

“Generic Online Reporting”

Robert Marks (98421042)

BSc Computing Science (Hons)

University Of Ulster, Jordanstown

Final Year Project

CONTENTS

CONTENTS	2
ABSTRACT	5
KEYWORDS.....	5
ACKNOWLEDGEMENTS	5
1 – INTRODUCTION.....	6
1.1 - PROJECT INTRODUCTION	6
1.2 - PROJECT AIM	6
1.3 - PROJECT OBJECTIVES	6
2 – REVIEW.....	7
2.1 - INTRODUCTION	7
2.2 - DATA REPORTING	7
2.2.1 - Available Reporting Technology.....	8
2.2.2 - Online Reporting.....	8
2.2.3 - HTML	9
2.3 - INTERNATIONALISATION	9
2.4 – METHODOLOGIES	11
3 – REQUIREMENTS.....	12
3.1 - INTRODUCTION	12
3.2 - USERS	12
3.3 - FUNCTIONAL REQUIREMENTS.....	13
3.4 - NON-FUNCTIONAL REQUIREMENTS	13
4 - HIGH LEVEL DESIGN	15
5 – DETAILED DESIGN.....	16
5.1 - INTRODUCTION	16
5.2 - INTERFACE DESIGN	16
5.2.1 – Site Navigation	18
5.2.2 - Logon Screen	18
5.2.3 - Database.....	19
5.2.4 - Database - Add.....	19
5.2.5 - Database - Open	20
5.2.6 - Database - Delete.....	21
5.2.7 - Report	22
5.2.8 - Report - New	22
5.2.9 - Report - Open.....	22
5.2.10 - Report - Edit.....	23
5.2.11 - Report - Edit - Tables & Aliases	24
5.2.12 - Report - Edit - Relationships.....	24
5.2.13 - Report - Edit - Query.....	26
5.2.14 - Report - Edit - Groups	27

5.2.15 - Report - Edit - Test	28
5.2.16 - Report - Edit - Labels	28
5.2.17 - Report - Edit - Format	29
5.2.18 - Report - Run	35
5.2.19 - Language	36
5.2.20 - Help	36
5.2.21 - Overall Interface Design	37
5.3 - DATA STRUCTURE DESIGN	38
5.3.1 - Database Connection File (*.dbd)	38
5.3.2 - Report File (*.gor)	39
5.3.3 - label_locales.dat	40
5.3.4 - interface_locales.dat	40
5.3.5 - users.dat	41
5.4 - COMPONENT DESIGN	42
5.4.1 - Data Classes	42
5.4.2 - Functional Classes	44
5.4.3 - Interfaces Classes	44
5.4.3.1 – Class diagram of Entire System	46
5.5 - ALGORITHM DESIGN	48
5.5.1 - Query Generator Algorithm	48
5.5.2 - ReportGenerator Algorithm	53
5.6 - CONCLUSION	56
6 - IMPLEMENTATION	57
6.1 - INTRODUCTION	57
6.2 - JAVA & JAVA SERVLETS	57
6.3 - HARDWARE USED	57
6.4 - SOFTWARE USED	58
6.5 - IMPLEMENTATION STRATEGY	60
6.6 - PROBLEMS ENCOUNTERED DURING IMPLEMENTATION	67
6.7 - CONCLUSION	68
7 - TESTING	69
7.1 - INTRODUCTION	69
7.2 - TESTING OBJECTIVES	69
7.3 - TESTING STRATEGIES	69
7.3.1 - White Box Testing	69
7.3.1 - Black Box Testing	70
7.4 - UNIT TESTING	70
7.5 - INTEGRATION TESTING	71
7.6 - FULL SYSTEM TESTING	72
7.7 - CONCLUSION	73
8 - EVALUATION	74
8.1 - INTRODUCTION	74
8.2 - USER EVALUATION	74
8.3 - DEVELOPERS EVALUATION	76
8.4 - ENHANCEMENTS AND RECOMMENDATIONS	77

8.5 - CONCLUSION	80
9 - CONCLUSION	81
9 - PROJECT SUMMARY	81
9.1 - REVIEW OF AIMS AND ACHIEVEMENTS.....	81
9.2 - AUTHORS FINAL NOTE	81
10 - BIBLIOGRAPHY	83
APPENDIX A – FORMAT INTERFACE DESIGN SCREENSHOTS.....	84
APPENDIX B – FORMAT8.HTM	91
APPENDIX C – GRAPHICS	103
APPENDIX D – EVALUATION GUIDE	105
APPENDIX E – COMPLETED EVALUATION FORMS.....	108
APPENDIX F – ACCOMPANYING CD.....	109

NOTE:

The remainder of the document contains the source code to the system.

Abstract

“Generic Online Reporting” is the name of this project. This is the author’s own proposal. Its aim is to create a highly generic, fully online, database reporting system available from a standard Internet browser. It will address areas like multiple database support, internationalisation, portability and ease of use. This project will use a structured lifecycle to investigate, analyse, design, implement, test and evaluate.

Keywords

Generic, online, reporting, Internet, multiple, web, database, language, browser, internationalisation.

Acknowledgements

I would like to thank the following for their help on this project:

- To the members of my PSG (Project Support Group) for their help.
- To Don McFall, who is my project supervisor and has helped me get started and who has given much sound advice in many different areas of this project.
- To all my friends for their critical advice and support throughout the project.
- To Emma Kelly for persuading me to buy a new laptop which has proved invaluable.
- To Jean Lacroix for aiding translation.

1 – INTRODUCTION

1.1 - Project Introduction

This aim of this report is to discuss the project idea, research various technologies, draw up a requirements specification, design, implement, test and the system. After this is done an evaluation of the system can be completed.

The name of this project is called “Generic Online Reporting”. The idea for this project came from the need for a more *generic* approach to database reporting. With the emergence of the Internet and the World Wide Web the world has become a much smaller place. Users (business, home etc) now demand multi-database reports from any computer, in various languages and from any where in the world quickly and easily.

To create a more generic reporting system this project will need to address certain issues. For example: to make it an online system, available from standard web browsers and to investigate multi-language technologies where ever possible.

1.2 - Project Aim

To create a highly generic online reporting system that solves problems with existing reporting technology e.g. lack of multiple language support, portability and ease of use for users in today’s online world.

1.3 - Project Objectives

- Research current reporting technologies.
- Choose an appropriate design methodology.
- Perform requirements analysis and create possible use cases.
- Create solid design for the system based on requirements (e.g. need for multi-language support).
- Research various software / hardware solutions that will be suitable.
- Implement the system.
- Thoroughly test the system.
- Evaluate the system.

2 – REVIEW

2.1 - Introduction

The project will need to be researched to ensure that the right techniques / technology are used to complete the project as correctly and accurately as possible. As the project is going to be an online database reporting system this will involve various different technologies.

2.2 - Data Reporting

The world has countless thousands of databases spread over the globe in many different languages and cultures. Successful reporting of this data is an incredibly important part of the business world. There are a wealth of different systems and methods to make this possible.

Reporting of data can mean several different things. It is primarily the retrieval of data from a data source, which is then formatted into a presentable form. These data sources are usually a database but other data sources are available.

The data can be altered before it is formatted onto an output device. This could be a monitor or printer etc. The majority of database reporting systems require the following three steps:

1. Connection to a database.
2. SQL query is run against the database that returns a result-set (table of data).
3. Formatting the data into a report that can then usually be printed out.

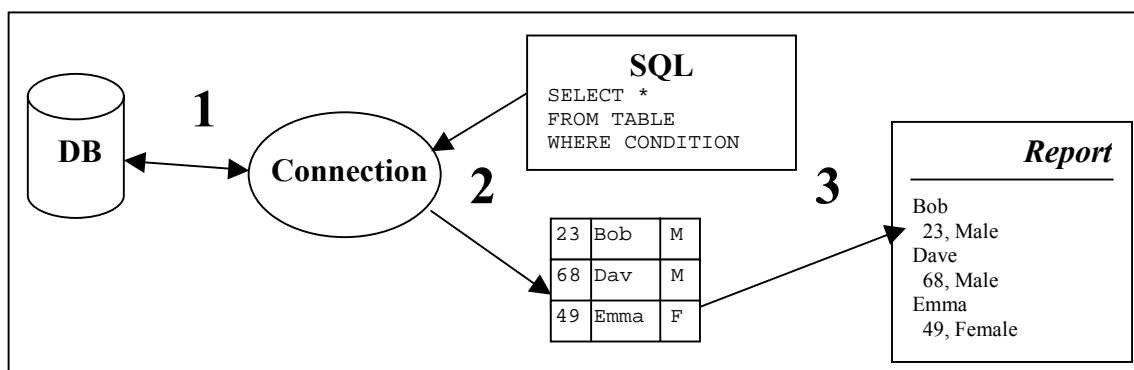


Figure 2.1

2.2.1 - Available Reporting Technology

To implement a reporting system a business must have a database of information to report on. Software is needed to interface with the database and create the final report. These systems can either be built in-house, so the system can be built to a company's exact specification. This is usually a very timely and expensive procedure.

An alternative is to buy/licence a third-party piece of software to do most of the hard work for you. The most famous and popular of these programs is Seagate Crystal Reports. This program can do the following:

- Connect to multiple databases e.g. Oracle, IBM-DB2 etc.
- Ability to query database using a graphical user interface easily and quickly.
- A “design” view is available where data can be formatted using various fonts, lines, labels, pictures, grouping levels etc.
- Ability to view, print, export reports.
- Parameters can be set at run-time to make the system interactive.
- Can be embedded in other programming applications e.g. Visual Basic.

Crystal Reports is however an offline program so it must be installed on a clients computer to create a report. The very latest version of Crystal Reports (8.5) can run reports from the Internet but the reports still have to be created on the client's computer off-line. Presently Crystal Reports is the industry standard for database reporting in the world at present, and it looks like it will remain that way for sometime to come. The creation of reports *online* from a standard Internet Browser is one of the areas that will be addressed to make the system more generic.

2.2.2 - Online Reporting

The massive boom of the Internet was one of the most important things to happen in the 1990's and it continues to change shape all the time. The future of data reporting seems to edge towards this medium. Record amounts of people are accessing the Internet:

“490 million predicated Internet users by year-end 2002 and over 765 million by year-end 2005”

CommerceNet (2001).

In the most recent version of Crystal Reports, the system is tying its reports to the Internet. At present there aren't any *fully online* reporting packages that are commercially available (although Crystal Reports comes close). Currently if a company wants to create an online report they usually have to undertake some degree of server side programming that runs through a web server. Users can only run these reports. Also users have very limited control over these reports (e.g. run-time parameters to redefine a query).

This project aims to be a fully online reporting system based on the client/server model so the various online technologies will be investigated and used. One of the most important of these will be HTML which is the mark-up language for displaying text, graphics that 99% of the Internet consists of.

2.2.3 - HTML

HTML (HyperText Markup Language) is a mark-up language for identifying the elements of a page so that a browser (such as Microsoft Internet Explorer or Netscape's Communicator) can render that page on your computer screen. HTML is going to have a vitally important part in this project, as it will make up the interface to the system. The output of a report will also be in HTML (but other formats may be used).

One of the main problems with using HTML as an interface to a system is the fact that HTML is a static language. This means that once a HTML page has outputted to the screen it doesn't have any interactivity. Some level of interactivity can be added with the use of JavaScript or VBScript. These will be discussed in more detail at the design and implementation stage.

Data from a web page) is transmitted from the Client (Web browser) to the Server through the use of an HTML form (consisting of elements such as textboxes, pull-downs, radio buttons etc). The server then reads the form using Server-Side Programming (e.g. ASP, Java Servlets), processes the form data and outputs an HTML page back to the user. This technique makes up almost all of online systems and this project will be no different.

2.3 - Internationalisation

With the Internet spreading across the globe different languages and cultures have to be catered for. This is one of the most important areas of concern in my project. For example, if a company wishes to create reports in 6 different languages they may require 4 different personnel

(who know these 6 languages between them) to create them. The user-interface to the system must be multi-language compatible too, so the users can choose their preferred language on screen.

In computer systems there are various areas that must be addressed for a system to be fully internationalised. These include:

- **Locale** – A locale is made up a specific language and a geographic region.
 - **Language** – Specified by a language code. This is a standard coding convention specified by the ISO 639 standard.
 - **Country** – Also specified by a 2-letter country code. This is also an international standard ISO 3166.

These are combined to give a particular Locale anywhere in the world.

- **Numbers** – These can be represented differently. E.g. 184: America – one hundred and eighty four. France = one hundred, four twenties and four.
- **Currency** – Each locale has currency symbols, negative amount format, leading zeros, group separators, decimal point symbol and currency symbol position. E.g. currency symbols: American = \$, UK = £, etc.
- **Date** – These can be displayed differently e.g. US = mm/dd/yyyy, UK=dd/mm/yyyy.
- **Character sets** – Some countries need two bytes (e.g. China) to represent text. This can be achieved using Unicode (supported by Java).
- **Layout managers** – These are quite important in international applications because text length differences may vary depending on which language is selected. The screen layout manager must be able to adjust to these differing sizes.

Research has been carried out on various programming languages for implementing internationalisation. Java seems to be the best language that can satisfy all of the previous points using standard libraries. Java also includes a standard class called ResourceBundle. It provides a mechanism for separating user interface elements and other locale-sensitive data from the application logic in the program. It does this through the use of *codes*. Elements like text, buttons, icons, and menus can all be replaced using these codes. A properties file is read which matches the codes up to the correct text for a specific locale.

e.g. Two Resource bundle properties files (English and French) are used to make an application's interface language independent (note the use of codes in their filenames).

MainPage_en_GB.properties	MainPage_fr_FR.properties
Title=Welcome Text1=My name is Robert	Title=Bienvenue Text1=Je m'appel Robert

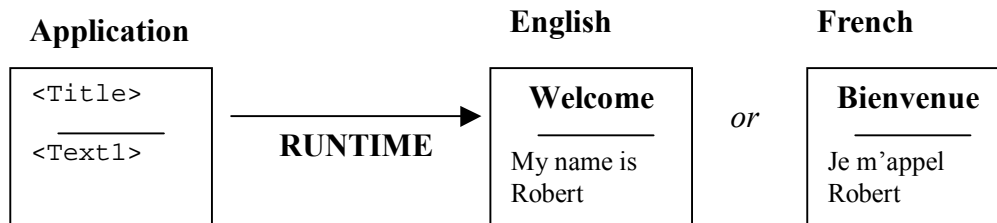


Figure 2.2

2.4 – Methodologies

Various methodologies have been researched for their suitability for this project. These include:

- **Waterfall approach** - separates various activities such as requirements, design etc and represents them as separate phases which usually done in a sequential order
- **Evolutionary development** – an initial system is rapidly developed from very abstract specifications. This is then refined with customer input to produce a system, which satisfies the customer's needs. Alternatively, it may be re-implemented using a more structured approach to produce a more robust and maintainable system.
- **Formal transformation** – this is based on producing a formal mathematical system specification, and transforming this specification, using mathematical methods, to a program. These transformations are 'correctness-preserving'. This means that you can be sure that the developed program meets its specification.

The methodology that is being used to undertake this project is the "Waterfall method". This is sometimes called the linear sequential as each part is done in a sequential order. This type of methodology has problems, but each of the part represents real work that must be done and it provides a very solid structure for carrying out this project successfully.

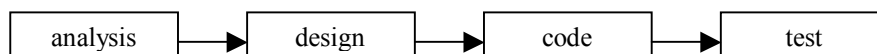


Figure 2.3 - Pressman (2001)

3 – *REQUIREMENTS*

3.1 - Introduction

This chapter is concerned with the requirements of the system.

The system will be a fully online system giving various users the ability to create/run online reports from any database source that the system can connect to. The system will run from a Web Server and be accessed by anyone using a Web browser (e.g. Internet Explorer). The system interface will be available in various different languages. Reports can be created in multiple languages if required. The system should be relatively easy to embed into a different online web systems. These should ensure that the system is as generic as possible.

3.2 - Users

There will be about four main types of users using the system. They are as follows:

- **Report viewer** – These will make up the vast majority of users on the system They are as follows:
 1. **Casual:** A casual user will only be interested in seeing a report being displayed. This user could be from a company website that links a report, that has been generated using the system. They will generally not know the existence of the system, but may still want to view reports in various different languages.
 2. **Non-Casual:** The difference between a casual and non-casual user is that a non-casual user knows about the existence of the system but may not necessarily know how to use it. Therefore he may get someone else to create a report specifically for him or herself. They would generally have a brief understanding of the data that the report is based on. E.g. sales personnel in a company.
- **Report editor** – These users will be responsible for the creation and editing of a report. They will need to be skilled in basic database skills – understanding of how a particular database is made up and also the ability to perform query-by-example queries (skills similar to what is needed to create a query in Microsoft Access visually). They will not require any SQL skills. Like report viewers they may still want to view reports.
- **Systems administrators** – These users will be responsible for installing the reporting system on a server. They will be expected to know a little about web servers and installing server side programs.

- **Database administrators** – Without databases then the system has no data to create reports. Database administrators will be responsible for connecting databases to the system. Hopefully this should be an easy task to do. Therefore a report editor would be able to do this, if he has sufficient database access privileges.

3.3 - Functional Requirements

The following functional requirements for the system describe the functionality or services that the system is expected to have (based on the user requirements):

1. Users should have the ability to log onto the system using a browser by providing a user name and password as identification.
2. Users must have to ability to create complex queries for the report data visually by using html forms (No experience of SQL necessary).
3. Multiple database connections may be added from within the system.
4. Users should be able create, save, open and run reports quickly and easily from within the G.U.I.
5. Users must be able to specify the graphical user interface in various different languages e.g. English, French, German etc.
6. Users must be able to create report labels in various different languages.
7. Provide limited report interaction for users using run-time parameters so they are able to redefine a query.
8. Reports must be viewable as HTML. Like many pages that exist they must be print-friendly if a user wishes to have a hardcopy. Other formats may be considered if there is adequate time (e.g. X.M.L., P.D.F.).
9. Users must have a limited form of help.

3.4 - Non-Functional Requirements

Non-functional requirements are requirements that the system must have, based on a range of software development principles. These requirements may not be directly concerned with the specific functions delivered by the system:

1. **Flexibility** - The system should be flexible. Various items have been previously stated that will make the system more flexible: such as multiple language support and the ability to run a report from a different system. The user may also want an acceptable level of flexibility in the creation of a report.

2. **Choice of Input Methods** - The choice of inputs methods will be very limited, but as in all web systems the mouse is the primary choice of input. The keyboard will be used as input where required, e.g. Textboxes. The availability of input methods is dependant upon the browser in operation. Microsoft Internet Explorer supports keyboard-only input. This is achieved using the tab and enter key in the movement and selection of G.U.I. elements. This frees the developer from certain G.U.I. worries.
3. **Choice of Output Modalities** - The user should have a choice of how the system controls output. This may include different colour schemes for the system but this will only be added if there is sufficient time, as it is not a necessity.
4. **Consistency** - The system should be consistent, so as not to confuse the user. This involves keeping colours, fonts and screen layout similar and coherent. For example, in some websites a user may be greeted with many different colour schemes / graphics to grab their attention. This can affect the usability of the system.

4 - HIGH LEVEL DESIGN

The software design of this project is a description of the structure to be implemented, the data which is part of the system, the interfaces between the system components and the algorithms. An architectural design was done of the entire design to show how the relationships between each part (Figure 4.1).

