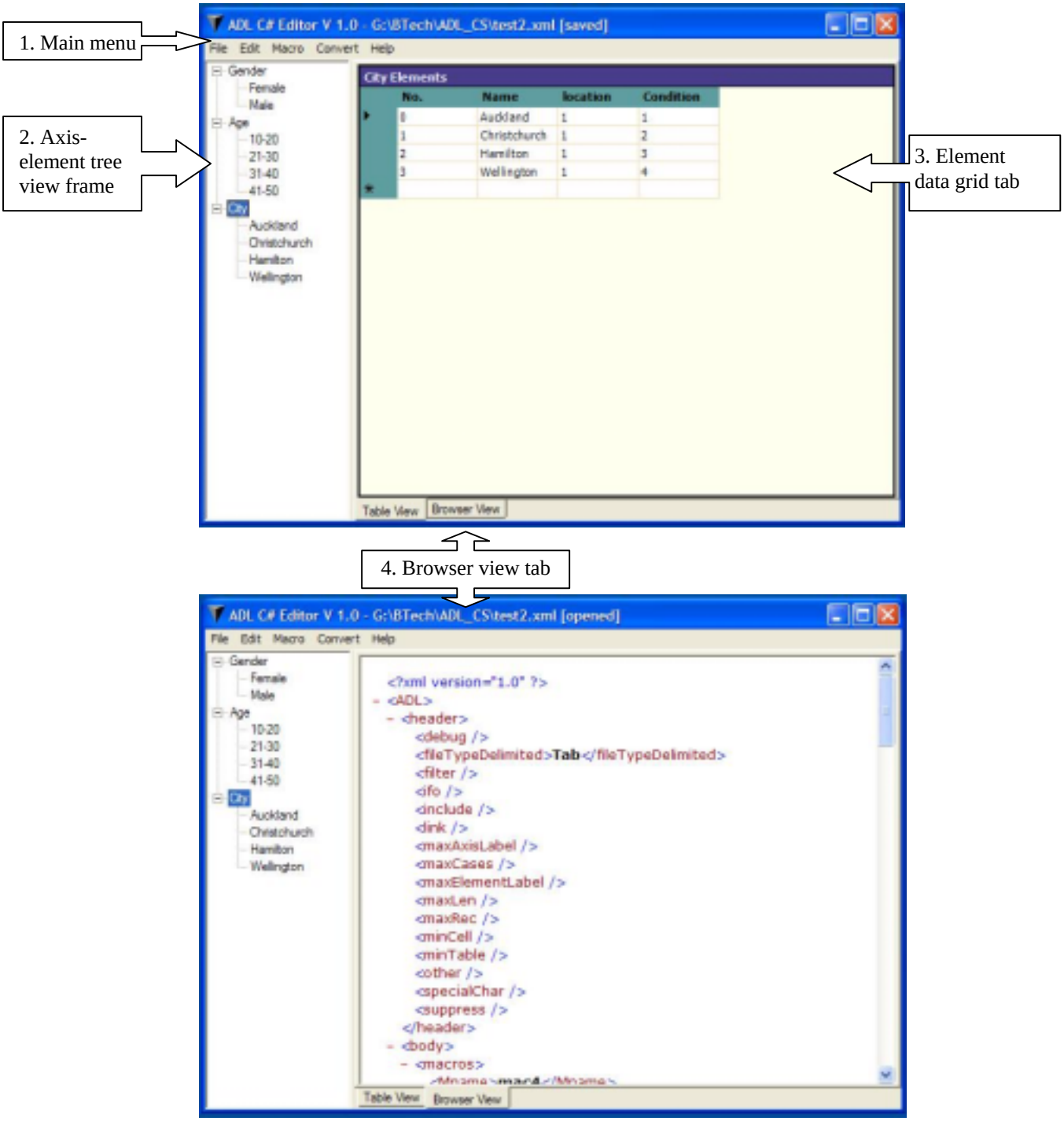
Because one of the issues with the current ADL language is the unusual syntax, it is hard to learn for people with or without computer programming background. Therefore, I built this “ADL_CS Editor V 1.0” aims to make writing ADL as easy as using a windows explorer. The application name indicates this is an ADL editor written in C sharp language, and the current version is 1.0.