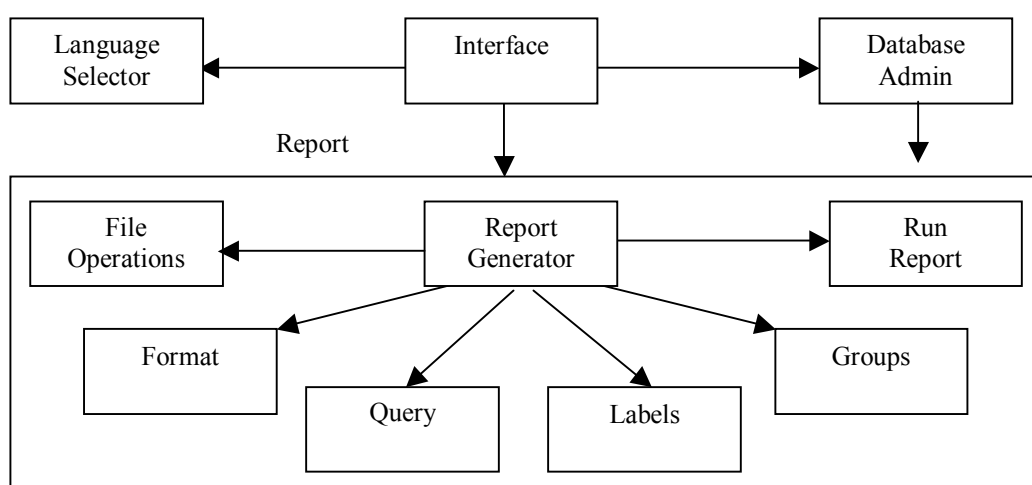


Figure 4.1

The following table shows a brief description of each section and it's purpose.

Sub system	Brief Description
Interface	This is what the user sees and interacts with. It will provide links to other parts of the system. The system is web based and so this will be HTML based.
Language Selector	Provides the user the option to change the desired language of the interface.
Database Admin	Should provide facilities to add, open or delete a database connection.
Report Generator	Creates an online report.
File Operations	Sub system of report that concentrates with report file operations like creating a new report, opening and deleting existing reports.
Query	Creates a query visually (using html forms). This query is run against the database, which will return data and will make up the report.
Labels	Multi-language report labels that match up with data columns. Users will have the ability to specify these in various different languages.
Groups	Categorizes a database query into sub groups for presentation purposes.
Format	Will provide simple formatting of a report, e.g. fonts, sizes, colours etc.
Run Report	Will provide a method of running a report from the system.

5 – *DETAILED DESIGN*

5.1 - Introduction

As this system is fairly large and complex this chapter will be quite large. It follows on from the design chapter in the “End Of Semester 1 Report”. There will be four main areas of design in the system. They are as follows:

1. **Interface Design** - As this is a web system, the entire interface will be in HTML. The full system interface and each individual screen in the system will be created.
2. **Data Structure Design** - The system will create and use various files to store database connection data, report data etc. All these files will be stored as simple text files and will be described in depth.
3. **Component Design** - This system will be an object-orientated system. The reasons for this will be discussed. All the classes will be identified and their uses in the system.
4. **Algorithm Design** - Any algorithms, which the system will use, will be discussed in detail here.

This chapter starts by showing all the interface screens of the system in HTML, various component designs, data structure design central to the system and also some detailed algorithms that the system will require.

5.2 - Interface Design

The next stage of the design process for this project is the interface design. To create the user interface for the whole system, each individual screen was prototyped and a final HTML version was created. Most screens were constructed successfully using this method with only one or two iterations to get the final user interface. These were sketched first and then produced in HTML with possible JavaScript (client side interaction). The only exception was the Format interface which is used to format a report with various fonts, colours, lines etc. This was prototyped almost entirely in HTML with very little sketching. It took eight different iterations to get right (see Appendix A for a description of the eight different screens).

(Author’s Note: I found the best way to create the majority of interface screens quickly was with a pencil and a sketchbook. I simply sketched various different designs and picked the most appropriate based on various design principles and web form constraints. Diagram 5.1 shows

the designing of the query section of the system in which I used some of the ideas from the query section of Microsoft Access. Diagram 5.2 shows the finished result in HTML.

① *Show column*

S	Tables & Columns	Sort	Con	Value
<input checked="" type="checkbox"/>	EMP <input checked="" type="checkbox"/> EMP10 <input checked="" type="checkbox"/>	ASC <input checked="" type="checkbox"/>	= <input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	= <input checked="" type="checkbox"/>	= <input checked="" type="checkbox"/>	
<input type="checkbox"/>				

List of table *List of table columns* *test box for value*

—, ASC, DESC =, <, <=, >, >=

↓ — This way (Vertical) is better for browsers as they generally scroll vertically.

② *or*

S	Tables & Columns	Sort	Con	Value
<input checked="" type="checkbox"/>	EMP <input checked="" type="checkbox"/>	DESC <input checked="" type="checkbox"/>	= <input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>	EMP10 <input checked="" type="checkbox"/>	= <input checked="" type="checkbox"/>	= <input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/>				
<input checked="" type="checkbox"/>				
<input checked="" type="checkbox"/>				

— Don't think this way is as good (MS Access) as it scrolls ~~vertically~~ horizontally.

— Going to me ①

Figure 5.1

S	Aliases And Columns		Sort	Con	Value
<input checked="" type="checkbox"/>	Products	ProductID	ASC	-	
<input checked="" type="checkbox"/>	Products	ProductName	-	-	
<input checked="" type="checkbox"/>	Category	CategoryName	-	-	
<input type="checkbox"/>	Products	UnitsInStock	-	>	0
<input type="checkbox"/>	-	-	-	-	

OK

Figure 5.2

5.2.1 – Site Navigation

Easy and understandable navigation of any web system is critical. It was decided that the system was going to be divided into five main sections, including some that have sub systems. These are as follows:

1. **Home section** - This will be a single screen welcoming the user to the system.
2. **Report section** - This section will be concerned with creating, opening, deleting, editing and running a report.
3. **Database section** - This section is concerned with creating, opening and deleting a database connection file.
4. **Language section** - This will be a single screen responsible for changing the current language of the interface.
5. **Help section** - This section will provide compressive help to the user.

Each of these will be accessed through a menu system at the top of the screen. The structure of the menu system will look something like the following diagram. (Figure 5.3)

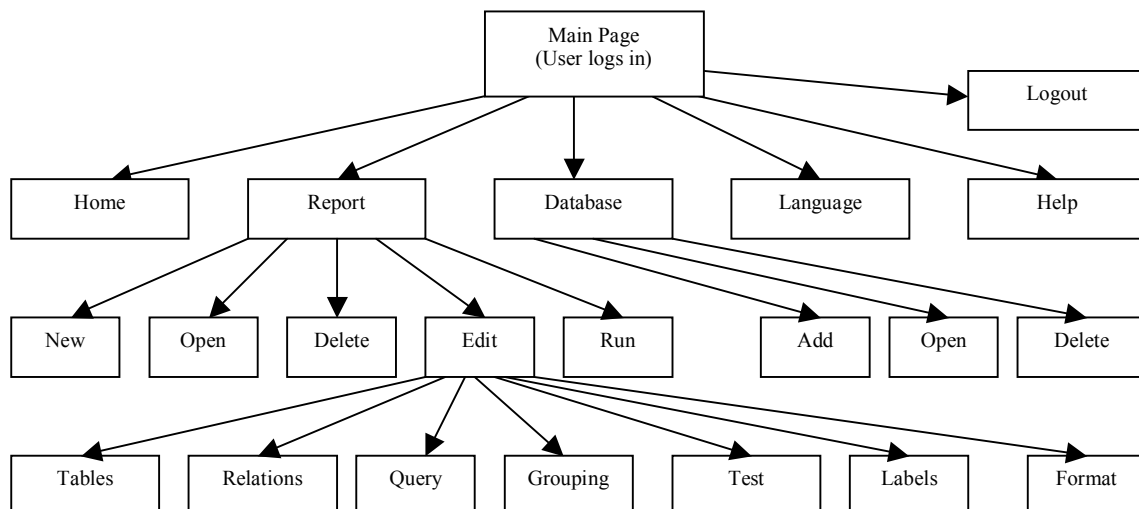


Figure 5.3

There will be a screen to log on and once a user has successfully logged on, there will be the option to log off again. Each interface in the system will be explained in more detail.

5.2.2 - Logon Screen

This is the first screen that a user will meet in the system. It is a very simple screen, which consists of a standard username and password textbox (see Figure 5.4).



Logon	
User Name	<input type="text" value="user"/>
Password	<input type="password" value="*****"/>
<input type="button" value="OK"/>	

Figure 5.4

Once a user has entered the system the system must take note that they are logged in. This can be achieved in a web system using cookies or session variables. The first screen a user will see is the home page. This will welcome the user and briefly explain the operation of the system.

This was the very first HTML interface made for the system. Careful consideration was made to its fonts (Veranda size 2), colours (navy and greys) and structure (position of buttons, borders, table cell spacing and padding). This was to ensure that the entire system interface was kept consistent and professional looking.

5.2.3 - Database

Before a report can be made a database connection must be established. This section of the system is concerned with its operation. It consists of 3 parts:

1. Adding a database connection (creates a small text file).
2. Opening an existing database connection (opens an existing database connection file).
3. Deleting a database connection (deletes an existing database connection file).

Before each screen was created in the database section, the information that a database connection file would store was recognised. This information includes the name of the database, drivers required to connect to a database (e.g. ODBC), connection URL (location of database), optional username and password (database security).

5.2.4 - Database - Add

This screen will provide a user with a means to create a database connection file. Information a user must complete in this HTML form include:

- The name of the file.
- The database type e.g. Access, mySQL, Oracle etc.
- Drivers that the system currently supports (standard ones include the ODBC).

- Connection URL. A user can pick from a list of predefined connections (to make this easy) or he can specify a connection else where (this could ever be on a different computer)
- The name of the connection.
- Optional username and passwords (if the database connection requires these).

Figure 5.5

See Figure 5.5 for interface design for this screen. Once this form is submitted to the Server, the system will create a database meta file with the above information stored in it. Additional information will be retrieved from the database and stored such too. This will be table and column information that the Report - Edit section of the system requires for its interface.

5.2.5 - Database - Open

This screen will provide a user access to the database connection file, which the previous section creates (Database – Add).

The first thing the interface will do is to list all the available database connection files on the server. A user simply can then select one of these from the list, click OK, and another screen will display. This screen will display all the information from the connection file. This includes connection information (entered from the previous interface) and also tables and columns in the database. This will be achieved using two HTML pull-downs. If the user clicks on a table in the first pull-down, then the respective columns that table is made up of should display on the other pull-down. This client side interaction is achieved using JavaScript and will be used throughout the system.

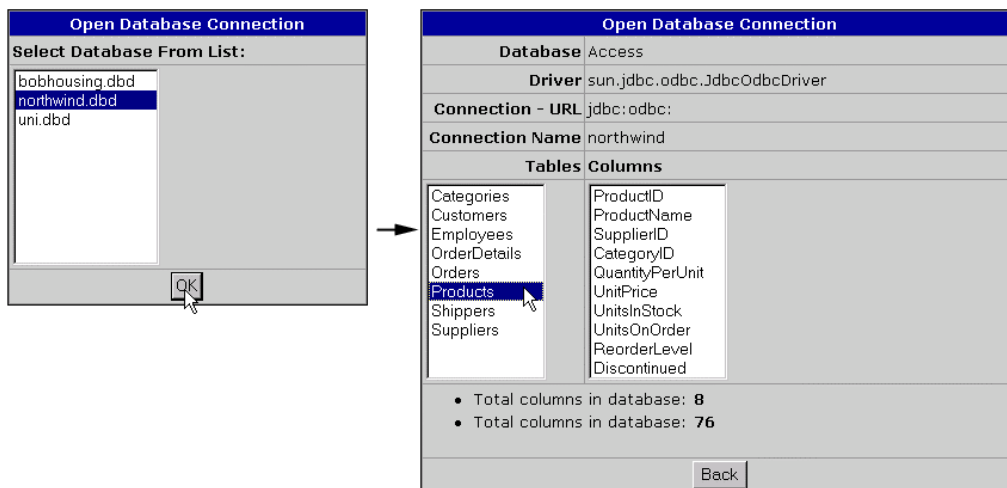


Figure 5.6

Figure 5.6 shows the user choosing a database connection file and the respective screen coming up and displaying its information.

5.2.6 - Database - Delete

This section of the system is very similar to the first screen of the open database section mentioned previously. It consists of a list of available database connection files. The user must select one from the list and also click on the “Yes” radio button to show that he is sure that he/she wants to delete it (Figure 5.7).

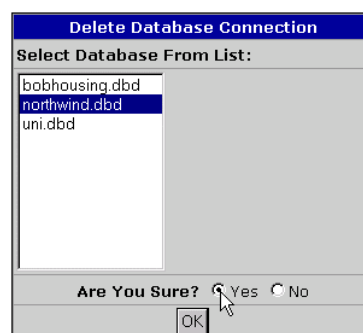


Figure 5.7

As this is deleting a file off the system, some kind of security features will need to be implemented such as permissions will need to be enforced here.

5.2.7 - Report

The report section of the system is much more complex than the database connection section and consists of many various parts (see section 5.3). Functionality that the report section of the system must include are creating, opening, editing, running and deleting report files. Of these the editing section will have more sub-sections for creating a query, adding labels and formatting the report.

5.2.8 - Report - New

To create a new report, a user must enter the name of the report and select a database connection file (see Figure 5.8). Once this form is submitted the system should create a report file with minimal information. (E.g. name of database connection file, date created, user name etc).

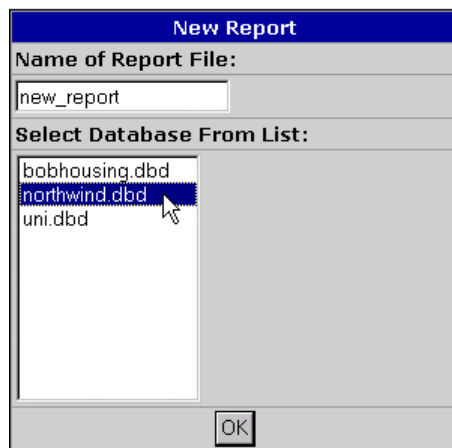


Figure 5.8

When a new file is created it must be edited. The system must remember that this file is currently open and that it is open for editing. This can be implemented in a web system using sessions.

5.2.9 - Report - Open

This screen displays a list of all the available report files. Most users may only wish to run a report (e.g. Managers), but they may also wish to edit a report. If a user selects the edit option then the file will be locked so that other users cannot edit it at the same time (Figure 5.9).

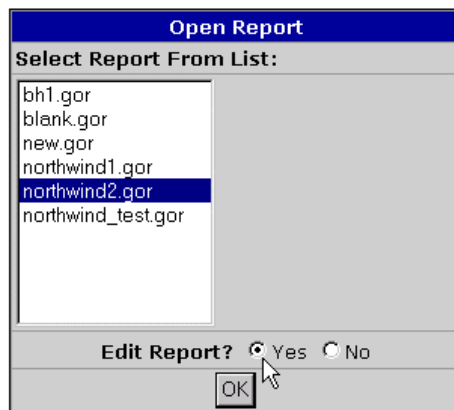


Figure 5.9

5.2.10 - Report - Edit

The Report - Edit section of the system will be the most complex part of the whole system and includes the following sub sections:

1. **Tables** – To create a query one or more tables must be declared. This section adds tables.
2. **Relationships** – If a query contains more than one table then relationships must be declared between the tables (joins).
3. **Query** – This section is where the user specifies which columns in the query and any conditions.
4. **Groups** – Columns in the query can be grouped which will be used to visually organise the data into groups.
5. **Test** – Runs the query against the database and if successful should display a table of data (result set).
6. **Labels** – This section is used to specify labels such as the title, subtitle and footer text of a report. Also data labels can be specified (e.g. Name, ID etc). More than one set of labels can specified in various languages.
7. **Format** – Used to format a report with various fonts, font sizes, **COLOURS**,

Bold, *Italics*, Underline, and horizontal lines used to organise data.

5.2.11 - Report - Edit - Tables & Aliases

The first thing a user must do to create a query is to specify what tables from the database are going to be used. An alias column is specified for the following reasons:

1. The same table can be used more than once with different aliases declared.

SQL example = `SELECT .. FROM EMPLOYES AS E1, EMPLOYES AS E2`

2. Aliases are also useful for making sense of badly named tables or making long table names shorter.

SQL example = `.. FROM A_UIJKDOCS_BACKUP_TABLE_94_95 AS DOCS_94_95`

Figure 5.10 shows how users will add tables and their aliases to a report. They first select the table from the pull-down list on the left and specify an alias on the textbox on the right. The table names will come from the database connection file.

Table	Alias
Employees	Emp
Shippers	Shippers
-	
Categories	
Customers	
Employees	
OrderDetails	
Orders	
Products	
Shippers	
Suppliers	

OK

Figure 5.10

5.2.12 - Report - Edit - Relationships

If more than one set of tables / aliases are declared then their relationships must be defined. The relationships screen is split into 2 sides, parent and child aliases / columns. The first pull-down of each side is a list of user declared aliases (from the Report – Edit – Tables section). Once an alias is selected its corresponding list of column are displayed. This is done using JavaScript and is very similar to the Database – Open section.

Parent Aliases / Columns		Child Aliases / Columns	
Products	SupplierID	Supplier	SupplierID
Products	CategoryID	Category	CategoryID
-	-	-	-
		<div> <div>CategoryID</div> <div>CategoryName</div> <div>Description</div> <div>Picture</div> </div>	

OK

Figure 5.11

For example: if a user selects the Products alias and the CategoryID column (foreign key) then he must match that up with the Category alias and CategoryID column (primary key). (See Figure 5.11) This is done very easily when creating a query in MS Access, by simply dragging a line from one column in a query table/alias to another. (Figure 5.12) To emulate this ease of use in a web form proved to be too difficult, and the design was abandoned.

Figure 5.12

If a user did not add the Category table then he/she would have only been able to include the Products.CategoryID in a report. If he/she adds the Category table to the query he/she can use all the columns in the Category table like Category.CategoryName, Category.Description etc. These textual fields are more self-explanatory than a simple id / code.

```
SQL Example = SELECT Products.ProductID, Category.CategoryName FROM
Products As Products, Category as Category WHERE Products.CategoryID =
Category.CategoryID
```

5.2.13 - Report - Edit - Query

This is quite a complex interface in the system. It is divided into 5 different sections (Figure 5.13). They are as follows:

1. **S** - This is the equivalent to the show column in MS Access (Figure 5.12). If this checkbox is ticked then that column in question will be displayed in the query.
2. **Aliases And Columns** - This is equivalent to the Table And Fields in MS Access (Figure 5.12). It is identical to the one of the sides in the relationships section. The first pull-down contains a list of aliases and when click brings up its columns in pull-down beside it.
3. **Sort** - This pull-down can be: - (not sorted), ASC (sorted in ascending order) and DESC (sorted in descending order).
4. **Con** - This pull-down (condition) provides a list of condition types a user can specify. They are – (no condition), = (equal to), <> (not equal to), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to).
5. **Value** - If a user specifies a condition then they must also specify a value into this text box. If it is left blank this means that a column must be equal (or not equal) to nothing (blank). The Con and Value is similar to the Criteria textbox in MS Access, although this is more restrictive e.g. Access has Like and Between operations.

S	Aliases And Columns	Sort	Con	Value
<input checked="" type="checkbox"/>	Products ProductID	ASC	-	
<input checked="" type="checkbox"/>	Products ProductName	-	-	
<input checked="" type="checkbox"/>	Category CategoryName	-	-	
<input type="checkbox"/>	Products UnitsInStock	-	>	0
<input type="checkbox"/>	- -	-	-	

Figure 5.13

```
SQL Example = SELECT Products.ProductID, Products.ProductName,
Category.CategoryName FROM Tables As Alasies, .. WHERE Joins AND
Products.UnitsInStock > 0 ORDER BY Products.ProductID
```

5.2.14 - Report - Edit - Groups

Once a query has been formed (at least 1 set of tables and aliases are declared and at least 1 visible column in the Query section) then groups can be set up. If there are many instances of the same element in a column, this element can be grouped under a sub heading. Generally this makes a report look more structured, tidier and easier to read (see Figure 5.14).

Report 1 - No Groups

Product ID	Product Name	Category Name
1	Chai	Beverages
2	Chang	Beverages
24	Guaraná Fantástica	Beverages
34	Sasquatch Ale	Beverages
35	Steeleye Stout	Beverages
38	Côte de Blaye	Beverages
39	Chartreuse verte	Beverages
3	Aniseed Syrup	Condiments
4	Chef Anton's Cajun	Condiments
5	Chef Anton's Gumbo Mix	Condiments
6	Grandma's Boysenberry	Condiments
8	Northwoods Cranberry	Condiments
15	Genen Shouyu	Condiments

Report 2 - Category Name as Group

Category Name: Beverages	
Product ID	Product Name
1	Chai
2	Chang
24	Guaraná Fantástica
34	Sasquatch Ale
35	Steeleye Stout
38	Côte de Blaye
39	Chartreuse verte
Category Name: Condiments	
Product ID	Product Name
3	Aniseed Syrup
4	Chef Anton's Cajun
5	Chef Anton's Gumbo Mix
6	Grandma's Boysenberry
8	Northwoods Cranberry
15	Genen Shouyu

Figure 5.14

To create groups, a user will select a column from a pull-down list of visible columns from the Query Section (Figure 5.15).

Priority	Visible Columns
1	-
2	-
3	Products.ProductID
	Products.ProductName
4	Category.CategoryName
	-
5	-

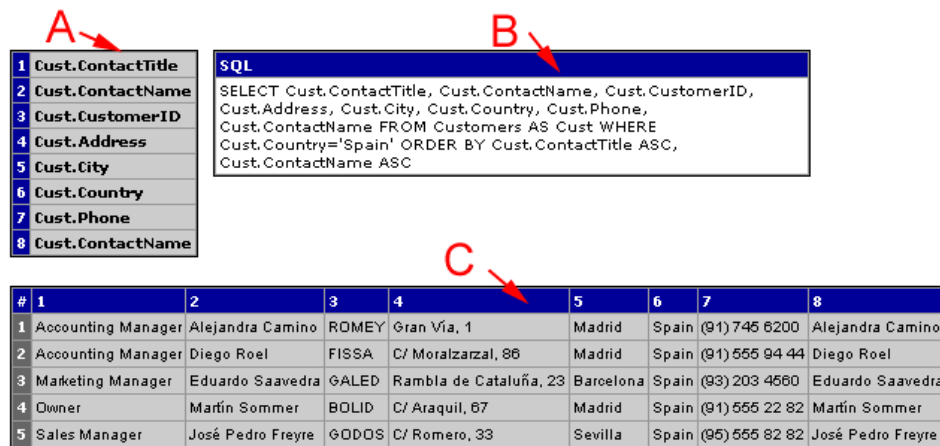
OK

Figure 5.15

A visible column is any column where the S (Show) checkbox is checked. More than one grouping level can be selected, but the order they are picked is important. For example a data column County would get a high priority than Town as there will be many, many more instances of the same county than of the same town. There will be a maximum of 5 groups (There would very rarely ever be a report big enough to have a greater amount of groups.)

5.2.15 - Report - Edit - Test

Once a query has been completed (at least 1 table and at least visible column) then running it against the database can test the query. If there has been an error then the relative error message should be displayed, otherwise something like the following should be displayed (Figure 5.16.):



A →

1	Cust.ContactTitle
2	Cust.ContactName
3	Cust.CustomerID
4	Cust.Address
5	Cust.City
6	Cust.Country
7	Cust.Phone
8	Cust.ContactName

B →

```
SQL
SELECT Cust.ContactTitle, Cust.ContactName, Cust.CustomerID,
Cust.Address, Cust.City, Cust.Country, Cust.Phone,
Cust.ContactName FROM Customers AS Cust WHERE
Cust.Country='Spain' ORDER BY Cust.ContactTitle ASC,
Cust.ContactName ASC
```

C →

#	1	2	3	4	5	6	7	8
1	Accounting Manager	Alejandra Camino	ROMEY	Gran Vía, 1	Madrid	Spain	(91) 745 6200	Alejandra Camino
2	Accounting Manager	Diego Roel	FISSA	C/ Morazarzal, 86	Madrid	Spain	(91) 555 94 44	Diego Roel
3	Marketing Manager	Eduardo Saavedra	GALED	Rambla de Cataluña, 23	Barcelona	Spain	(93) 203 4560	Eduardo Saavedra
4	Owner	Martin Sommer	BOLID	C/ Araquil, 67	Madrid	Spain	(91) 555 22 82	Martin Sommer
5	Sales Manager	José Pedro Freyre	GODDS	C/ Romero, 33	Sevilla	Spain	(95) 555 82 82	José Pedro Freyre

Figure 5.16

The test section shows 3 different parts:

- The first box shows the names of the visible columns in the query.
- This box will show the SQL statement that is created from Tables / Aliases, Relationships and Query section of the system.
- This table shows the result set that the query produces when run against the database. This also shows the data that the report will be made up of.

5.2.16 - Report - Edit - Labels

Once a query has been successfully completed the next step is to specify labels. For example, a user does not want to see “Cust.ContactTitle” (see Figure 5.16) as a label at the top of a table of data. The user would much rather see something like “Title”, or if a user was French he would

rather see “Titre”. A user must be able then to create labels, and also edit existing labels (e.g. needed if a new column was added to the query.) The labels interface is divided into two different screens. The first screen is where the user creates a new set of labels by choosing a locale (language and country) from the list, or edits an existing set of labels he has created (Figure 5.17).

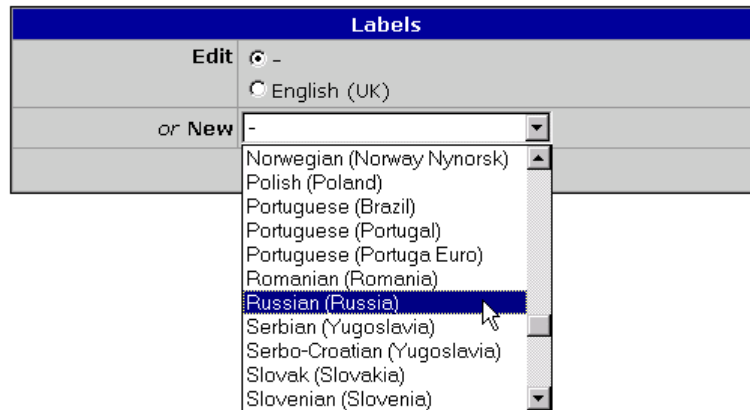


Figure 5.17

Once a user has selected a new language a screen should be displayed where they can specify the title, sub-title, footer and data labels in the language they have picked. Figure 5.18 shows a user specifying the labels of a report in English.

Labels - English (UK)	
Title	Northwind - Employee Reports
Subtitle	The following report shows details for all all current employees.
Footer	http://www.northwind.com
Products.ProductID	ID
Products.ProductName	Product Name
Category.CategoryName	Category
<input type="button" value="Back"/> <input type="button" value="OK"/>	

Figure 5.18

5.2.17 - Report - Edit - Format

This is the last step of creating a report and is concerned with the aesthetics of it. Common ways to brighten up a report (or any document) is to use different fonts, font sizes, colours etc.

As mentioned earlier (see section 5.2), most of the interfaces were simply sketched and then produced in HTML first time. This was because they were fairly straightforward or there was a standard design that many applications shared (e.g. opening a file).

The following screenshot (Figure 5.19) shows a MS Access report from the Northwind database, which comes with MS Access. As you can see it is very complex with images, various fonts and elaborate placement of elements such as labels and data. Emulating an online version of this power / flexibility is beyond the scope of this project, although it will be relatively powerful.

INVOICE

One Portland Way, Twin Rivers WA 98156
Phone: 1-206-555-1417 Fax: 1-206-555-5938

Date: 9-mars-2002

Ship To: Rattlesnake Canyon Grocery
2817 Milton Dr.
Albuquerque NM 87110
USA

Bill To: Rattlesnake Canyon Grocery
2817 Milton Dr.
Albuquerque NM 87110
USA

Order ID:	Customer ID:	Salesperson:	Order Date:	Required Date:	Shipped Date:	Ship Via:
11077	RATTC	Nancy Davolio	06-mai-1998	03-jun-1998		United Package

Product ID:	Product Name:	Quantity:	Unit Price:	Discount:	Extended Price:
2	Chang	24	\$19,00	20%	\$364,80
3	Aniseed Syrup	4	\$10,00	0%	\$40,00
4	Chef Anton's Cajun Seasoning	1	\$22,00	0%	\$22,00

Page: 1 of 1

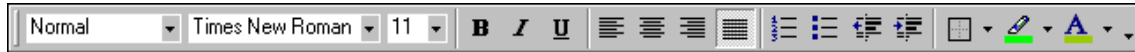
Figure 5.19

Creating the interface to the Format section was not straightforward. First of all, HTML Web forms provide a very limited range of form controls as opposed to e.g. MS Visual Basic, C++ etc (see Figure 5.20).

Textboxes	<input type="text" value="23"/>
Checkboxes	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/>
Radiobuttons	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>
Pulldowns	<input type="text" value="March"/>
Buttons	<input type="button" value="Submit"/> <input type="button" value="Reset"/>

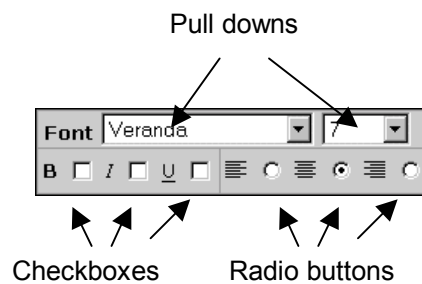
Figure 5.20

When designing the interface one of the issues was recreating some of the formatting elements which programs like MS Word use in their tool bars (Figure 5.21).



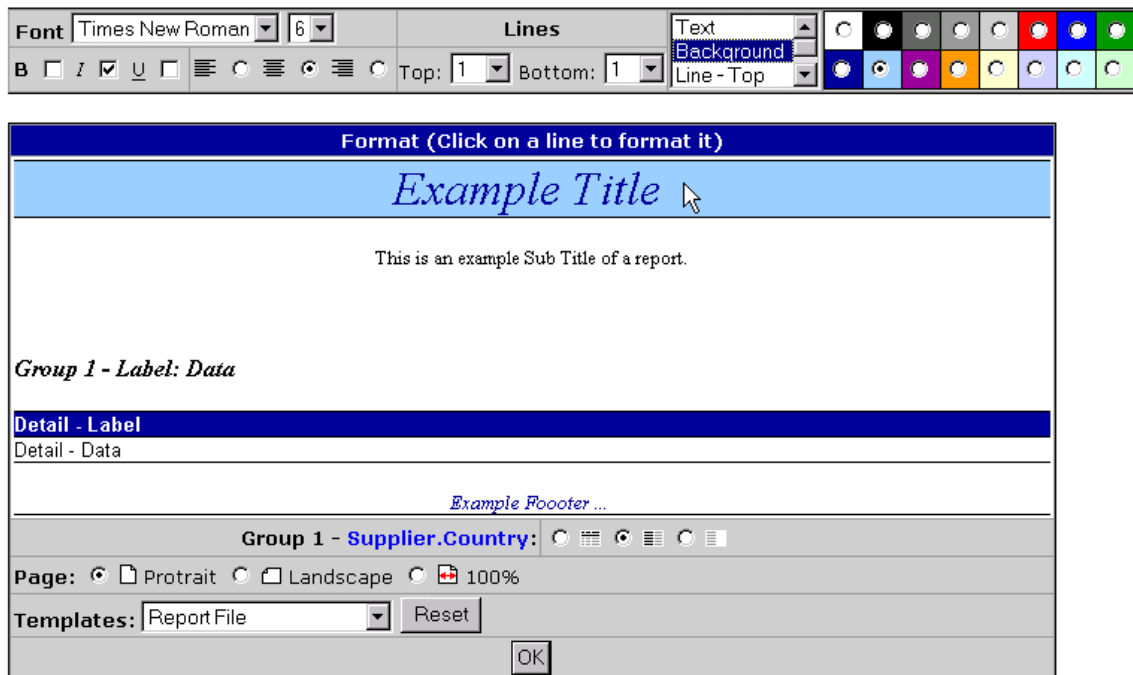
(Figure 5.21)

Emulating certain elements like the bold and alignment buttons in exactly this way proved too complex for a HTML web page. They were done using standard HTML form elements (Figure 5.22).



(Figure 5.22)

In the design of this section an evolutionary approach was taken. With much work and roughly eight iterations later, the following interface was created (Figure 5.23).



(Figure 5.23)

A report will be divided into various sections. They are as follows.

1. Title - short title of report.
2. Subtitle - longer description of report.
3. Report Labels & Data - main part of report.
4. Footer - text that appears at bottom of screen.

Each section can have a line at the top and bottom of it. These lines can be of an exact pixel size or they can be a spacer line (blank row) to space out the text.

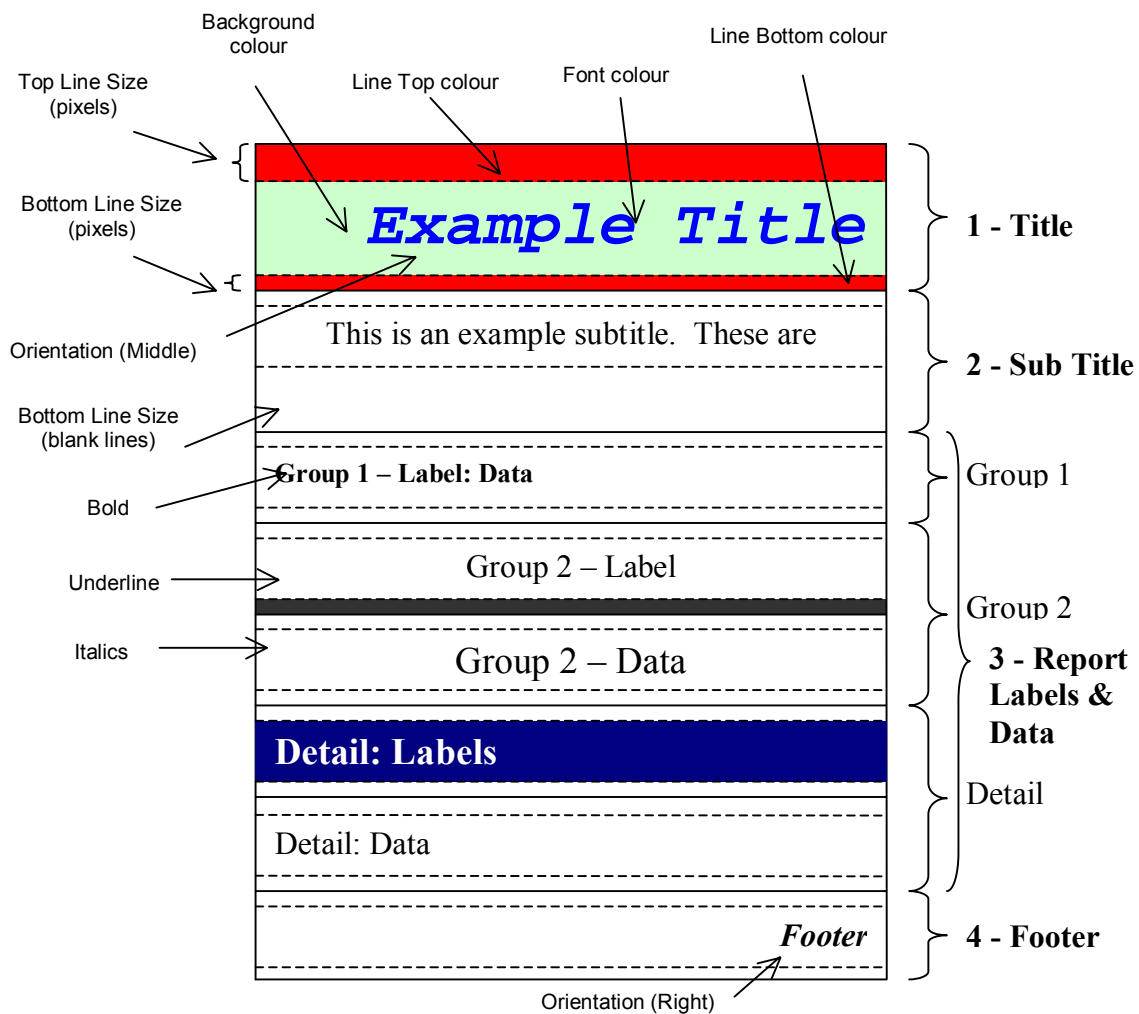


Figure 5.24

Each section of the report can change its format in 12 different ways (There were many other things which could change e.g. images, borders etc, but were omitted to ensure project completion).

1. **Fonts** – These will be HTML fonts e.g. Times new Roman, Arial, etc.

2. **Font size** – These are also HTML font sizes. They will range from 1 to 7.
3. **Bold** – Yes / No
4. **Italics** – Yes / No
5. **Underline** – Yes / No
6. **Align** – Left, Middle, Right
7. **Line Size Top** – 0 = None *or* 1 to 15 pixels *or* 1 to 4 blank lines
8. **Line Size Bottom** – 0 = None *or* 1 to 15 pixels *or* 1 to 4 blank lines
9. **Colour (Text)** – 1 of 16 different colours
10. **Colour (Background)** - 1 of 16 different colours
11. **Colour (Line Top)** - 1 of 16 different colours
12. **Colour (Line Bottom)** - 1 of 16 different colours

The title, subtitle and footer sections will become 1 row each in the report. The report labels and data section is more complex. It consists of a detail labels row and one or more detail data rows (depending on how many rows the query produces). Data can be organised in groups, up to a maximum of 5 groups. Groups can be formatted such as the detail section, in that they have 1 set of labels with data underneath, or the group label and data can be on the same line.

The number of possible formatting rows are as follows: title + subtitle + footer = 3, detail labels and data = 2, 5 group label and data rows = 10. This creates a total of 15 different possible rows that can be formatted. The entire formatting for report can be represented as a 12 x 15 two-dimensional integer array.

Vertical columns in array:

0. Fonts: 0 = Default, 1 = Arial, 2 = Times new Roman, 3 = Courier New, 4 = Georgia, 5 = Veranda,
1. Font size – 0 to 7
2. Bold – No = 0 / Yes = 1
3. Italics – No = 0 / Yes = 1
4. Underline – No = 0 / Yes = 1
5. Align – Left = 0, Middle = 1, Right = 2
6. Line Size Top – 0 = None, 1 to 6 (size of lines in pixels) 1 = 1, 2 = 2, 3 = 4, 4 = 6, 5 = 10, 6 = 15, 7 = None, 8 to 11 (size on lines in blank lines) 8 = 1, ..., 11 = 4.

7. Line Size Bottom – 0 = None, 1 to 6 (size of lines in pixels) 1 = 1, 2 = 2, 3 = 4, 4 = 6, 5 = 10, 6 = 15, 7 = None, 8 to 11 (size on lines in blank lines) 8 = 1, ..., 11 = 4.
8. Colour (Text) - 0 to 15
9. Colour (Background) - 0 to 15
10. Colour (Line Top) - 0 to 15
11. Colour (Line Bottom) - 0 to 15

Horizontal rows in matrix:

0. Title
1. Sub Title
2. Group 1 Labels
3. Group 1 Data
4. Group 2 Labels
5. Group 2 Data
6. Group 3 Labels
7. Group 3 Data
8. Group 4 Labels
9. Group 4 Data
10. Group 5 Labels
11. Group 5 Data
12. Detail Labels
13. Detail Data
14. Footer

It is now relatively straightforward to tell what the alignment of the sub title text is: `format_data [row][element] = format_data [1][5]`.

Other elements in the report that can be formatted are how the groups will be displayed in a report. There are 3 different ways.

1. A group can be displayed like the detail section (i.e. Labels at top with all data underneath)
2. Label followed by data on the same line (Label: Data)
3. Data on its own.

Because there are a maximum of 5 different groups this adds another 5 additional numbers. A final number will have to be added. This is the page orientation of a report. The entire report can either be:

1. Portrait
2. Landscape
3. 100% - fits the entire width of the browser.

This creates a total of $(12 \times 15) + 5 + 1 = 186$. This means a simple list of 186 integer numbers can turn a flat unformatted table of data with labels into a report with lots of colours, fonts, lines etc.

5.2.18 - Report - Run

This is the most important part of the system as this is where the reports are run. A user will be given the choice to run the report full screen or to run it from with the system interface. If a report is run in full screen mode (only the report in the browser), then a report will be able to be printed out. Another option a user can specify is the current language of the labels. The full screen option will be specified as radio-buttons, the language selection as a pull-down and a button for refreshing the screen (see Figure 5.25).