Quite a number of ADL features were not included in this application due to various limitations. Also, because my understanding for ADL language is still limited, I might no realize the existence of some ADL functionalities.

As this is a research project, the emphasis is on exploring possibilities of improvement; I concentrated on improving the main features of ADL and skipped some unusual ADL features, or set only a skeleton for future development purpose.

### 5.5.2 Application Functions

The application consists of 4 main interface sections:




1. Main menu:
    - File
        - "New ADL": Creates a new ADL file
        - "Open ADL": Opens an existing ADL file
        - "Close ADL": Closes the currently opened ADL file
        - "Save": Saves any changes been made to the opened ADL file
        - "Save As": Saves the currently opened ADL file as a different file or to the save file name.
        - "Exit": Closes the application interface.

- Edit
  - "Axis"
    - "New": creates a new axis and add it to the current file
    - "Remove": removes the currently selected axis
    - "Property": opens the axis property dialog, allows user to edit properties of the selected axis.
  - "File Property": opens the file property dialog, allows user to edit properties of the current file.

- "Macro"
  - "New Macro": opens the "*Macro generator*", allows user to create a new macro list file (in XML format), or modify an existing macro list file.
  - "Select Macro": opens the select macro dialog, allows user to select macros from an existing macro list file and add to the current ADL file, or delete unwanted macros.

- "Convert": opens the "*XML → ADL converter*", allows user to convert the current XML formatted ADL file to the traditional ADL syntax script file; once the traditional ADL file is created, user can run the ADL programme from the interface to create X-file database.

- "Help":
  - Runs the compiled HTML ADL help manual
  - Version information of this ADL_CS editor

2. Tree view panel

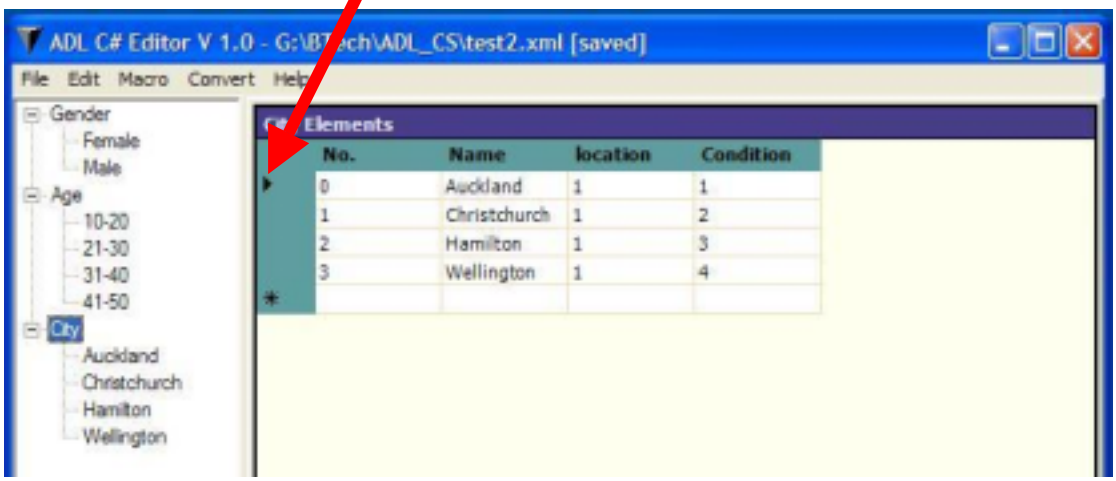
All the axes and elements are listed in the tree view by their names. The parent nodes are axes. Each axis node may contain zero or many elements shown as child nodes belong to the axis node.