Full Screen: ☐ Yes ☒ No **Language:** English (UK)
 English (UK)
 French (France)
Northwind
The following report shows employee details grouped by title.

Title: Accounting Manager

Name	ID	Address	City	County	Phone
Alejandra Camino	ROMEY	Gran Vía, 1	Madrid	Spain	(91) 745 6200
Bernardo Batista	QUEDE	Rua da Panificadora, 12	Rio de Janeiro	Brazil	(21) 555-4252
Carlos González	LILAS	Carrera 52 con Ave. Bolívar #65-98 Llano Largo	Barquisimeto	Venezuela	(9) 331-6954
Diego Roel	FISSA	C/ Morazarzal, 86	Madrid	Spain	(91) 555 94 44
Elizabeth	BOTTM	22 Tottenham Blvd	Toronto	Canada	(416) 555 4733

Figure 5.25

5.2.19 - Language

This screen will provide the user with the option to change the language of the system interface. Elements that can change include all text on the screen. Dates can also change e.g. UK date = dd/mm/yyyy and US Date = mm/dd/yyyy.

The language selection screen is very straightforward (Figure 5.26). The user simply clicks on the radio button above the language that he/she wishes to change. In designing this screen, flag icons were used to make it easier for a user selecting a language.

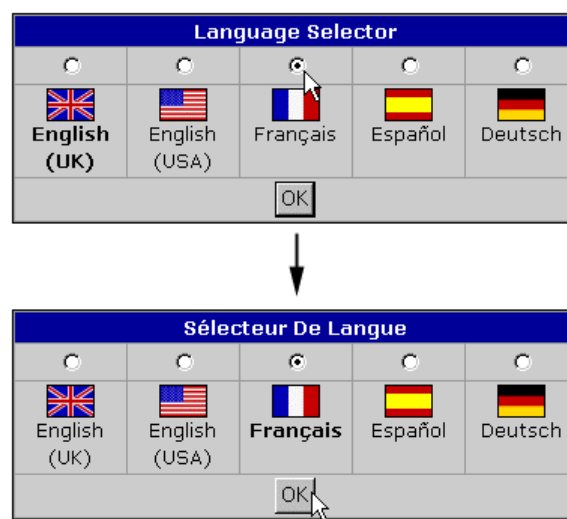


Figure 5.26

5.2.20 - Help

As this is a web system, users will expect this system to be easy to use and they will not wish to have to look up help documentation. Therefore it has been decided that help in the system will be provided as a Quick Help box at the bottom of each page in the system. Extra help documentation will be made available but the quick help box should provide enough information to help with most queries or problems. The Quick Help box itself is very simple (Figure 5.27). The title uses white text on a grey background. The main area uses small blue on a light yellow background.

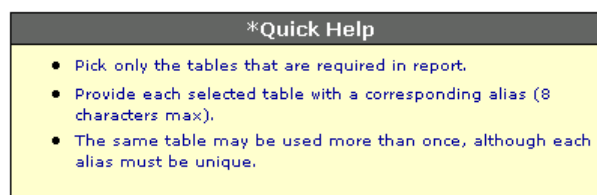


Figure 5.27

5.2.21 - Overall Interface Design

All of the individual screens of the system have now been designed. The next step is to design the entire system interface. The basic interface layout is fairly straightforward and is of a sequential nature. Like the majority of web systems, the first element is the title image of the system “Generic Online Reporting”. The menu that provides structure and links through the system follows this. The next section is the main section of the screen and is the main area of functionality. This area is where all the individual screens designed previously are. The bottom element of each screen will be the Quick Help section. Figure 5.28 shows the full interface design for the Report -Edit - Tables section.

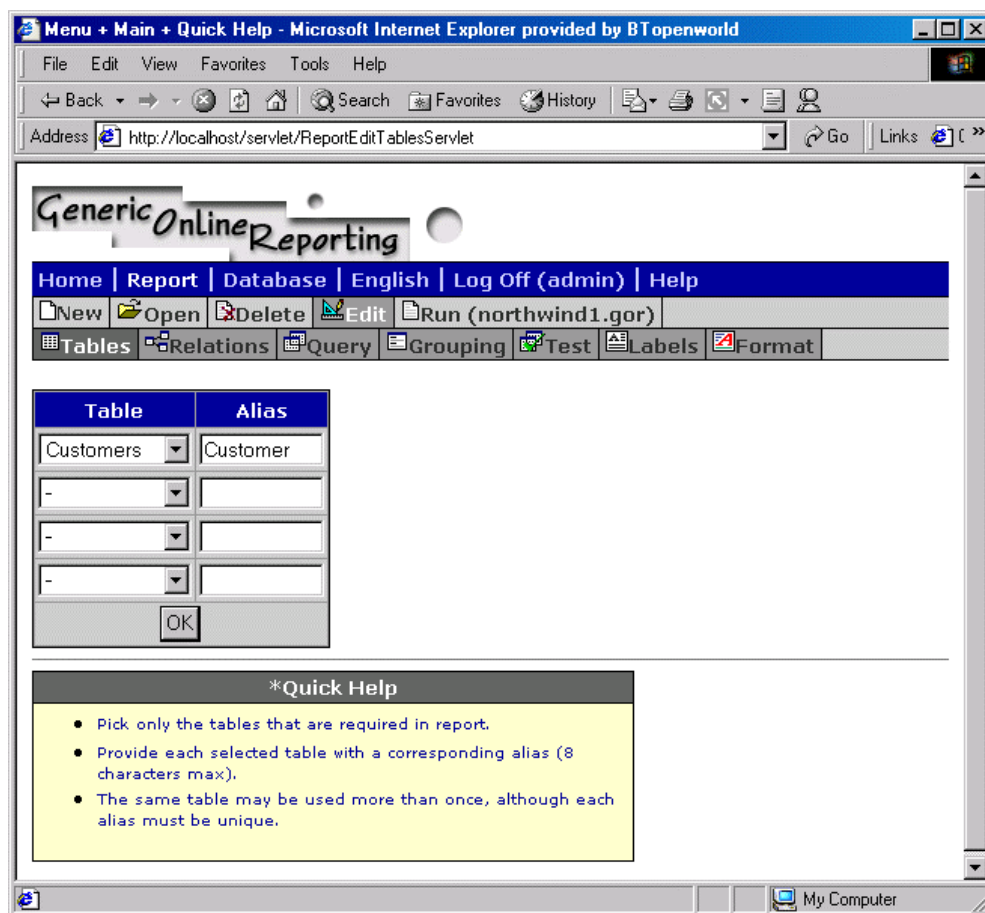


Figure 5.28

5.3 - Data Structure Design

When designing the interface to the system, the data files which these screens created/changed were kept in mind. In the brief design of the system, it was recognised that there would have to be two files necessary to create/run a report. These are the database connection file, which the database interface of the system creates, and the report – edit section creates.

5.3.1 - Database Connection File (*.dbd)

The first file is the database connection file. This file must hold enough information necessary to create a connection to the database. This sort of information includes the database type (e.g. Access), database name (e.g. Northwind) and connection information. This includes a driver (e.g. sun.jdbc.odbc.JdbcOdbcDriver which connects a ODBC connection to a java program) and a connection URL which specifies the location of the database (e.g. jdbc:odbc:northwind).

Other information that will be required will be tables in the database (e.g. EMPLOYEE) and their columns names and types (e.g. EMP_ID - Integer, EMP_NAME – varchar etc). All this information will be stored in a text file with extension *.dbd (made up extension). The following is the exact syntax of the file with example data:

```
[DATABASE_INFO]
DATABASE=Access                // Name of database
DRIVER=sun.jdbc.odbc.JdbcOdbcDriver // Name of driver
CONN_URL=jdbc:odbc:            // Connection URL.
CONN_NAME= northwind           // Connection name. In ODBC this is the system DNS
USERNAME=admin                 // optional username require
PASSWORD=admin                 // password

[TABLES]
Employees                      // Names of all tables in database
...

[COLUMN_NAMES_AND_TYPES]
EmployeeID,4,LastName,12,FirstName,... // List of Column names and types
...
```

As mentioned, the [DATABASE_INFO] is what the user specifies. This is needed to create a database connection and retrieve the database metadata (table and column names/types). The system will write then this file to the web server so the report section of the system can use it.

5.3.2 - Report File (*.gor)

This file will contain all the information necessary to run one report. This information includes:

1. General information - name of user who created the report, name of database connection, date etc.
2. Tables and Aliases – names of all the tables and aliases used in a report.
3. Relationships – list of 2 tables and 2 columns where a join is necessary. This is only needed if there are more than 1 set of tables and aliases.
4. Query – This is a list of columns in a query, with option information such as conditions and sorts.
5. Groups - There is going to be maximum of 5 groups in a report.
6. Labels - List of labels in a report.
7. Format - information about the formatting of a report.

The following is the exact syntax of a report file with example data:

```
[GENERAL_INFO]                // -- GENERAL INFORMATION --
DATABASE=northwind.dbd        // database connection filename
CREATED_BY=admin              // user who created file (from logon)
CREATED_DATE=2002.04.24       // data file was created
EDITED_BY=robert              // user who is currently editing file

[TABLES_ALIAS]                // -- LIST OF TABLES AND ALIASES --
1,Employees                   // table number from .dbd file (0 to number of
5,Orders                       // tables - 1), name of alias.

[RELATIONSHIPS]               // -- LIST OF RELATIONSHIPS --
0, 5, 1, 2                    // alias number (0 to number of TABLES_ALIAS - 1),
                               // column number from .dbd for that particular
                               // table. 3rd and 4th number are the same (two
                               // pairs needed for join).

[QUERY]                       // -- LIST OF COLUMNS AND CONDITIONS --
1,0,3,0,0,                    // show (1=yes, 0=no), alias number, column number,
0,0,4,0,1,34                  // sort (0=none, 1=ASC, 2=DESC), condition type
                               // (0=none, 1=equal to, 2=not equal to, etc),
                               // condition value (if blank then none exists)

[GROUPS]                      // -- LIST OF GROUPS --
0,3                            // alias number, column number

[LABELS]                      // -- LIST OF LABELS --
DEFAULT_LOCALE=0              // default locale
```

```

[LABELS_0]                                // First set (there can be any number of sets)
LOCALE=0                                  // locale number (from list of locales)
TITLE=Northwind                          // Title
SUBTITLE=Employee details.                // Subtitle (usually longer)
FOOTER=www.northwind.com                  // Footer at bottom of page.
0,3,Title                                // Data labels: alias number, column number, Text
0,0,ID

[FORMAT]                                  // -- FORMAT INFORMATION --
PAGE_ORIENTATION=0                        // Page width: 0=Portrait, 1=Landscape, 2=100%
5,5,0,0,0,1,2,0,11,8,3,0,5,2,0,1,0,1,0,2,4,8,0,4,5,2,0,0,0,0,0,0,1,0,0,0,5,2,1,0,0,0,9,8
,8,0,0,0,5,2,0,0,0,0,0,0,0,1,0,0,0,5,2,1,0,0,0,0,2,1,0,0,1,5,2,0,0,0,0,0,0,1,0,0,0,5,2,1,0
,0,0,0,2,1,0,0,1,5,2,0,0,0,0,0,0,1,0,0,0,5,2,1,0,0,0,0,2,1,0,0,1,5,2,0,0,0,0,0,0,1,0,0,0
,5,2,1,0,0,0,0,2,1,0,0,1,5,2,1,0,0,0,0,0,0,8,1,1,5,2,0,0,0,0,12,1,1,0,13,1,5,2,0,0,0,0,8
,0,1,0,0,0,1,0,0,0,0,                                // List of numbers which can format entire report

[END]                                      // -- END OF REPORT FILE --

```

There are other smaller files in the system which must carry out essential tasks. They are as follows:

5.3.3 - label_locales.dat

This file will hold information on all the locales that a user can specify the labels of a report in.

```

English (UK),en,GB                        // Locale - Language (Country), language code,
English (USA),en,US                       // country code.
Albanian (Albania),sq,AL
Arabic (United Arab Emirates),ar,AE
.. etc

```

5.3.4 - interface_locales.dat

This file will hold information on all the locales that a user can specify the interface of the system in.

```

NUM_OF_INTERFACE_LOCALES=5                // Number of languages system provides
English<br>(UK),en,gb,flag_britain.gif     // Locale Text, language code, country
English<br>(USA),en,US,flag_usa.gif        // code, name of image (flag icon).
Français<br>,fr,FR,flag_france.gif
Español<br>,es,ES,flag_spain.gif
Deutsch<br>,de,DE,flag_germany.gif

```

It has been decided that about 5 different locales will be used (although only 4 languages as English is used for both the UK and USA).

5.3.5 - users.dat

Each line of this file will hold a username and password to the system in encrypted form.

```
gxZoQDLZhHiWlo
IGA*gK0m
Uxg_WpK0mPo
geggQD66LXDxV5kZW
```

When a user completes the Logon form (section 5.2.2) the username and password that is entered will be encrypted. The system will then inspect each line of the user.dat file for a match. If a match is found then the user is granted access to the system (figure 5.29).

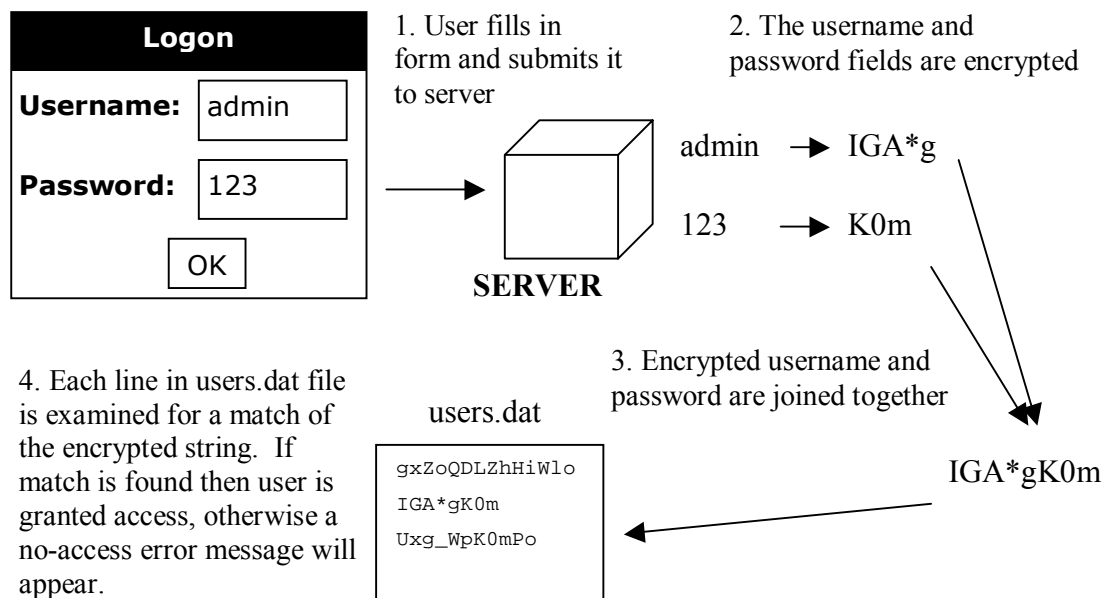


Figure 5.29

Any encryption on the site is will be done using the following simple algorithm:

1. Take two strings containing all the letters in the alphabet (lower and uppercase) plus numbers 0 to 9 and any extra characters you wish to use e.g. _*. Keep one in order and scramble the other.

Position: 0123456789 etc

Normal: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890_*

Scrambled: I18ZGePoHu6*WUAxLg1h_QaiDspfXRc29TzCMqw7JjdN3SrBvnO4Kb0Vm5EkytFY

2. To encrypt a string, find the position of each of its characters in the normal string and create a new encrypted string with the characters at that position in the scrambled string. E.g. `ace` = `I8G`.

3. To further encrypt the string shift its characters to the right depending on its position in the string. E.g. `ace` = `lzp` (positions of `ace` in normal String are 0,2,4. After shifting right depending on position they become: $0+0, 2+1, 4+2 = 0,3,6$).
This ensures that a string such as `aaaaa` doesn't become `lllll` but rather becomes `l18zg`. This should make it harder for a hacker to crack the encryption.
4. To decrypt a string simply do the reverse i.e. find its position in the encrypted string and shift characters to the left depending on its position.

Encryption could be done using much more advanced techniques like RSA encryption with uses public and private keys etc, but this beyond the scope of what is already a large project.

5.4 - Component Design

Now that all the interface screens have been created the next step is to design the components of the system and interactions with each other. In the design of any system it is good practice to keep the component, data and presentation (interface) layers separate from each other. This increases the maintainability of a system. For example, if a developer would like to create an offline version of the system (can be run from the client program), then the developer only needs to create the classes for the presentation layer. This saves a considerable amount of work.

It has been decided that the system is going to be programmed in Java for various reasons (see chapter 6 - Implementation). One of the main advantages of using Java as the programming language is that it is fully "object orientated". To help represent the system, UML diagrams will be used as they used for creating object orientated system designs.

5.4.1 - Data Classes

Most of the data files described in the previous section (see section 5.3), will have dedicated classes, which will "interface" with these files. Out of these files, the ReportFile class will be the most complex of these, as it will be the only one to provide methods to update a file.

- **DatabaseConnectionFile** - This class will have methods to create a new file, open an existing file and methods for retrieving table names and column names/types. See section 5.3.1.
- **ReportFile** - This class will have methods to create a new blank report, open and read all the sections that make up a report, update individual sections of a report file (e.g.

Query) or to retrieve sections of a report file (e.g. TablesAndAliases). See section 5.3.2.

- **TablesAndAliases** - this class will represent the [TABLES_ALIASES] section in a report file. It will provide methods for adding a new set of tables and aliases, accessing existing tables and aliases and displaying the current number of sets of tables and aliases it currently holds.
- **Relationships** - this class will represent [RELATIONSHIPS] section in a report file. It will provide methods for adding a new pair of aliases and columns that create a join, accessing existing relationships and displaying the current number of sets of relationships it currently holds.
- **Query** - this class will represent the [QUERY] section in a report file. It will provide methods for adding a new aliases and columns with optional conditions or sorts. It will have methods for accessing any of these and displaying the current number of columns the query currently holds.
- **Groups** - this class will represent the [GROUP] section in a report file. It will provide methods for adding a group, accessing existing grouped aliases and columns and displaying the current number of groups.
- **Labels** - this class will represent the [LABELS] section in a report file. It will provide methods for adding new labels, accessing existing labels and displaying the current number of labels.
- **Format** - this class will represent the [FORMAT] section in a report file. It will provide methods for accessing the many different parts of format such as font, font size etc.
- **InterfaceLabels** - class will load all the interface labels in the interface_locales.dat file (see section 5.3.4) and provide methods for accessing each its elements such as the label, language and country.
- **LabelLocales** - class will load all the interface labels in the label_locales.dat (see section 5.3.3) file and provide methods for accessing each its elements such as the label, language and country and image (name of the flag in the language selector interface - see section 5.2.19).

Note: the users.dat file will not have class associated with it because the only time it is accessed is when a user logs on and a scan is made through each line of the file.

5.4.2 - Functional Classes

Functional classes are classes that will provide functionality to the system. These are generally specific tasks such as algorithms.

- **HtmlMenu** - this class will create the menu at the top of the screen. It will do this by reading the menu_ *language_country*.properties resource bundle and matching up the codes with text and links to other sections of the system.
- **HtmlOutput** - this class will be responsible for creating error messages and the Quick Help section at the bottom of the screen.
- **QueryGenerator** - this class is responsible for taking data from the first 4 sections of the report file ([TABLES_ALIASES], [RELATIONSHIPS], [QUERY] & [GROUPS]) and creating an SQL string. The algorithm that does this will be explained in more detail in the algorithm section 5.5.1.
- **ReportGenerator** - This algorithm will generate a report based data created from the QueryGenerator algorithm, current set of Groups, Labels and format information. The algorithm that does this will be explained in more detail in the algorithm section 5.5.2.
- **SimpleENC** - this class will do any encryption needed in the system. This class provides methods for encrypting a String and also decrypting a String back again. Its main use will be the users.dat file (see section 5.3.5).