Different "selected node" switching actions may cause the application respond differently:

**From one axis node to another axis node** causes refresh of the element details shown in the Data Grid tab on the right-hand-side of the windows form, any change made to the previous selected axis object is saved automatically.

**From element node in one axis node to element node in another axis node** causes refresh of the element details shown in the Data Grid tab on the right-hand-side of the windows form, any change made to the previous selected axis object is saved automatically.

**From one element node to another element node in the same axis** causes the table **record indicator** jump to the current record. Note that the indicator points at the top record by default.




3. Data Grid tab

For the currently selected node in the tree view panel, it displays Sequence Number, Name, Location and Condition values of each element object in that belongs to the axis object.

The record indicator points at the currently selected element node in the tree view panel or points at the top record by default.

4. Built-in web browser

Displays the original XML formatted ADL file with no parsing. This is for user reference only, not editable.
Once any change made to the current file is saved, by right-clicking in the browser area then refresh, the displayed file details are updated.

### 5.5.3 Programme Walkthrough

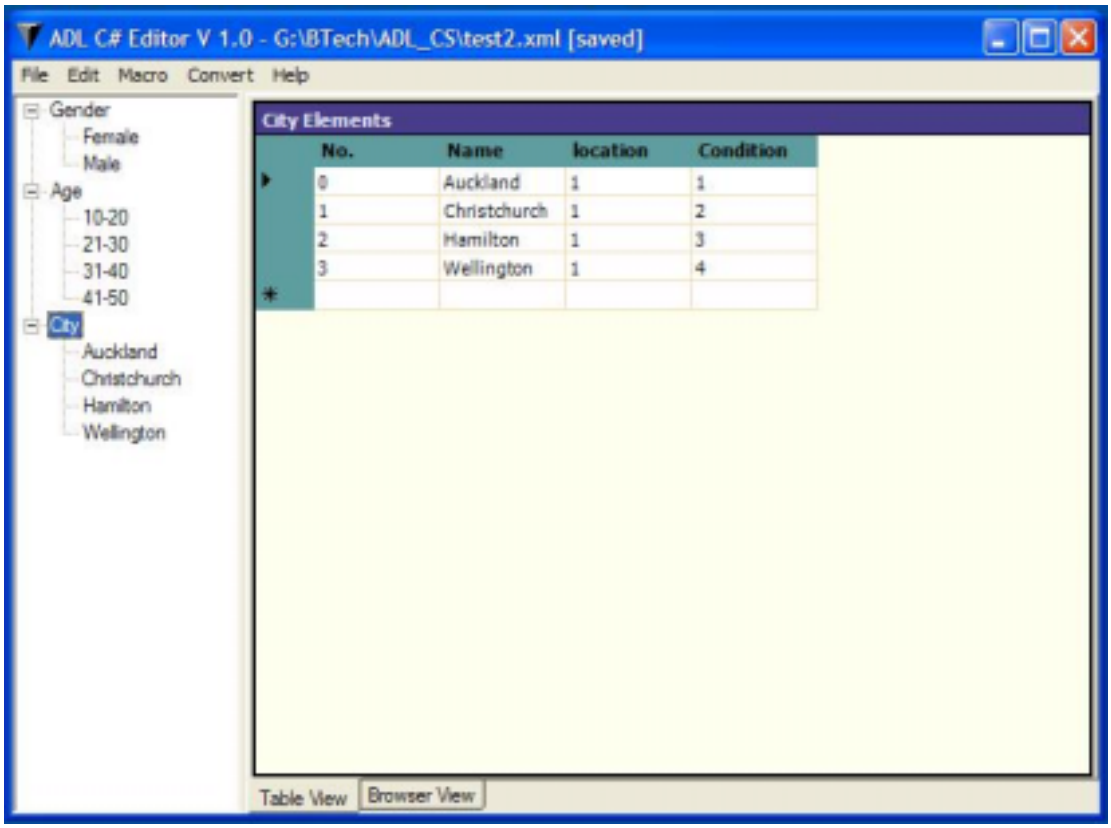



**Figure 4: ADL_CS Editor Interface**

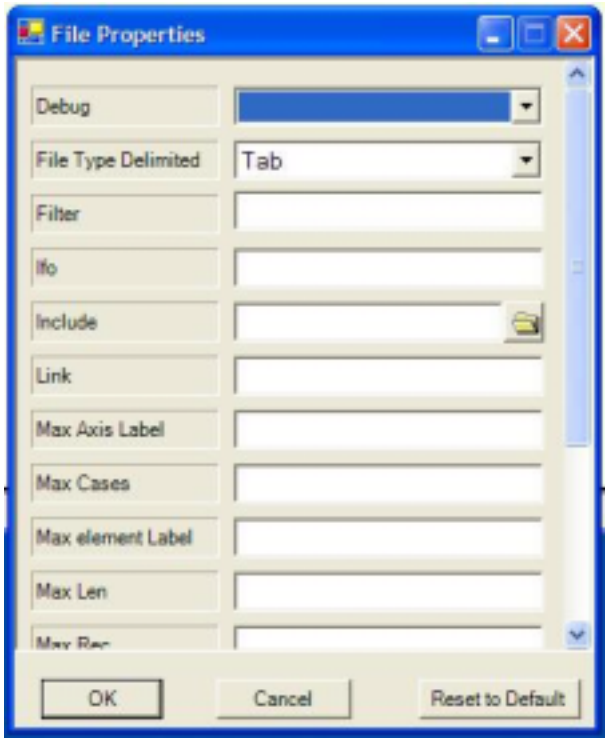



**Figure 5: File Property Dialog**

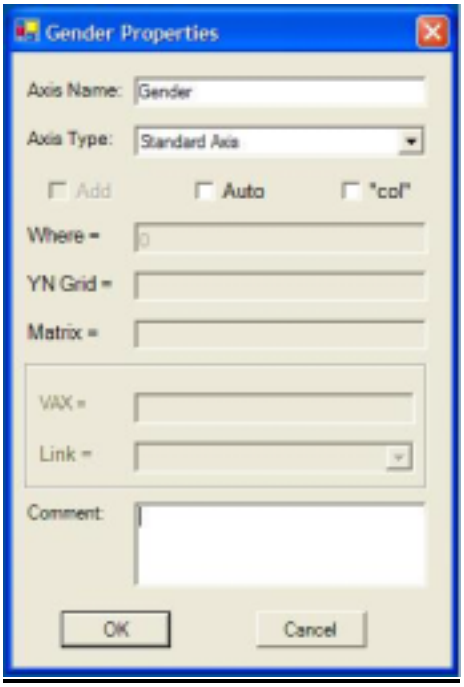



**Figure 6: Axis Property Dialog**

## Create

1. Main menu: "**File**" → "**New ADL**" → "File Property Dialog" appears
2. Configure the file properties in the "**File Property Dialog**" (***Figure 5***) if required. Click "**OK**" button when completed.
3. Main menu: "**Edit**" → "**Axis**" → "**New**" → "**Axis Property Dialog**" appears (***Figure 6***)
4. Set the Axis name -- "Gender", Type – "Standard Axis", as well as other properties if necessary. Click "**OK**" button to save and exit this dialog
5. Add two elements into the table on the right-hand-side of the application:
    a. **Element 1:** Name = "Female"; Location = "2"; Condition = "1"
    b. **Element 2:** Name = "Male"; Location = "2"; Condition = "0"
6. Repeat step 2 –5 to add two more axes: "Age" & "City". Change the names and values where suitable.
7. The final structure will be as illustrated in ***Figure 4.***

## Open

1. Main menu: "**File**" → "**Open**". Open an XML formatted ADL file to be modified.
2. Tree Structure of the file will be printed out on the left-hand-side of the windows form.
3. Collapse or expand the sub-tree Nodes to hide or show all the elements that are belonged to that particular axis. All element details of the selected axis will be printed in the table on the right-hand-side of the windows form.