5.4.3 - Interfaces Classes

These classes are responsible for providing an interface to the user. All the screens, which were designed in the Interface section of this chapter, will be inserted into these classes. Each of these interface classes will provide a specific task and match up with the interface section of this chapter (see section 5.2).

- **DatabaseAddServlet** - This class is responsible for providing an interface for a user adding a database connection to the system (creates a new dbd file). It will use the DatabaseConnectionFile to actually create the file.
- **DatabaseDeleteServlet** - This class provides an interface for deleting a database connection file from the system.
- **DatabaseOpenServlet** - This class provides an interface for opening an existing database connection file.
- **DatabaseServlet** - This class shows what is available in the Database section of the system.

- **HomeServlet** - This class will display the welcome screen that greets a user to the system.
- **LanguageServlet** - This class will provide an interface for a user to change his desired language for the system. This class will use the InterfaceLabels class to display what languages are available. This ensures that a recompile of the system isn't necessary to add a new set of interface labels.
- **LogoutServlet** - This class will log a user off the system.
- **LogonServlet** - This class will display a username and password box for a user to log onto the system. It will also check the users.dat (section 5.3.5) file to see if that user name and password exist. If it does, it will allow the user access, otherwise it will bring up an error message.
- **ReportDeleteServlet** - This class provides an interface for deleting a report file.
- **ReportEditFormatServlet** - This class will retrieve the current [FORMAT] data from a report file and display this in a visual manner. A user can then change this and update the [FORMAT] section of the report file. This class will use the ReportFile and Format classes for retrieving and updating.
- **ReportEditGroupingServlet** - This class will retrieve the current [GROUP] data from a report file and display this in a visual manner. A user can then change this and update the [GROUP] section of the report file. This class will use the ReportFile and Group classes for retrieving and updating.
- **ReportEditLabelsServlet** - This class will retrieve the current [LABELS] data from a report file and display which labels have been used. A user can either add a new set of labels update existing labels. The [LABELS] section of the report file will then be updated. This class will use the ReportFile and Labels classes for retrieving and updating. It will also use the LabelLocales class to show the available locales the report labels can be created in.
- **ReportEditQueryServlet** - This class will retrieve the current [FORMAT] data from a report file and display this in a visual manner. A user can then change this and update the [FORMAT] section of the report file. This class will use the ReportFile and Format classes for retrieving and updating.
- **ReportEditRelationsServlet** - This class will retrieve the current [RELATIONSHIPS] data from a report file and display this in a visual manner. A user can then change this and update the [RELATIONSHIPS] section of the report file. This class will use the ReportFile and Relationships classes for retrieving and updating.

- **ReportEditServlet** - This class will show what is available in the Report - Edit section of the system.
- **ReportEditTablesServlet** - This class will retrieve the current [TABLES_ALIAS] data from a report file and display this in a visual manner. A user can then change this and update the [TABLES_ALIAS] section of the report file. This class will use the ReportFile and TablesAndAlias classes for retrieving and updating.
- **ReportEditTestServlet** - This class will use the QueryGenerator class to create and run an SQL statement and show the query as a table of data. This allows the user to know that their Query has been successfully constructed.
- **ReportNewServlet** - This class provides an interface for a user to create a new report file.
- **ReportOpenServlet** - This class provides an interface for a user to open an existing report file. A user will be given the open to open a file for editing or simply for viewing.
- **ReportRunServlet** - This class will provide an interface for running a report. It will use the ReportGenerator class to create the report using all the information in a report file.
- **ReportServlet** - This class will show what is available in the Report section of the system.

5.4.3.1 – Class diagram of Entire System

Figure 5.30 shows a class diagram of the entire system. Although it looks like a bunch of lines it conveys several important pieces of information:

1. The diagram shows how the Servlet classes map the overall structure of the interface.
2. The diagram displays how certain Servlet classes use a dedicated data class. E.g. ReportEditQueryServlets uses the Query class.
3. The diagram shows how the TablesAndAlias, Relationships, Query, Groups, Labels and Format class are all dependant on the ReportFile class. The ReportFile class is also dependent on a DatabaseConnectionFile class. NOTE: The ReportFile class is being more by other classes than any other class in the system.
4. The diagram shows what classes are required by the QueryGenerator and ReportGenerator classes. Note that the ReportGenerator requires the QueryGenerator class.

5.5 - Algorithm Design

The final step in the detail design of the system is the algorithm design. Algorithms in any system carry out a specific task or function. In this system two major tasks have been identified. The first is creating an SQL statement from Tables, Relations, and Query sections of the system. The second is creating a report using this SQL statement, labels and format information.

5.5.1 - Query Generator Algorithm

One of the most complex algorithms in the system will be the “query generator”. It will take data from the first 4 sections of a report (1. Tables & Aliases, 2. Relationships, 3. Query and 4. Groups) and construct a single SQL string. This string will be run against the database in question, which will then return a result-set (table of data which the report will consist of). Diagram 5.31 shows a scenario using sample data from the Northwind database, followed by the SQL string that this scenario should create.

1. Tables And Aliases

Table	Alias
Products	Products
Suppliers	Supplier
Categories	Categors

4. Groups

Priority	Visible Columns
1	Categors.CategoryName

2. Relationships

Parent Aliases / Columns		Child Aliases / Columns	
Products	SupplierID	Supplier	SupplierID
Products	CategoryID	Categors	CategoryID

3. Query (Columns And Conditions)

S	Aliases And Columns	Sort	Con	Value
<input checked="" type="checkbox"/>	Products ProductID	ASC	-	
<input checked="" type="checkbox"/>	Products ProductName	-	-	
<input checked="" type="checkbox"/>	Supplier CompanyName	-	-	
<input checked="" type="checkbox"/>	Categors CategoryName	-	-	
<input type="checkbox"/>	Products UnitPrice	-	<	1000
<input type="checkbox"/>	-	-	-	

Figure 5.31

SQL Statement: (coloured text comes from Figure 5.30, black text is additional).

```
SELECT
Categories.CategoryName, Products.ProductID, Products.ProductName,
Supplier.CompanyName
FROM
Products AS Products, Suppliers AS Supplier, Categories AS Categors
WHERE
Products.SupplierID=Supplier.SupplierID AND
Products.CategoryID=Categors.CategoryID AND Products.UnitPrice<1000
ORDER BY
Categories.CategoryName ASC, Products.ProductID ASC
```

The Tables And Aliases section of the system declared 3 tables: Products, Suppliers and Categories and gives them 3 aliases: Products, Suppliers and Categors respectively. These are put in the FROM section of the SQL and are separated by commas.

```
FROM Products AS Products, Suppliers AS Supplier, Categories AS
Categors
```

These 3 tables must be joined together. This is done in the Relations section of the system. The SupplierID from the Products table (foreign key) is joined with the SupplierID from the Supplier Table (primary key). The CategoryID from the Products table (foreign key) to the CategoryID from the Category Table (primary key). These are declared in the WHERE section of the SQL (condition section). SQL Joins can be carried out using a variety of methods, such as INNER JOINS, nested SELECT statements, but this type of join was the easiest and did what was needed.

```
WHERE Products.SupplierID=Supplier.SupplierID AND
Products.CategoryID=Categors.CategoryID AND
```

The Query section is next, and as the diagram shows it is the most complex with 3 different boxes. They are as follows:

1. **Visible columns** – columns where the “S” checkbox is ticked (orange box).
2. **Condition columns** – columns where the “Con” pull-down has been selected and a value specified in the “Value” textbox (light blue box). It doesn’t matter if the “S” box is checked or not for this section.
3. **Ordered columns** – columns that are ordered (purple).

The following columns Products.ProductID, Products.ProductName, Supplier.CompanyName, Categories.CategoryName are visible columns in the query (S checkbox is checked). The Categories.CategoryName is moved to the first in the SELECT section, as it is a group. If a column in a query is grouped then the column is shifted to the front of the SELECT part of an SQL query depending in its priority (first priority group columns go first, second go second etc).

This in turn makes simpler the process of converting a query with groups, into a report. The reason for this will be explained in more detail in the “Report Generator” algorithm (5.5.2).

```
SELECT Categori.CategoryName, Product.ProductID,
Product.ProductName, Supplier.CompanyName
```

The condition SQL is put into the WHERE section of the SQL (after the relations section):

```
AND Product.UnitPrice<1000
```

The last section SQL is the ORDER BY section. In the query, the Product.ProductID was the only column which was sorted (SORT) pulldown, but as you can see the Categori.CategoryName (Column which was grouped) is included at the start of the ORDER BY section. Groups are sorted to keep their rows together. They are sorted in ascending order automatically (ASC) but this can be changed in the Query section (because only visible columns can be grouped).

```
ORDER BY Categori.CategoryName ASC, Product.ProductID ASC
```

When creating a query the minimum you can have is 1 table/alias and 1 visible column. This would translate to the following:

```
SELECT Alias.A_Column FROM Table as Alias
```

As you can see, the WHERE and ORDER BY parts of the SQL statement aren’t used here at all. To create the entire SQL string the following syntax will be used:

```
// Full SQL String is created from 4 small SQL strings.
String fullSQLString, selectSQL, fromSQL, whereSQL, orderBySQL;

// the selectSQL and fromSQL MUST be created but the whereSQL and orderBySQL
// are optional. Presume initially they won't be used.
boolean useWhereSQL = false;
boolean orderBySQL = false;
...
// CREATION OF selectSQL, fromSQL, whereSQL and orderBySQL
...
String fullSQL = "SELECT " + selectSQL + " FROM " + fromSQL;
if (useWhereSQL) fullSQL = fullSQL + " WHERE " + whereSQL;
if (orderBySQL) fullSQL = fullSQL + " ORDER BY " + orderBySQL;
```

An interesting point is that much of the report information is stored as numbers (this greatly simplifies implementing the interfaces, e.g. a number can be easily translated to a position in a pull down).

The following is what the report and database file would look like for the previous query.

- Report File -	- Northwind Database Connection File (bold denotes table or column used in query) -
[TABLES_ALIAS]	[TABLES]
5,Products	Categories
7,Supplier	Customers
0,Categors	Employees
	OrderDetails
[RELATIONSHIPS]	Orders
0,2,1,0	Products
0,3,2,0	Shippers
	Suppliers
[QUERY]	
1,0,0,1,0,	[COLUMN_NAMES_AND_TYPES]
1,0,1,0,0,	CategoryID ,4, CategoryName ,12,Description,-1,Picture,-4,
1,1,1,0,0,	CustomerID,12,CompanyName,12, , ... etc
1,2,1,0,0,	EmployeeID,4,LastName,12,FirstName, ... etc
0,0,5,0,3,1000	OrderID,4,ProductID,4,UnitPrice,2, ... etc
	OrderID,4,CustomerID,12,EmployeeID,4,OrderDate,93, ... etc
[GROUPS]	ProductID ,4, ProductName ,12, SupplierID ,4, CategoryID ,4,QuantityPerUnit,
2,1	12, UnitPrice ,12, ... etc
	ShipperID,4,CompanyName,12,Phone,12, ... etc
	SupplierID ,4, CompanyName ,12,ContactName,12,ContactTitle,12, ... etc

For example the first line in the [TABLES_ALIAS] section is 5,Products (table number, alias String). To get the text of a table (5) the DatabaseConnectionFile class (e.g. instance dcf) would be needed to retrieve the name from it i.e. dcf.getTable(5) = Products. This applies to all the other sections in the report file. See section 5.3.2 for a detailed description of the report file.

The algorithm shows how to populate the four different Strings selectSQL, fromSQL, whereSQL, orderBySQL is as follows (pseudo-code form):

```
// ===== selectSQL =====
create 2 arrays to hold visible aliases and columns (Size is number of visible query
columns in query)
String [] visibleAlias, [] visibleColumns;
if any visible alias and column is a groups, put these at the start of array
populate with rest of visible aliases columns
for (i = 0; i < size of visibleAlias; i++) {
    selectSQL = selectSQL + visibleAlias[i] + "." visibleColumn;    // alias.column
    if (i < size of visibleAlias array - 1)    // add a comma except for the last
        selectSQL = selectSQL + ", ";
}
// ===== fromSQL =====
for (int i = 0; i < number Of aliases; i++) {
    fromSQL = fromSQL + getTable(i) + " AS " + getAlias(i);
```

```

        if (i < number of aliases - 1)           // add a comma except for the last.
            selectSQL = selectSQL + ", ";
    }
    // ===== whereSQL =====
    if (numOfConditions != 0 || numOfRelationships != 0)
        useWhereSQL = true;

    // do relations first
    for (int i = 0; i < number of relations; i++) {
        whereSQL = whereSQL +
            getPrimaryRelationAlias(i) + "." + getPrimaryRelationColumn(i) + "="
            getSecondaryRelationAlias(i) + "." + getSecondaryRelationColumn(i);
        if (i < number of relations - 1)           // add a " AND " except for the last.
            whereSQL = whereSQL + " AND ";
    }

    // do conditions next
    declare String array conTypeList [] = {"-", "=", "<>", "<", ">", "<=", ">="};
    String conditionValue;
    for (int i = 0; i < number of conditions; i++) {
        // check database meta data. If current column is text based then wrap value from
        // query text box with single quotes.
        if (column is text based)
            conditionValue = "'" + getConditionValue(i) + "'";
        else
            conditionValue = getConditionValue(i);

        whereSQL = whereSQL +
            getConditionAlias(i) + "." + getConditonColumn(i) + // alias.column
            conTypeList[getConditionType(i)] +                  // con type e.g. =, >, <, etc
            conditionValue;
    }

    // ===== orderBySQL =====
    create 3 arrays to hold ordered aliases, columns and sort num (0 = -, 1 = ASC, 2 = DESC)
    String orderdAlias [], orderedColumn [], orderBySortNum []
    // Size is number of ordered query columns in query)
    declare String array orderList [] = {"", "ASC", "DESC"};
    if any are groups, put these at the start of array
    populate with rest of ordered aliases columns
    for (i = 0; i < size of orderdAlias; i++) {
        orderBySQL = orderBySQL +
            orderdAlias [i] + "." + orderedColumn [i] + " "
            orderList [orderBySortNum[i]];
        if (i < size of orderdAlias array - 1)
            orderdAlias = orderdAlias + ", ";
    }
}

```

5.5.2 - ReportGenerator Algorithm

The QueryGenerator algorithm creates an SQL from the first four sections of the Report – Edit section. However, the aim of this system is to create a report. The system uses the SQL String created from the QueryGenerator to create a result set (table) of data from a database. This table of data, along with labels, groups (optional) and formatting information merge together to create the final report (see Figure 5.32). As mentioned in the QueryGenerator section, the Group section's purpose is simply repositioning columns in the SQL to the start. This organises the data and makes it easier when it comes to creating the report HTML.

The algorithm in English terms for creating an HTML report is as follows:

1. Run the QueryGenerator and run the generated SQL against the database. This will retrieve a result set from the SQL.
2. Retrieve group data from the report file. The class Groups will provide an interface in which to access groups. E.g. groups.getGroup(0).
3. Retrieve labels data from the report file. The class Labels will provide an interface to accessing the various labels such as title, sub title. E.g. labels.getTitle().
4. Retrieve formatting data from the report file. The class Format will provide an interface for accessing the various formatting elements e.g. format.getColourBackground.
5. Declare a String called reportHTML. This will hold the entire report. The report will be in the form of a large one-column table with many rows. Each section in the report is a table row (or a HTML <tr>). Each optional line above or below each section is also a row also.
6. Start constructing the reportHTML String. The first HTML tag to create is the <table> tag. The table will have no border, no cell padding (spacing inside each <td>) and no cell spacing (spacing between each <td>). This basically creates an “invisible” table and is used frequently in web systems for element alignment. The orientation of the page will affect its width (e.g. Landscape = 635 pixels).

```
<table width='635' border='0' cellpadding='0' cellspacing='0'>
```

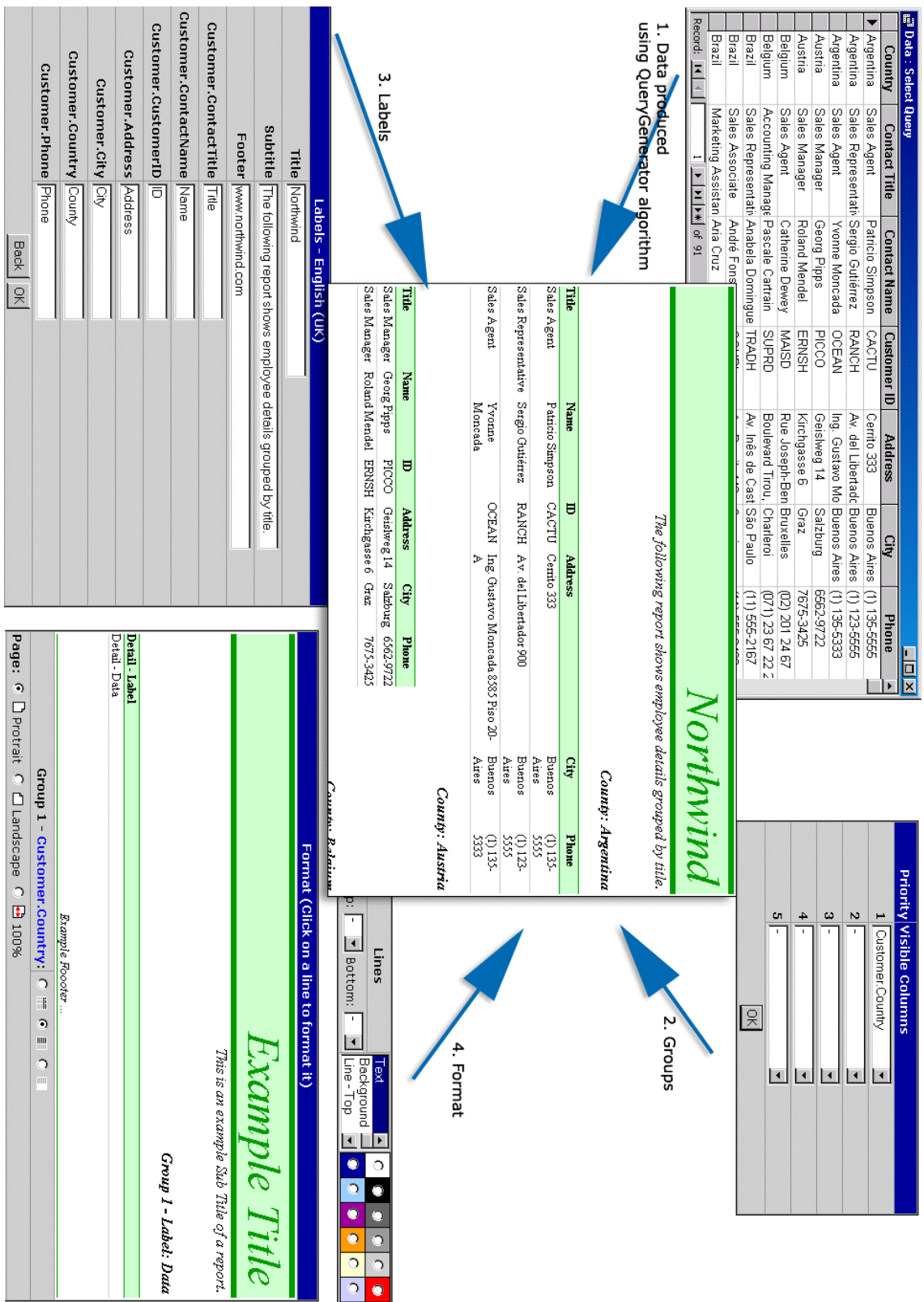


Figure 5.32

7. Add the title and sub title to the HTML string. The text for the titles will come from the Labels class. The formatting will come from the format class. The following box shows the HTML for the title of the report from Figure 5.31. The first <tr> displays the line at the top of the screen which is basically an empty table row with a particular background colour (note that “1p.gif” is a very useful transparent image of 1x1 pixel size. If it wasn’t used the <tr> would be the height of standard text size). The text in bold denote colours, line heights, fonts, font sizes, italics.

```
<tr bgColor='009900' height='6'>
  <td><img src=/gor/images/1p.gif width='1' height='1'><td>
</tr>
<tr>
  <td align='right' bgcolor='CCFFCC'>
    <font face='Times New Roman' size='7' color='009900'>
      <i>Northwind</i>
    </font>
  </td>
</tr>
// Bottom line is same as top (first tr)
```

8. The next section to be added to the reportHTML string is main section of the report. This includes all the labels and data. If no groups have been declared then it is quite straightforward. The next row to be added is the labels at the top of detail section. The detail section has generally more than one column so another table must be generated. This table will be *nested* inside the next row following the sub title section. This table is similar to the main table as it has no cell padding or no cell spacing. One difference is that it doesn’t have any width. The width of the table will be determined by the length of the text inside each of its cells. The table will not become wider than the main table but may stretch to the full width of the main table.

```
<tr>
  <td>
    <table border='0' cellspacing='0' cellpadding='0'>
```

The labels for the detail section will use the formatting techniques used as the title sections but will have one or more <td> for each of its columns. Another point to note is each column (<td>) excluding the last column will have another column with a width of 10 and displays a single blank space (HTML character). This column creates a small column between every column to space them out.

```
<td width='10'>&nbsp;</td>
```

9. Once the labels have been appended to the reportHTML String, all the data labels must then be added. To achieve this each row of the query result-set will create one HTML

- row. Again the HTML is very similar to the creating labels except data from the result set is used. The detail section is ended when there is no more data in the result set.
10. If a report has groups, then the column label and data for that group is attached to the reportHTML String before the detail section is created. The detail labels are then created and each row in the result-set creates a row in the HTML until the current group columns changes in the result-set from the previous column. E.g. Figure 5.32 shows the group Customer.Country changing from “Argentina” to “Austria”. Once this happens detail table is terminated (`</table>`) and this process repeats until the end of the result-set.
 11. The final step is to create the footer of the report (not shown in Figure 5.31). This is practically identical to creating a title.