**Edit**

File property and Axis property can be modified at any time

1. Configure "**File Property":** Main menu, "**Edit**" → "**File Property**" → "**File Property**" dialog will appear
2. 2 ways to bring up the "**Axis Property**" dialog: Highlight the axis to be configured.
    a. Main menu: "**Edit**" → "**Axis**" → "**Property**"
    b. Mouse right-click → "**Axis Properties**"

**Save**

To save the file: Main menu, "**File**" → "**Save**" (or "**Save As**"). The file will be saved as an XML file with the name specified by user.

**Delete**

- Axis: Select an axis to delete
    1. Main menu: "Edit" → "Remove"
    2. Context menu: Mouse Right-click → "Remove"
- Element: Highlight the element record to be deleted, and then press the "Delete" button on keyboard.

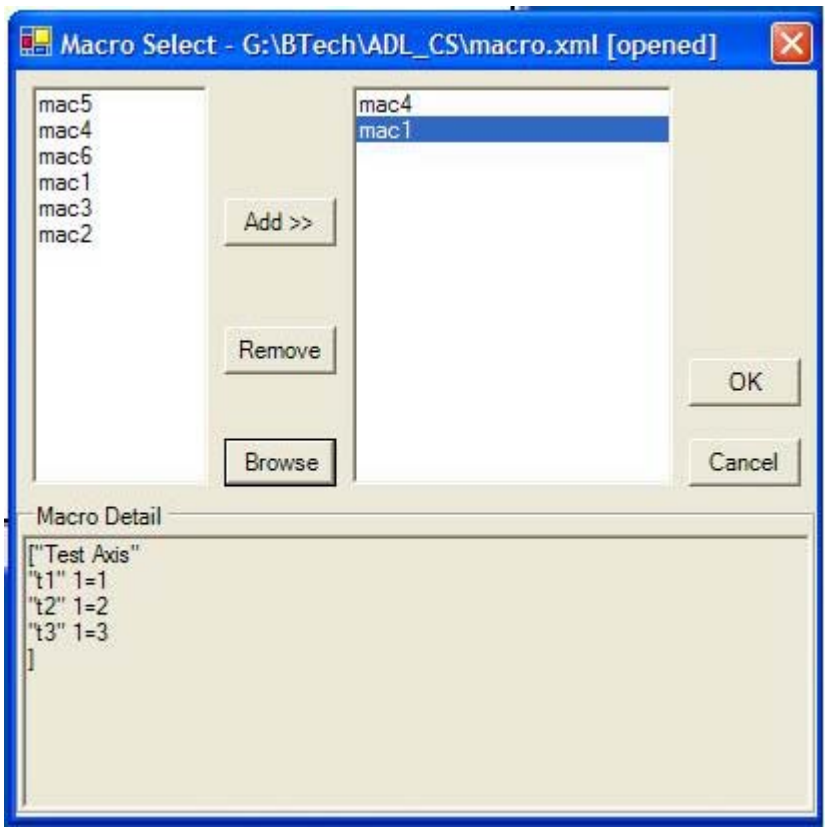



**Figure 7: Macro Select Dialog**

**Macro**

Main menu: "Macro" → "Select Macro" to bring up the macro select dialog. The macro list belongs to the current file might be empty, or already contains some macros.

- Add:
  - Click "**Browse**" button to load in a macro list from a macro XML file. Once loaded successfully, the macros' names will be listed in the list box on the left-hand-side of the form.
  - Select a macro to view its detail. If the macro matches the user requirement, click on the "**Add >>**" button to add it to the current file.
- Delete:
  - Select a macro's name from the right-hand-side list box to view its detail. If the selected macro is no longer required, click on the "**Remove**" button to remove it from the current file.