5.6 - Conclusion

A detailed and comprehensive design has now been completed. The next stage of the system is Implementation. This will involve using this detailed design to create the system. At this stage the majority of the interface screens, data files, components and major functionality (algorithms) have been designed.

Work will be required now and the next step is bringing functionality to each of these interfaces. The ultimate goal of the implementation is to create a report (ReportAlgorithm, but many intermediary steps will need to be carried out first e.g. creating database connections, creating new report files etc.

6 - IMPLEMENTATION

6.1 - Introduction

This chapter of the report will give a brief overview of the implementation stage of the project. This chapter includes a short description of Java Servlets, which provided the majority of the functionality of the system. Hardware and software used in the implementation of the project will also be discussed and the task required in turning the design on the system into a solid implementation.

6.2 - Java & Java Servlets

Java was the chosen language in which to implement this system. Some of the reasons for this are as follows:

1. Java is the only current programming language to support Internationalisation on all levels. (See section 2.3)
2. Java can be run on any system which has a program called a JRE (Java Run-time Environment). This guarantees the system could be run under Unix, Windows 98 or Solaris.
3. Java has large and very complete standard libraries. For example retrieving meta data from data was very easy using Java as it has a class called DatabaseMetaData.

Java Servlets is Java class, which reside on a server and can accept HTTP requests and responses. They are similar to CGI programs but include all the advantages of using Java. Other server side programming technologies that could have been used include ASP (Active Server Pages). ASP however has disadvantages such as lack of portability (will only run under Microsoft operating systems) and doesn't include internationalisation functionality. One advantage of ASP is that it is easier to program but this was not enough to justify using it.

6.3 - Hardware Used

Two different PCs were used in the implementation. The first PC was used in the first 2-3 months of implementation. The second PC was a laptop, which was used for the rest of the implementation.

PC 1 Specification:

- **Type** - Desktop
- **Processor Speed** - 450MHz
- **Memory** - 256MB RAM
- **Hard drive capacity** - 28GB
- **Operating System** - Windows 98.
- **Other points of interest** - DVD drive, CDRW drive, 17" Monitor, LS120 floppy drive, 100/10Base Network Card, TV Card, 56K Modem.

PC 2 Specification:

- **Type** - Laptop
- **Processor Speed** - 1GHz
- **Memory** - 128MB RAM
- **Hard drive capacity** - 20GB
- **Operating System** - Windows 2000 Professional (NT 5).
- **Other items of interest** - DVD drive, 14" Screen, 100/10Base Network Card, 56K Modem.

6.4 - Software Used

A range of different software was used in creating and running the system.

- **JDK1.3 (Sun)** - The JDK (Java Development Kit) is a collection of classes used to create RESEARCH. The JDK is a development environment for building applications, applets, and components that can be deployed on implementations of the Java 2 Platform.
- **JBuilder 3 (Borland)** - This IDE (Integrated Development Environment) is one of the best for developing Java programs. It uses various elements to make a programmers life easier: such as projects files for organisation, syntax colouring of code. All of the Java code of the project was produced using this program. One of the best things that aided coding was the ability for a class or an instance of a class to display its available methods (see Figure 6.1).

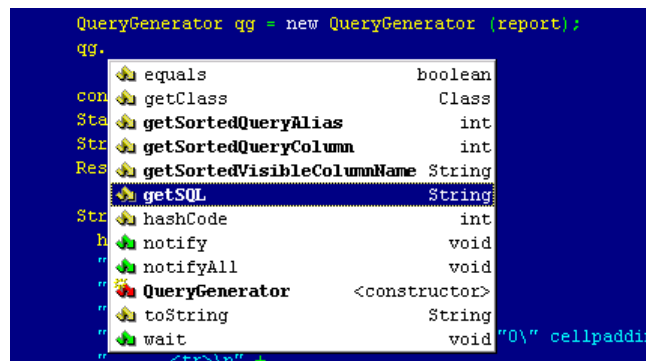


Figure 6.1

- **Photoshop (Adobe)** - This program is the market leader in creating and manipulating graphics. All the images in the system and in this report were created using this program. See Appendix C for a detailed description of all the images that were created.
- **Dreamweaver (Macromedia)** - This program is one of the best HTML web page editors available. It was used extensively for designing all the various pages in the system.
- **Java Web Server (Sun)** - This web server is created by Sun which is the same company that make Java, and its therefore is compatible with Java Servlets. This web server was used in the initial stages of development, but was abandoned as it did not support Java JDK1.3, which was needed for retrieving database meta data. Sun have now stopped making this web server and so another web server was needed.
- **Apache Web Server (Apache Group)** - This web server was used for the remainder of the project (after Java Web Server). This is the most famous and popular web server in existence, with about 70% of the Internet servers using it to host their web pages. The main reasons for this choice are the fact that it is free, fast and very reliable. However this web server doesn't support Java Servlets out-of-the-box, so an additional plug in was required.
- **Apache JServe (Apache Group)** - Various plug-ins are available for running Java Servlets under Apache Web Server. This one is made by the same company that make Apache Web Server so I decided to use this one, which in turn proved to be much faster than Java Web Server.
- **Access (Microsoft)** - This database was used in implementation of the system for the unit testing purposes. Microsoft Access is one of the most famous databases in the world and although it is not really designed for the large amounts of data, it proved to

be valuable in aiding implementation. There were initial problems with retrieving table and columns from the database using the java class DatabaseMetaData in JDK1.2 although these were fixed in version JDK1.3. This database also proved to be invaluable for inspiration in the design of the user interface of the query section.

- **MySQL (MySQL Finland AB)** – MySQL is a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. It is very popular for hosting web sites, as it is free and distributed under the GPL licence. As this project supports multiple databases MySQL was used as well as MS Access for testing purposes. It does not have a easy to use graphical user interface like Access, so all test tables, columns and data had to be generated by hand using SQL statements.

6.5 - Implementation Strategy

To fully implement this project successfully a strategy had to be created. After the detailed design of the system was completed the following steps were carried out:

1. Hardware and software were set up and installed respectively, to run the system. See the previous section for a description of the software and its required task.
2. Classes were implemented to create HTML easily. The HtmlOutput class that created the page top (title image menu) and page bottom (quick help) and also displayed HTML errors. The HtmlMenu class actually created the menu depending on its position. E.g. 2.4.3 = Report – Edit – Query.
3. All the Java Servlets (interface classes) were created in the system. There is one class for each position in the menu of the system (see Figure 6.2). The naming of each of these classes reflected their position in the menu e.g. The Servlet class for the Report – Edit – Query section was named ReportEditQueryServlet. Each of these classes displayed the menu depending on their position in the menu hierarchy of the system. This provided a skeleton of the entire system. Figure 6.2 shows what the ReportEditQueryServlet displayed. Note that the class did not provide any functionality apart from linking other servlets together (e.g. figure shows a link to the ReportEditRelationsServlet). The blue arrow depicts where the all the interface screens from the detailed design were inserted.

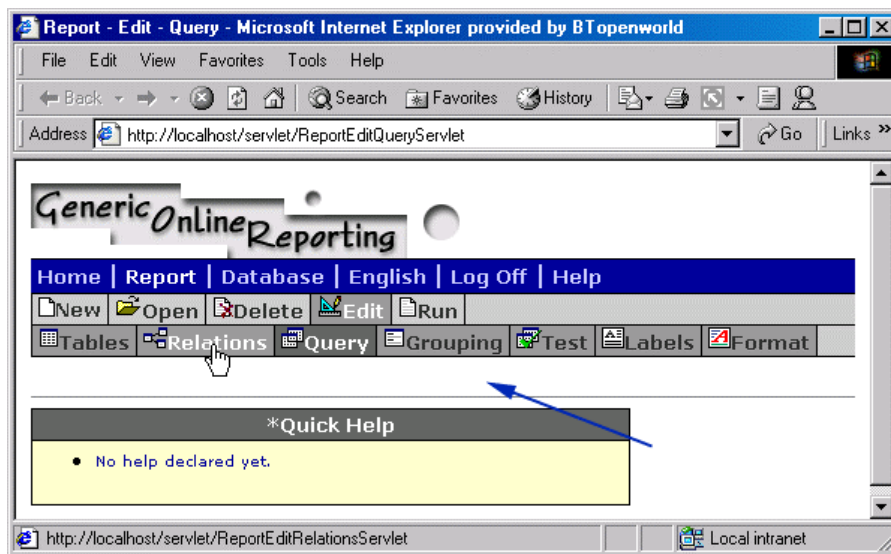


Figure 6.2

The following is the code showing the ReportEditQuery Servlet at this stage:

```
// import files
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ReportEditQueryServlet extends HttpServlet
{
    // doGet (method is called when page first loads up)
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        doPost(req, res);          // Page is passed onto doPost
    }

    // doPost
    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        // Set up session and connection
        HttpSession session = req.getSession(true);
        res.setContentType("text/html");
        ServletOutputStream out = res.getOutputStream();

        // Create instance of the HtmlOutput class.
        // Pass position of menu and session variable in a constructor
        HtmlOutput htmlOutput = new HtmlOutput("2.4.3", session);

        out.println (
            htmlOutput.pageTop() +
            // Here is where the HTML of each screen will be
            htmlOutput.pageBottom()
        );
    }
}
```

The page loaded up doGet method then loaded up the doPost method. The first line sets up the session. The session holds variables that could be accessed across other pages. Variables that became session variables included the username, current locale of the user, name of a report and if that report is being edited or not. The next line sets the

content type of the response to HTML. A `ServletOutputStream` is then declared which provided a mechanism for outputting HTML to the browser. The next line created an instance of the `HtmlOutput` class which as described earlier created HTML and includes methods such as `pageTop()`, `pageBottom()`, `error(String errorMessage)`, etc. An important point to make here is that every Servlet in the system had this code in it. The only difference was the menu position in the constructor of the `HtmlOutput` class.

4. Once all these “skeleton” classes were created it was time to “flesh” them out with the required interface that each screen is supposed to provide. The HTML interfaces, which were created in the detailed design, were then inserted into each Servlet between the `pageTop()` and the `pagebottom()` methods. The only Servlets that had HTML added to them was the `ReportServlet`, `ReportEditServlet` and the `DatabaseServlet`, as the only purpose of these servlets was to expand the menu system.
5. These servlets are still static in the sense that they are not generated depending on the fact that the user has opened a report file or just create a new file. Each Servlet was then given the necessary functionality that is required. Servlets were created in roughly the same order, as a user would use the system. This meant the `LogonServlet` was the first to be created, followed by the database servlets (e.g. `DatabaseAddServlet`), and then follow by the report servlets. The reason the database servlets were implemented before the Report servlets are because a database connection file must exist before a report file can be created. The `ReportRun Servlet`, the most important and one of the most complex servlets was created last as it tied in everything from the rest of the system. This servlet provided the interface for running a report.
6. Once every Servlet was completed and tested, the final step was to make the system multi-language. See section 2.3 for more details.

There isn't enough room in this report to explain in detail all the functionality implemented in this system. One section that will be explained is how the system was implemented in various languages. Figure 6.3 shows two screen shots of the `ReportEditTablesServlet`, one English and one in French.

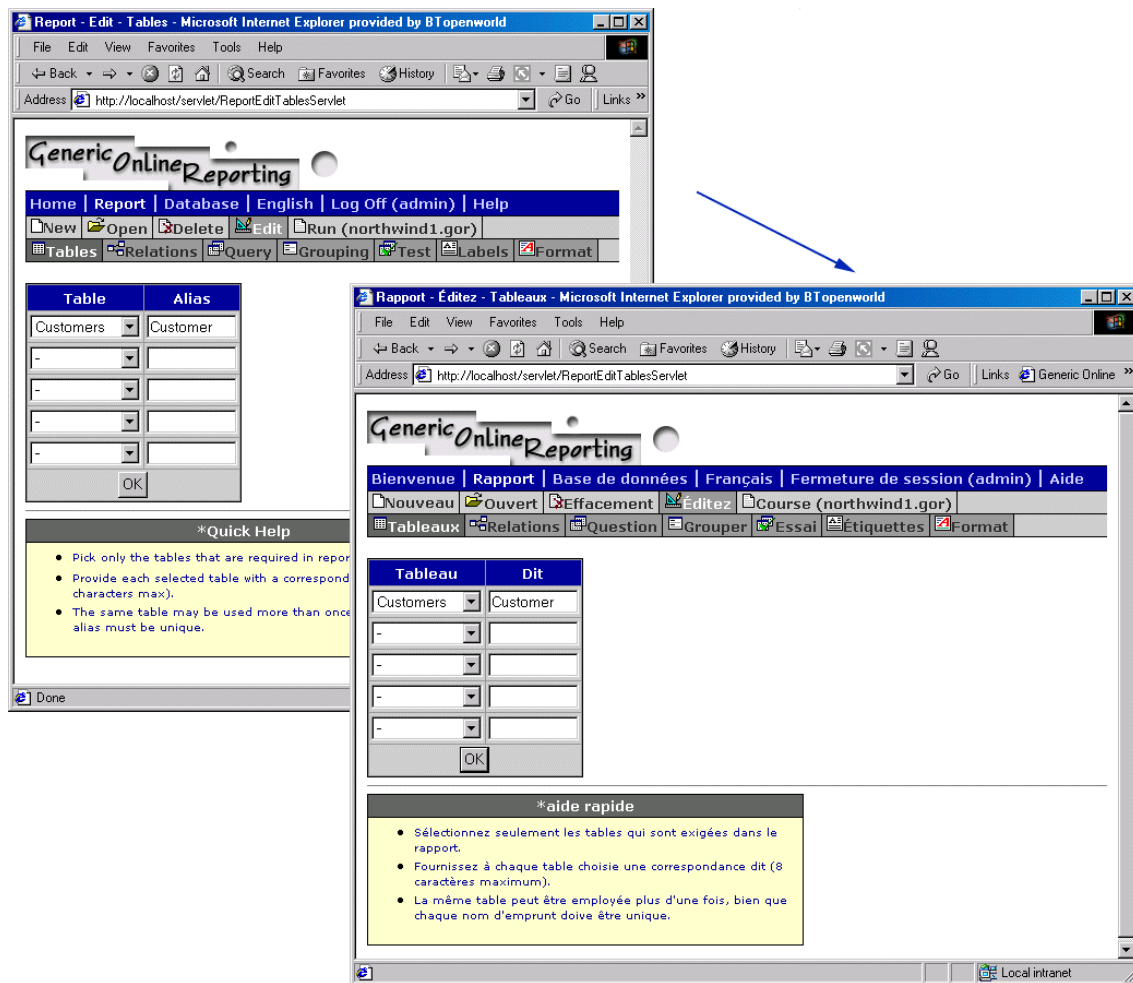


Figure 6.3

Text labels, which change on the screen, include the menu, the quick help on the bottom of the screen and the various labels on the main screen (e.g. Table, Alias and OK button). Other labels that required internationalised were any error messages that the system threw.

As explained in section 2.3 (Internationalisation), Java uses a standard class called `ResourceBundle`. This file can access properties files in which each line is saved as a `code=string`. The file name of the property file indicates the locale. e.g. `text_en_gb.properties` (English), `text_fr_fr.properties` (French), etc. The code must stay the same in each file, but the text differs (e.g. English (GB) = colour, English (US) = color).

To accommodate all these different types of labels in the system, four different files were created for each language. They are as follows:

1. `menu_language_country.properties` - holds labels for all the menu items such as text and links for each of the positions, page titles, plus the names of images.

The following are snippets from `menu_en_GB.properties` file.

```
text.2=Report
text.2.1=New
text.2.2=Open
...
title.2=Report
title.2.1=Report - New
title.2.2=Report - Open
...
link.2=/servlet/ReportServlet
link.2.1=/servlet/ReportNewServlet
link.2.2=/servlet/ReportOpenServlet
...
image.2.1=/gor/images/icon_new.gif
image.2.2=/gor/images/icon_open.gif
```

2. `quickhelp_language_country.properties` - This file holds the quick help text at the bottom of each screen. If there isn't a match then the quick help window isn't displayed at all e.g. the Home and Language selector page doesn't display any. The following are snippets from `quickhelp_en_GB.properties` file.

```
3.t1=Add - Add a connection to a database (new connection file)
3.t2=Open - View a database connection file
3.t3=Delete - Delete a database connection file

3.1.t1=To create a database connection using JDBC-ODBC ...
3.1.t2=Direct connections can be made to databases ...

3.2.t1=To view the columns of a database click on the relative table.
```

3. `labels_language_country.properties` - This file holds the words to all the labels in all individual screens in the system. This file uses the English word as its code as opposed to alphanumerical codes. The following are snippets from the `labels_en_GB.properties` and `labels_fr_FR.properties` files.

```
(labels_en_GB.properties)
AreYouSure=Are You Sure?
Back=Back
Columns=Columns
...

(labels_fr_FR.properties)
AreYouSure=Êtes-vous sûr?
Back=Dos
Columns=Colonnes
...
```


4. errors *_language_country*.properties - This last file contains error messages. Types of errors, which the system may throw, include general errors, IO file errors, SQL errors, no permission errors, etc. The following snippets of error messages are from the errors_en_GB.properties file.

```
ErrorPage=Error Page
FollowingErrors=The following error\s have occurred:
...

1=A general error has occurred!
2=User not logged onto system.
...
```

The `HtmlOutput` class uses errors and quickhelp properties files when it creates HTML error messages and the quick help. The `HtmlMenu` class uses the menu properties to create the menu. This provides a high level of transparency when these classes are used in all the individual servlets. The following snippet of code shows how easy it is write code to display the menu (`htmlOutput.pageTop()`), quick help (`htmlOutput.pageBottom()`) and error handling.

```
try {
    HtmlOutput htmlOutput = new HtmlOutput("2.4.3", session);

    out.println (
        htmlOutput.pageTop() +
        // Here is where the HTML of each screen will be
        htmlOutput.pageBottom()
    );
}
catch (IOException ioe) {      // Here is where the HTML of each screen will be
    out.println(htmlOutput.errorIO());
}
catch (Exception e) {
    out.println(htmlOutput.errorGeneral());
}
```

Error handling is quite easy to implement in Java. This is achieved using the try – catch blocks. i.e. try and run a section of code, and if this causes an error forward execution to the correct catch e.g. file error redirects to the catch (`IOException ioe`) block of code.