Once the modification is done, click the "**OK**" button to save the changes and exit the dialog, or "**Cancel**" button to abort all the changes.

### 5.1.3 Pros & Cons

**Pros**

- Provides an easy-to-use graphical user interface for building ADL script files.
- User does not necessarily need to know the ADL syntax for writing an ADL file to process a data file.
- A platform that integrates all the other tools. User can perform multiple actions without having to open each tool individually.

**Cons**

- As the application contains several little tools and many components, each part might not coordinate with another properly. Possible gaps between components.
- ADL language Experts would possibly think of this an extra step that is not that useful. Could even slow down the database set-up speed for them.
- Since this application parses the newly designed XML formatted ADL files only, it has not been tested against real world jobs.
- Hidden errors / exceptions that could only be discovered through more complex testing processes.

### 5.5.4 Problem Discussion

I encountered on “interesting” problem during the coding process of this ADL_CS editor.

**Source:** Microsoft Visual Studio V1.1 C#

**Description:** The ADL_CS editor application is made up of one solution containing multiple windows forms: Convert tool, Macro select dialog, and file property dialog, axis property dialog. Each of them has multiple controls: button, text box, combo box, label, etc. Up to a certain stage, the Microsoft IDE is not able to recognize and manage all the user controls any more, so at design view, all the controls are out of order. This blocked the coding process completely. For example, one of the error messages for **button control** says: “`Duplicate declaration of Member 'Location'. The variable 'Button' is either undeclared or was never assigned.`”

**Investigation:** Researches were done on the Internet. Amazingly, I found that many other VS .NET programmers encountered the same error. No official solution to this yet. Some forum posts gave comments and solutions of how they fixed the error. Unfortunately, they did not work in my case.

**Solution:** I divided the one solution into a few separate ones and assigned different namespace names to them, and the problem was fixed.