This covers three of the four properties files discussed previously. Each Servlet however must add the labels properties file and declared Strings at the top of each file.

For example in the `ReportEditTablesServlet` the following code is declared at the top of the class.

```
//=====
// ResourceBundle stuff
//=====

ResourceBundle rb = ResourceBundle.getBundle (
    "language.labels", Site.getLocale(session)
);

String rbTable = rb.getString ("Table");
String rbAlias = rb.getString ("Alias");
String rbOK = rb.getString ("OK");
```

The `Site.getLocale (session)` finds out the current locale the users wishes to view the system interface in and returns the correct locale. The system uses this locale and loads the correct file into the `ResourceBundle rb`. The text for the codes “Table”, “Alias” and “OK” from retrieved from the `ResourceBundle` and inserted into the Strings `rbTable`, `rbAlias` and `rbOK` respectively. The following code shows how these Strings are used to create the HTML output.

```
StringBuffer html = new StringBuffer (
    htmlOutput.pageTop() +
    " <tr><td>&nbsp;</td></tr>\n" +
    ...
    " <font color='#FFFFFF' face='Verdana' size='2'><b>" + rbTable +
    " </b></font>\n" +
    " </td>\n" +
    " <td align='center'> \n" +
    " <font color='#FFFFFF' face='Verdana' size='2'><b>" + rbAlias +
    " </b></font>\n" +
    ...
    );
```

All the other servlets in the system use this code for making the interface multi-language, although different codes are used.

Once all the servlets were using the properties files to display their text it was simply a case of translating these files. As mentioned earlier, the code stays the same but the text changes depending on the language. Five locales were created, which are as follows:

1. English (UK) – `en_GB.properties`
2. English (USA) – `en_US.properties`
3. French – `fr_FR.properties`
4. Spanish – `es_ES.properties`
5. German – `de_DE.properties`

This created a total of 20 properties files in the system. None of these files were translated professionally, as this would cost something in the region of 30 pence a word! The popular website Google was used to translate the `en_GB.properties` files into French, Spanish and German. The four English files were firstly uploaded onto a server. It was then just a matter of

supplying Google with the URL of these files and it would then translate the file in the specified language (Figure 6.4).

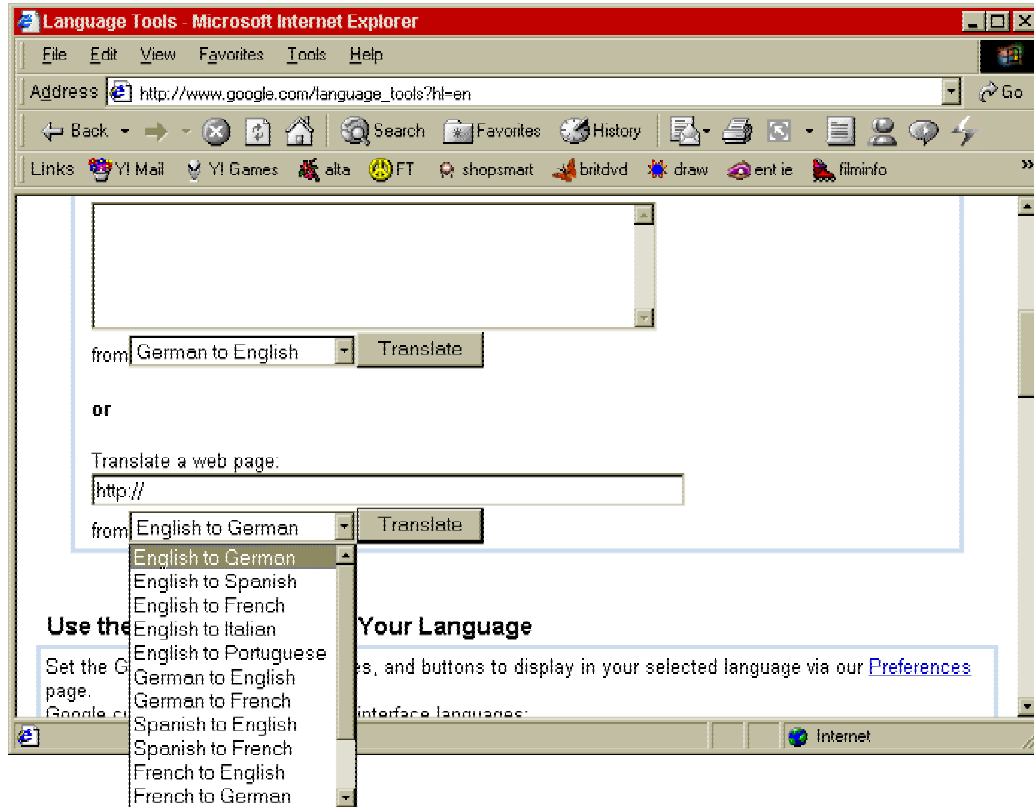


Figure 6.4

However, the translation of these files was not completely accurate. Certain words were in the wrong context (e.g. French translation of “Run” could mean “Jog”) and the grammar of sentences was incorrect (e.g. quick help). However the translation was certainly aesthetically pleasing.

(Author’s Note: I would like to thank my French friend Jean Lacroix for editing the French labels).

6.6 - Problems Encountered During Implementation

The main problem encountered in the implementation of the project was lack of time. This prevented two of the functional requirements being completed. The first uncompleted requirement was the ability of the system to view reports in formats other than HTML, e.g. PDF (portable document formats). The second was implementing run-time-parameters. See section 8.4 - Enhancements and Recommendations for further information. Having mentioned that, an

estimated 90% of the system functionality was implemented, of what is unquestionably a large and complex system.

6.7 - Conclusion

Apart from not fully completing the system, no major problems were experienced during implementation. This was down to a complete and good quality design. A strong understanding of the underlying technologies such as Java, HTML, etc has also helped considerably.

It is if felt that the source code of the system is of the highest quality. All the Java source files have been well documented using appropriate comments. The javadoc program (which comes with the Java JDK) was run to generate HTML help documents (See CD) using source code comments. The Java classes have paid attention to good object orientated principles such as encapsulation (hiding data and functionality from other classes) and polymorphism (e.g. ReportFile.update methods make good use of overloading). General good program practise has also been adhered to such as consistent and correct indentation of source code, blank lines to space out code, good naming of classes, methods, variables and good overall structure (no “spaghetti code” exists).

7 - TESTING

7.1 - Introduction

An important, and sometimes overlooked part of any software system is the testing phase.

“Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation.”

Pressman (2001)

Before a system can be delivered to users it is critical that the application is thoroughly tested and evaluated. Although no amount of testing will guarantee 100% error free programs it does expose most problems that generally are not obvious to developers and designers alike.

7.2 - Testing Objectives

The main testing objective is to uncover errors in the software. At the same time testing demonstrates that the software functions according to the specification of the system. Data files will be examined and this will provide a level of software reliability and some indication of software quality as a whole. It must be pointed out however that testing cannot show the lack of errors; it only shows that software errors and defects exist.

7.3 - Testing Strategies

For professional testing to take place it is important that a structured and consistent testing strategy is followed. Various different testing strategies exist. A few of these are outlined as follows:

7.3.1 - White Box Testing

White box testing uses the control structure of the procedural design to create test cases. The program can create test cases, which ensure that all paths through a module/class have been executed. It also ensures all logical decisions have been checked, all loops have been checked for their boundaries and internal data structures are validated.

7.3.1 - Black Box Testing

Black box testing is also known as behavioural testing. It focuses more on the functional requirements of a system. Black box testing is usually used along side white box testing (it is not really an alternative to it). It tries to find errors in incorrect or missing functions, interface errors, errors in data structures, behaviour / performance errors and initialisation errors. This type of testing is also generally done at the latter stages of testing. It does not concentrate on the control structure of a system but rather concentrates on the data.

7.4 - Unit Testing

During the implementation of the system a high amount of unit testing was carried out. This system is an online web based system so testing was carried out online and offline. Generally off-line testing involved testing each of the functional classes and the data classes. For example the QueryGenerator class has one main method (getSQL()), which returns a String. When testing the QueryGenerator class the following little off-line program (known as a test harness) can display the SQL a report file creates with very little code (see following code and Figure 7.1).

```
// This little program performs an offline test on the QueryGenerator
// Class and returns an SQL String

public class OfflineTestQueryGenerator {
    public static void main (String [] args) {
        QueryGenerator queryGenerator =
            new QueryGenerator ("northwind.gor");

        System.out.println (queryGenerator.getSQL());
    }
}
```

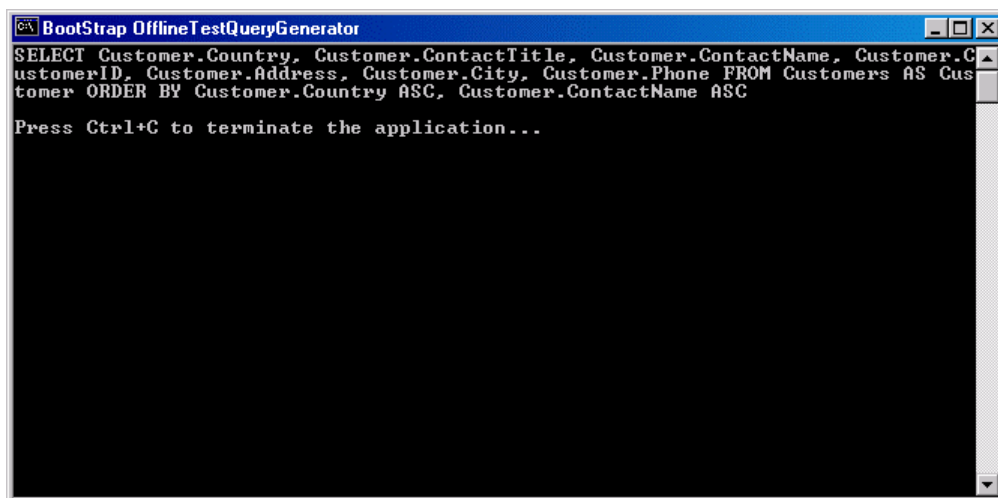


Figure 7.1

Another file was also used for more general testing. A number of different classes in the system were unit tested in this way. This included TablesAndAlias, Relationships, Query, Groups, Labels (see code and Figure 7.2), Format, ReportFile, DatabaseMetaData, SimpleENC to name a few.

```
// This little program performs an offline test on gernal classes.

public class OfflineTestGeneral {
    public static void main (String [] args) {
        ReportFile reportFile = new ReportFile ("northwind.gor");

        Labels labels = reportFile.getLabels();
        labels.setCurrentSetOfLabels(0);    // English

        System.out.println ("Title: " + labels.getTitle());
        System.out.println ("Subtitle: " + labels.getSubTitle());
        System.out.println ("Footer: " + labels.getFooter());
    }
}
```

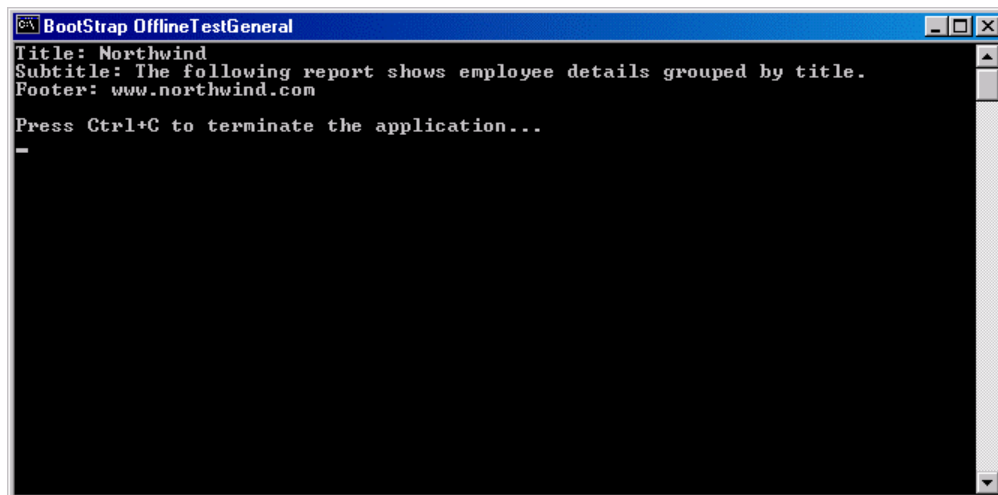


Figure 7.2

This type of testing proved to be very valuable in debugging the code, and helped to construct a very robust system. However this method of testing only proved suitable for testing all the functional and data classes in the system. All of the interfaces of the system had to be tested online.

7.5 - Integration Testing

This type of testing is an intermediate phase between module testing and full system testing. This basically involved testing parts of the system that integrated other parts together. For

example testing the QueryGenerator algorithm required the unit testing of the Tables, Relations, Query and Group sections to have been completed.

Integration testing also involves testing the interfaces between modules and the interface. In this system, this involved setting up the web servers, installing classes onto them and testing each interface of the system and its effects on the data files.

The following shows a sample test case, which were carried out.

Test Case 1	Change the existing tables and aliases of a sample report.
Preconditions	There must be at least one set of tables and aliases already available.
Test Prodedures	Click on the Report – Edit – Tables section of the system and add / amend the existing tables and aliases. Check the interface displays the new set OK and also check the data in the report file.
Post conditions	There must exist at least one alias and table and these must be different from original set.
Variations	Do the same test for the Relations section of the system.
Result	Test worked successfully.

7.6 - Full system Testing

Once all the components had been tested to a satisfactory level the system was tested as a whole. Full system testing generally goes beyond the correctness of the system and deals with issues such as the overall performance, business goals and high-level requirements.

In this system this involved the general steps:

1. All existing classes were deleted, existing database connection and report files taken off the server.
2. A new build of the system was done, i.e. recompile all of the source code.
3. All the classes were copied onto the server.
4. Each screen in the system was checked and ensured there were no inconsistencies and that the action did the desired task e.g. editing the table and aliases section creates the desired changes in the file.
5. If any changes were needed, the appropriate changes to the code were made and this phase was then restarted.

7.7 - Conclusion

After the implementation and all the testing of this system was completed, a very solid and reliable system was produced. A methodical and intensive process of testing, along with a high level of attention to detail ensured that this system was well tested and to a high standard. Due to the size of the system it was expected that there would be lots of errors and defects. There were however not as many as first thought. It is felt that this was due to the large nature of modularity that this project had. Most elements of the system were well separated from each other. For example the Labels section is totally separate from the Format section. This ensured that large amounts of unit testing removed the majority of the bugs in the system.

8 - EVALUATION

8.1 - Introduction

This chapter involves a detailed evaluation of the system. It will discuss its strengths and weaknesses from the developer's perspective (my own), and also from a users perspective. Other issues that will be discussed are additional functionality that users would like to see in system.

8.2 - User Evaluation

A good user evaluation of the system is important as it gives others the chance to see and make use of the system. It provides them a means to comment on the various areas of the system. These include the level of system functionality, interface and ease of use.

To achieve better results, different types of users were used in the evaluation. They were classified as follows:

1. **Novice** - These types of users have limited computer experience. They are expected to have used a database before and be able to create simple one-table queries. These users don't study computers at a high level.
2. **Amateur** - These types of users must have a sound understanding of most areas of computers, especially database systems. They are expected to have created database queries using more than one table in a visual manner (e.g. MS Access query-by-example). They are also expected to be familiar with editing capabilities used to format documents and thus be able to format a report easily. These include other students in the BSc Computing Science class.
3. **Expert** - An expert user is expected to be proficient in most areas of computers. They should be able to create complex queries using several tables in a visual manner or using SQL statements. These include a few individuals in the BSc Computing Science class and lecturers.

Eight different users in total evaluated the system. Each of them had to carry out the following steps:

1. The users were each given a short five-minute demo of the system.

2. The users were then handed a three-page evaluation guide (See Appendix D). This instructed them to create an ODBC connection to the Northwind database (example database which comes free with MS Access), log onto the system, create a database connection file, create a report etc.
3. Each of the users completed the various tasks described on guide. They were expected to complete all of the tasks on their own. Limited help was only offered if they were completely stuck.
4. After all the tasks had been completed, the users then filled in an evaluation form, which was in the form of a questionnaire (see Appendix E).
5. A short interview was then carried out with questions covering key sections of the system such as interface design, system functionality, elements which could be changed and additional functionality which could be incorporated into the system in the future.

Eight users in total evaluated the system (2 novice, 3 amateur and 3 expert). The results of the user evaluations were very satisfactory. The expert users thought the system was very good and gave 5 out of 5 in most areas. The novice users found problems with the system, especially creating a query, but this was mainly due to inexperience in databases in general. The following graph (Figure 8.1) shows the number of users who give full marks for the following areas of the system:

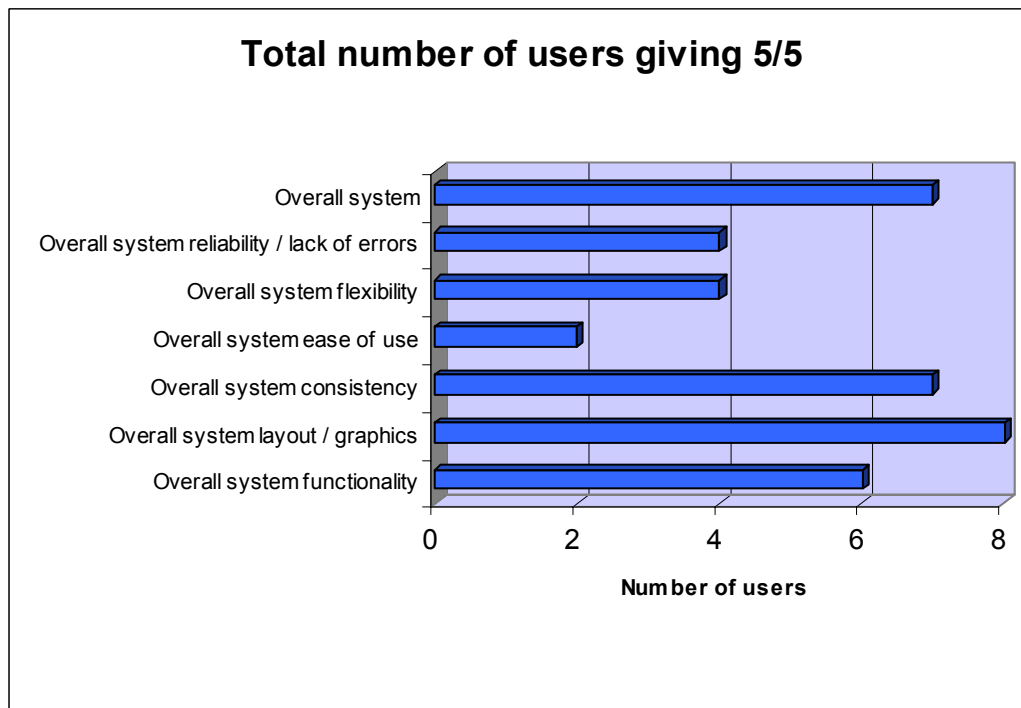


Figure 8.1

The graph shows 7 out of 8 people give the system 5 / 5 overall. It also shows that all the users give 5 / 5 for the overall system layout and graphics.

One thing which most of the users did not use was the Quick Help at the bottom of the screen. Only one of all the users (novice user) used it frequently. Any of the users that did use it thought it was of limited help. Most of them did not use the help, as they thought the system was intuitive.

8.3 - Developers Evaluation

The developer evaluation of the program will give a different perspective on the system compared to the user evaluation. It will not be as accurate as the user evaluation because it may be slightly biased (although hopefully this isn't intentional). This is mainly because the developer has been involved with the system from inception to implementation. The developer knows the system inside out and can do any task quickly and easily. Compared to a user who uses the system for the first time, he doesn't need to learn the interface and the order that tasks need to be carried out in.