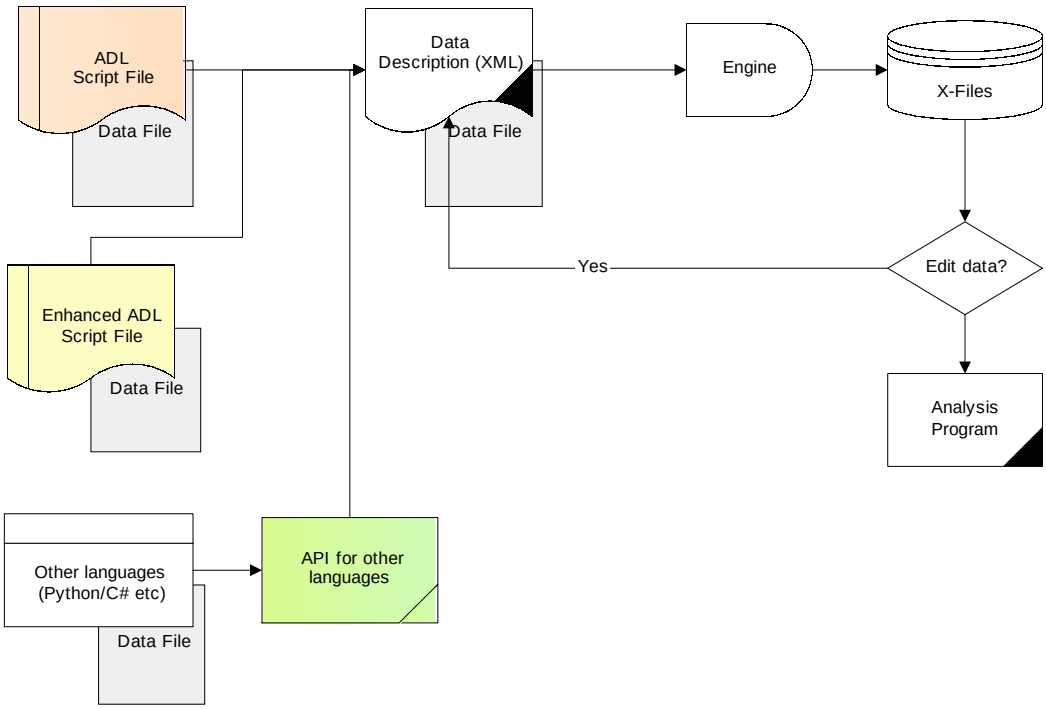
### 5.5.5 Future Ideas

One of the improvements to this application is to give it the ability to parse a traditional ADL file as well as an XML formatted ADL file. So the modification done through this interface is reflected in the ADL file simultaneously. This improves the conversion between the new and traditional formats of an ADL file as well.

The imported macros are not too well handles by ADL_CS editor yet. Future development should link the macro selection to element part of a particular axis.

Ideally, ADL_CS will have its own manual, User Guide or Help system in compiled HTML version.

Enhance this editor to make it capable for handling various ways of generating ADL file. ADL syntax is parsed completely by this application. This conforms to the ideal model of database set-up process:


ADL
Script File
Data File
Data
Description (XML)
Data File
Engine
X-Files
Yes
Edit data?
Enhanced ADL
Script File
Data File
Analysis
Program
Other languages
(Python/C# etc)
Data File
API for other
languages

# 6. Conclusion

## Status of Completion:

| Month | Objective |
|---|---|
| March | Training |
| April | ADL language documentation |
| May | Design an enhanced ADL language |
| June | Implementation plan for the enhanced language design |
| July | Program development stage 1: Re-write ADL in other programming language (C# .NET) |
| August | Program development stage 2: Re-write ADL continues |
| September | Program development stage 3: XML to X-Files convert engine. |
| October | Final report and presentation |

The project is finished following the schedule step by step. Looking back at the requirements specified at the start of this project, and check to see ***how well the project results meet the requirements***:

**1) Training**
Training process is much simpler than before. New staffs do not need training on understanding and familiarizing with the traditional ADL syntax. Even people with no programming background can start building ADL files easily, as long as he has basic windows explorer knowledge.

**2) Speed**
I had an attempt on improving the speed of ADL data file processing. Although the experiment result was negative, it proved the original ADL's great advantage in data processing efficiency.

**3) Correctness**
The correctness of ADL files is improved. By using the ADL_CS editor, many property settings are preset, so user can only choose from the given values. The editor writes out traditional ADL syntax for user, so the completeness of a file and the axes included are enhanced as well.

**4) Easy to maintain and extend to meet new challenges.**
XML format of ADL file provides great **Flexibility** for future development and will be easily converted among similar type of scripting languages, e.g. Triple-s.
The "Macro List" XML file is maintained separately and made reusable, this makes the XML formatted ADL more manageable and extensible.

**5) Retain backwards and forwards compatibility**
No tool was made to parse a readily written traditional ADL file, so I one wants to open and modify an ADL file from previous job, he/she must manually convert it to the XML formatted one.
For forward compatibility, if any changes taken place to the ADL specification, the current XML formatted ADL file parser must be updated in order to handle the newly created settings or symbols. The XML format of the new ADL file must be updated accordingly.

**6) Extra features**
The new ADL format is created according to the original ADL format, and is to be parsed by the original ADL programme for data processing, so not too much extra features were added. However, the "Macro List" feature could be considered an extra feature in a way. And by dividing the entire ADL file into two major parts: "Header" & "Body", the file has clearer structure and becomes more manageable.

The project was done in:

- **Computer labs of University of Auckland**: Debug, Documentation, Final Report and Presentation writing.
- **Information Tools Ltd. Auckland office, Milford:** All software coding and testing.
- **Home:** Research, Programme Design & Planning, periodic summary of project progress.

Both technical and practical knowledge were developed during this period. In particular, I appreciate being given the chance to attend the ITL software development team's weekly meeting and been given time to report my project progress in the meeting.

Technical knowledge gained

- File type conversion
- XML knowledge
- Language parsing
- Documentation skill
- Language Design concepts

- Practical knowledge gained
  - Research technique
  - Document writing
  - Report writing

- Presentation skills
- Team work
- Communication skills
- Business Rules (business workflow, terms of confidentiality)

The experiences I gained through this one-year project are extremely valuable and will definitely be useful for future work. I treasure the technical knowledge gained, but more importantly, the practical knowledge about real world company environment and standard workflow.

# 7. Reference

## *Information Tools Ltd:*

| ***Document Name*** | ***Author*** | ***Date*** |
|---|---|---|
| ADL V3.1m Manual | ITL | 04/2002 |
| ADL V3.3m Manual | ITL | 07/2003 |
| "ADL" Resource CD | ITL | 18/03/2003 |
| ITL TIF Proposal document | Grant Black. | 05/03/2003 |
| "X-Files" presentation doc | Dr. S Manoharan | 14/02/2003 |
| "Discussion Document Extended Character Set Databases" | Grant Black | 19/09/2003 |

## *Internet:*

| ***Document Name*** | ***Address*** |
|---|---|
| XML on-line tutorial | *http://www.w3schools.com* |
| "The BNF Web Club Language SQL, ADA, JAVA, MODULA2, PL/SQL…" | *http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html* |
| On-line forum discussing topics on XML | *http://slashdot.org/comments.pl?sid=57483&cid=0 &pid=0&startat=&threshold=1 &mode=thread&commentsort=3&op=Change* |
| 5 questions about language design | *http://www.paulgraham.com/langdes.html* |
| "Creating A Great Design Document" | *http://www.geocities.com/SiliconValley/Bay/2535/design_doc.html* |
| "Implementation Plan for Web Accessibility" | *http://www.w3.org/WAI/EO/Drafts/impl/* |
| "A Strategy for the Successful Implementation of New Systems" | *http://www.fenwicksoftware.com.au/implementation.asp* |
| Triple-S Site | *http://www.triple-s.org/index.htm* |
| "Error in Design view of Visual Studio .Net" | *http://www.dotnet247.com/247reference/msgs/19/98058.aspx* |

## *University of Auckland:*

| | |
|---|---|
| COMPSCI734 Resource CD set | VS. NET, MSDN |
| COMPSCI330 lecture notes | Semester one, 2003 |

# Appendix A: What is ADL?

What does ADL stand for? Nothing, it is just the tool that is used by ITL.
If really keen on how ADL was named, well, it could be anything, **A**nalytical **D**ata **L**anguage, **A**dvanced **D**ata **L**anguage …you name it!

ADL consists of two parts:

- ADL Program: a single Windows 32 command line driven executable
- ADL language: a data description language

***ADL program:*** Internally, it consists of language syntax parser 'front end' and backend that writes ITL databases using a “blob manager”. The ADL program is sometimes described as a “compiler”, which technically it is not, as it does not produce executable code. It does however share some features of a language compiler such as a C++ or the TAWK compiler in that it parses a language and generates syntax errors/warnings at both parse ‘compile’ time and when building ITL databases (run time).

***ADL language:*** The ADL language is much more difficult to describe. It is not a full programming language, but is more like XML in that it essentially describes or defines data. Unlike XML, an ADL script can describe not only the location and size of data located in a separate data file, but also manipulate or construct data.

For instance, in the ADL language, the line:

```
; Q22
        ["Gender” where=195_2
        "Male"              =1
        "Female"            =2
        ]
```

Specifies the data that indicates the respondent’s gender, is stored in columns 195 and 196