From a developers view, it was felt that the system carried out almost everything that it intended to do. It must be said that this system is complex, but a great deal of effort has been carried out to ensure that its complexity has been distributed evenly using sub systems e.g. The Report – Edit section of the system has 7 sections in total. This makes using the system much easier and more understandable to use as it splits large tasks (e.g. creating a report) into a number of smaller tasks. Some users have criticised the fact that creating a query involves using 3 different screens (Tables, Relations and Query), whereas Access and other database programs do this using one screen. This could be done using HTML in one screen but it would have required some incredible complex JavaScript (this was discovered in the design) and was deemed unsuitable. Having said that, other users found creating a query using this system easier than MS Access.

Parts of the systems which are felt have been implemented to a great degree include the Report - Edit - Format section. This section of the system was designed without an exact idea of what was required and ended up being innovative.

8.4 - Enhancements and Recommendations

As with most computer systems, the list of additional enhancements and recommendations would be almost endless. The first recommendations for the system would be to add the two functional requirements, which weren't fully implemented (see chapter 3.3 for all the functional requirements).

These two requirements are:

1. *“Provide limited report interaction for users using run-time parameters so they are able to redefine a query”.*

Run time parameters allow users to have a limited amount of interaction with a report. For example when a report runs, a user can be required to set two dates and the report would only return values in this range. For a developer to implement this functionality the QueryGenerator algorithm class (see section 5.5.1) would need to be updated to cope with a dynamic query. The Report – Edit – Query interface would need updated so a user can specify that a column is a run time parameter (see Figure 8.2 for an example design). The Report – Run interface would need to be updated so a user can specify the values for the run time parameter when he is running a report (see Figure 8.3 for an example design).

S	TABLES AND COLUMNS		SORT	CON	VALUE
<input checked="" type="checkbox"/>	HOUSE	HOUSE_ID	-	-	[RTP]
<input type="checkbox"/>	HOUSE	ADDRESS	-	-	

Figure 8.2

Full Screen: <input type="radio"/> Yes <input checked="" type="radio"/> No	Language: English (UK)	Refresh
HOUSE.HOUSE_ID		

Bob Housing

Figure 8.3

This could be improved further by updating the Report – Edit – Labels section so that a user can specify a textual description for the run time parameter in different languages (see figure 8.4 for an example design). This in turn makes running a report more understandable to users (see Figure 8.5).

Labels - English (UK)	
Title	Bob Housing
Subtitle	Following report shows all houses less than £100,000
[RTP] - HOUSE.HOUSE_ID	Please enter the House ID:
TOWN.TOWN_DESC	Town
STYLE.STYLE_DESC	Style

Figure 8.4

Full Screen: <input type="radio"/> Yes <input checked="" type="radio"/> No	Language: English (UK)	Refresh
Please enter the House ID: <input type="text"/>		

Bob Housing

Figure 8.5

2. “Reports must be viewable as HTML. Like many pages that exist they must be print-friendly if a user wishes to have a hardcopy. Other formats may be considered if there is adequate time (e.g. X.M.L., P.D.F.)”.

The majority of this functional requirement was completed. Reports are available as HTML pages and if the full-screen option is checked then a user can print the report out from Internet browser. However the reports aren't available in PDF format or XML. As the functional requirements this would have only been completed if there was adequate time and this was the case. If a developer were to extend this project to allow reports to be in alternative formats then the ReportGenerator algorithm class (section 5.5.2) would have to be updated to allow for this. The Report – Run interface would need an extra format option so a user can select the desired format of a report. Figure 8.6 shows a possible design using a pull-down.

Full Screen: <input type="radio"/> Yes <input checked="" type="radio"/> No	Language: English (UK)	Format: HTML	Refresh				
<table border="1"> <tbody> <tr> <td>HTML</td> </tr> <tr> <td>MS Word</td> </tr> <tr> <td>PDF</td> </tr> <tr> <td>XML</td> </tr> </tbody> </table>				HTML	MS Word	PDF	XML
HTML							
MS Word							
PDF							
XML							

Sample Report

Following report shows all houses less than £100,000

Figure 8.6

Most of the existing sections of the system could be enhanced in some form or other. This includes the query section (Tables, Relations and Query), formatting and even the full system interface which all could be improved in some way or other.

The query section could be enhanced significantly. The most obvious enhancement, based on user opinion, would be to combine these three screens into one. Another enhancement would be to allow more sophisticated queries using operators such as LIKE which searches strings for smaller strings etc.

Another important enhancement would be declaring totals for the bottom of columns in a query e.g. average, sum, count etc. This functionality is important if the data is number intensive.

To create a report using any query of unlimited complexity the user would then need to type the SQL statement himself. This however goes against the functional requirement *“Users must have to ability to create complex queries for the report data visually by using html forms (No experience of SQL necessary).”* This would make the system really powerful, but in turn would create additional problems to deal with this new level of complexity. Presently the system creates a SQL statement from data created from the Tables, Relations, Query and Group sections using the QueryGenerator algorithm (see chapter 5.5.1). If a user has the ability to specify their own SQL statement then the following scenario arises. Does the system use an algorithm to generate the correct values in the Tables, Relations section etc? This is basically doing the reverse of the QueryGenerator algorithm. This could perhaps be solved by forcing the user to either use the query-by-example style of query or by manually typing the SQL. Figure 8.7 shows an example design that could provide this functionality.

Figure 8.7

The format section of the system is another obvious area of the system that could be considerably enhanced. Currently the format section can handle different fonts, font sizes, bold, italics, underline, horizontal alignment, lines, different colours for text, background and lines. Additional elements, which could be added, include images and vertical lines. The present

system splits the report into various sections and there can be only one font, colour etc for each section. This level of functionality is very limited compared to the absolute positioning of elements which programs like MS Access offer (Figure 8.8).

The screenshot shows a window titled "Invoice : Report". The form is divided into several sections:

- Page Header:** Contains the Northwind Traders logo, address ("One Peninsula Way, Twin Points WA 98156"), phone ("1-206-335-1411"), fax ("1-206-335-3930"), and a date field ("Date: (mmm-yyyy)").
- OrderID Header:** Contains fields for "Ship To" (ShipName, ShipAddress, ShipCountry) and "Bill To" (Customers.CompanyName, Address, Country).
- Detail:** A table with columns: Order ID, Customer ID, Salesperson, Order Date, Required Date, Shipped Date, Ship Via, Product ID, Product Name, Quantity, Unit Price, Discount, and Extended Price.
- OrderID Footer:** Contains summary fields: Subtotal (=((ExtendedPrice))), Freight, and Total (Subtotal+Freight).

Figure 8.8

To add the formatting functionality that programs like MS Word would probably require the use of a Java Applet or some incredible JavaScript.

8.5 - Conclusion

The evaluation of this system has proved invaluable for retrieving feedback in the many different areas of the system. It was interesting to see the various opinions coming from the varying range of users. It seems this system is really only suitable for users with a good grasp of creating a database queries, as this was an area quite a few users had problems in. The novice users had initial problems with most areas of the system, but these same users would have problems with any new piece of software. Novice users also found it very easy to logon, open a report and run it, which in a real world scenario is what this system would be used to must (simply running reports).

9 - CONCLUSION

9 - Project Summary

This chapter will conclude this project report. The aim of this project will be compared with the actual achievements made. The report will then end with a final note from the author as he reflects over the project as a whole.

9.1 - Review of Aims And Achievements

The aim from the very start of the project (see section 1.2) is as follows:

“To create a highly generic online reporting system that solves problems with existing reporting technology e.g. lack of multiple language support, portability and ease of use for users in today’s online world.”

As this project draws to an end, we will review this statement and see if has been achieved. The project has been successfully implemented, tested and evaluated by various users. Areas of language support have been addressed through the use of report labels. The system is also available in various languages, so users that use different languages such as German can make use of the system. Portability has also been addressed by using Java Servlets in the implementation, which are server independent. Finally, implementing a good menu system, consistent interface and spreading functionality over various subsystems have also addressed ease of use.

9.2 - Authors Final Note

This project has been a slightly stressful but otherwise a great experience from start to finish. As mentioned earlier, this project was my own idea and when I was designing the system I realised there was a lot of work to be done. However I managed my time well and completed the majority of the requirements. I feel I achieved this with a solid design, an understanding of what was to be accomplished and a strong amount of expertise of the technologies used in the implementation of the system. The area of this project in which I would be best in and enjoy the most would have to be programming and I feel the quality of the implementation shows that.

I have enjoyed doing certain sections of the system more than others. My favourite section of the system, and the one I have enjoyed designing and implementing the most would have to be Report - Edit - Format section of the system. This was one of the most complex sections and once it had been completed successfully it was a very satisfying experience. Another part of the system that I enjoyed creating was the QueryGenerator algorithm (see section 5.5.1). It was quite long and complex, but still fairly straightforward to design and implement.

A section of the systems, which I did not enjoy, were creating the system interface to be available in various languages (see section 6.5 point 6). Although the interface looks very impressive, when it is running under different languages, it was relatively easy to implement, but required quite a considerable amount of work. This resulted in doing the same type of work over and over to create the output.

I believe that in striving to create a generic system, I have succeeded. This can be seen from the finished system as it addresses issues such as multi-language support, multi-database functionality and server independence.

I feel that in the future a greater number of applications will make use of the Internet. Browser technology will become increasingly important and successful integration with all applications will be necessary.

To finally conclude this report I would like to thank all involved (see acknowledgements) who have encouraged and helped me with this project.

10 - BIBLIOGRAPHY

- Pressman, R.S. (2001), *Software Engineering - A Practitioner's Approach* 5th Edition, McGraw-Hill Publishing Co., London, 0-07-709677-0,
- Sommerville, I.S., (2001) *Software Engineering – 6th Edition*, Addison-Wesley, England, 0-201-39815-X
- Deitel, H.M, Deitel, P.J, Nieto, T.R., (2000). *Internet & World Wide Web – How to Program*, USA, 0-130-161438
- CommerceNet - <http://www.commerce.net/research/status/wwstats.html>, 28th, October 2001.
- <http://java.sun.com>

Appendix A – Format Interface Design Screenshots

The final design was created using an evolutionary approach with various increments. The following show 8 of these from inception until completion. All of these prototypes are available on the CD, should which accompanies this report.

Prototype 1

- There was no real idea how to start.
- Structure was taken from MS Access reporting section but quickly released this wasn't working that well.

REPORT - HEADER
-HOUSE.COUNTY_DESC - HEADER
-DETAIL
-HOUSE.COUNTY_DESC - FOOTER
REPORT - FOOTER

(format1.htm)

Prototype 2

- Starting to create common formatting objects using HTML form elements. For example the font and font size uses pull-down lists, bold and italics options as checkboxes, alignment and colours as radio buttons (only one can be selected).
- Still very limited functionality.

Formatting bit of the program	
Times New Roman	2 B I [align icons] [bullet icon]
[color icons]	
Title	<input type="text"/>
SubTitle	<input type="text"/>

(format2.htm)

Prototype 3

- Ambitious step forward where the first two prototypes were combined into one interface.
- Title and subtitle added as textboxes.
- Added overall page orientation option (width of page). This could be either landscape or portrait.
- Lines added.
- Still unsure how each section was going to be formatted.
- Very little JavaScript functionality in interface.

Font Default Font None

B **I** **U**

REPORT PROPERTIES + DEFAULTS

Default Language French

***Title**

Sub Title

Page ☒ Portrait ☐ Landscape

Background Colours Page Background Head Background Body Background

Lines (size) TOP: Page Header 0 Page Footer 0 Page Detail 0
BOTTOM: Page Header 0 Page Footer 0 Page Detail 0
Not Specified

REPORT - HEADER Num of Lines 3

-HOUSE.COUNTY_DESC - HEADER

-DETAIL

<HOUSE.HOUSE_ID> <HOUSE.ADDRESS> <STYLE.STYLE_DESC>

-HOUSE.COUNTY_DESC - FOOTER

REPORT - FOOTER

(format3.htm)

Prototype 5

- Line pull-downs removed from each section and placed at top with other formatting elements.
- Matrix now holds the following information in its columns 0 to 11 (12):
 0. Font - 0 to 5 (Default Font, Arial, Times New Roman, Courier New, Georgia, Veranda)
 1. Font Size - 0 to 7
 2. Bold - 0 or 1 (No or Yes)
 3. Italics - 0 or 1 (No or Yes)
 4. Underline - 0 or 1 (No or Yes) *NEW*
 5. Alignment - 0, 1 or 2 (Left, Middle or Right)
 6. Line Top Size - 0 to 9 (0 = no line, 1 to 9 size of line in pixels) *NEW*
 7. Line Bottom Size - 0 or 9 (0 = no line, 1 to 9 size of line in pixels) *NEW*
 8. Text Colour - 0 to 15 (16 different colours in total)
 9. Background Colour - 0 to 15 (16 different colours in total)
 10. Lines Top Colour - 0 to 15 (16 different colours in total) *NEW*
 11. Lines Bottom Colour - 0 to 15 (16 different colours in total) *NEW*

Font	Default Font	None	Lines	Text	Background	Line-Top	
B	<input type="checkbox"/>	I	<input type="checkbox"/>	U	<input type="checkbox"/>	Top: -	Bottom: -

FORMAT (Click on a line to format it)	
<input type="radio"/>	Title
<input type="radio"/>	Sub Title
<input type="radio"/>	<i>blank line</i>
<input checked="" type="radio"/>	GROUP (COUNTY.COUNTY_DESC): <input checked="" type="radio"/> <LABEL> at top ...
<input type="radio"/>	... <DATA> below <input type="radio"/> <LABEL>: <DATA> (if selected, ignore previous line)
<input type="radio"/>	DETAIL: <input checked="" type="radio"/> All other labels at top ...
<input type="radio"/>	... <DATA> below <input type="radio"/> <LABEL>: <DATA> (if selected, ignore previous line)
<input type="radio"/>	<i>blank line</i>
<input type="radio"/>	Footer
PROPERTIES	
Page: <input checked="" type="radio"/> Portrait <input type="radio"/> Landscape	
Submit	

(format5.htm)

Prototype 6

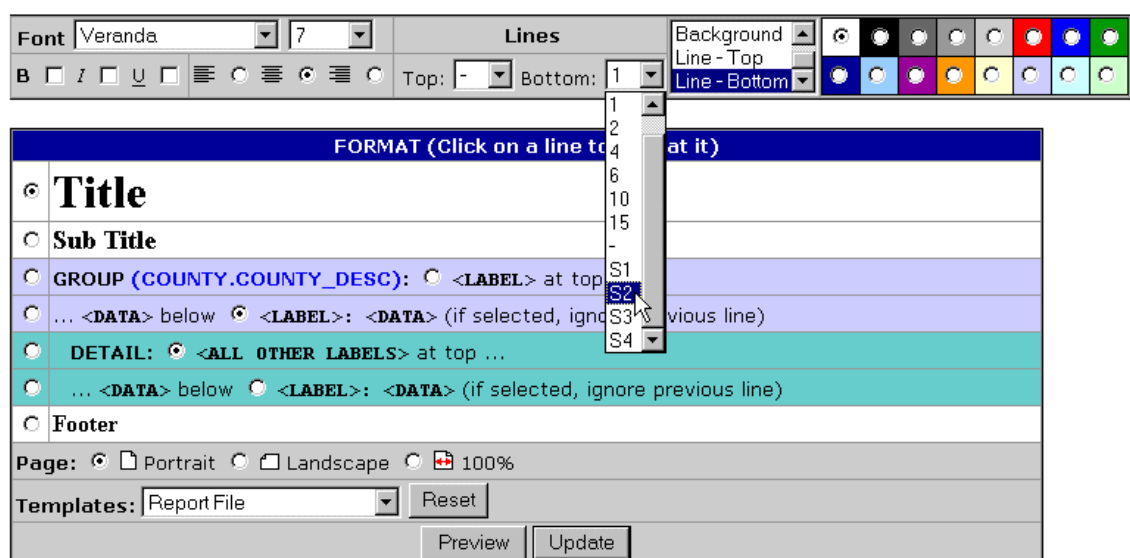
- Added template pull-down, which resets the Matrix with a new set of numbers.
- Important “Preview” option added (Figure x). This uses JavaScript to translate the JavaScript Matrix into a little preview screen. Up to this point a user had no immediate way of visualising what his formatting look like.

(format6.htm)

(preview.htm)

Prototype 7

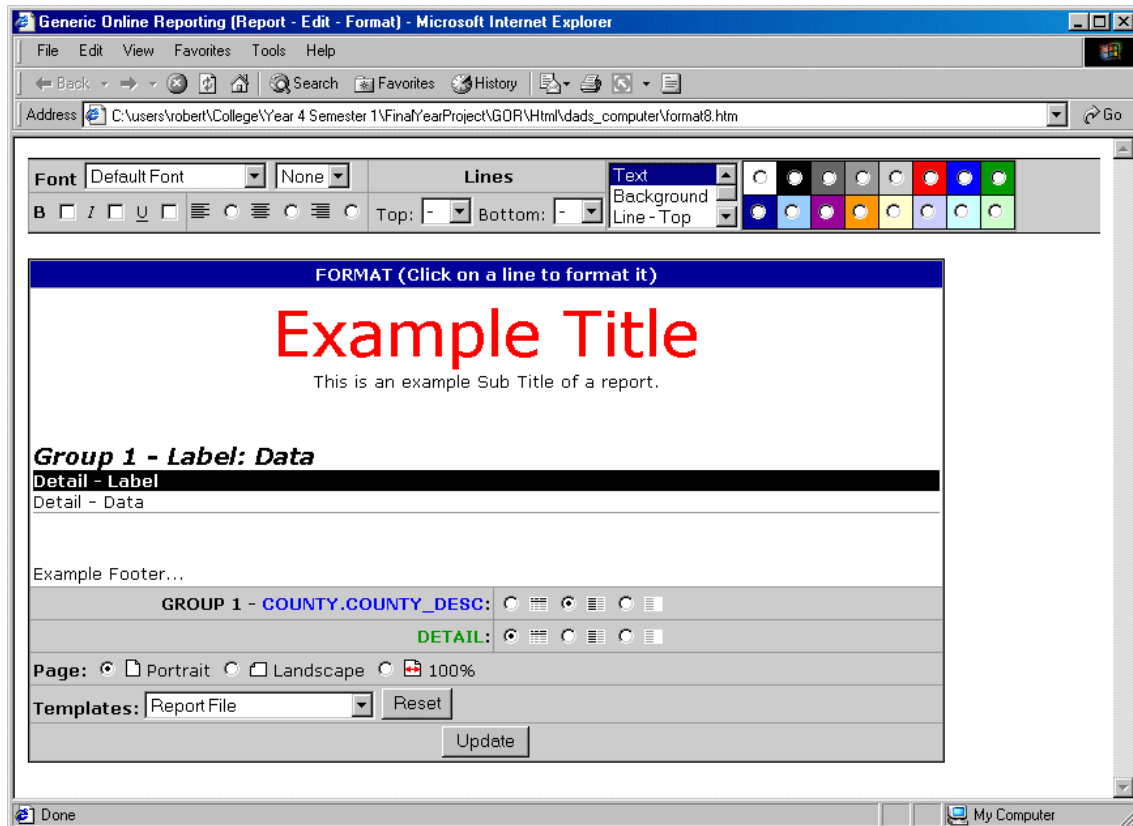
- Removed blank lines from screen and Matrix rows.
- Line size pull-downs (top and bottom) changed to compensate:
 0. 0 - no line
 1. 1 to 6 - exact pixel lines (1 = 1 pixels, 2 = 2 pixels, 3 = 4 pixels, 4 = 6 pixels, 5 = 10 pixels and 6 = 15 pixels)
 2. 7 - no line
 3. 8 to 11 - spacer lines (8 = 1 spacer line , 9 = 2 spacer line, ...)
- Colour values changed to get a good mix of greys, strong colours and contrasting pastel colours.
- 15 rows finalised in Matrix:
 - 0, 1 - title, subtitle
 - 2, 3 - group 1 labels, group data (or labels and data if option selected) ...
 - 4 to 11 - ... group 2 to 5 formatting
 - 12, 13 – detail labels, detail data (or labels and data if option selected).
 - 14 – footer
 - **NOTE:** Rows 2 to 11 may not be used but will still exist.



(format7.htm)

Prototype 8 (FINAL DESIGN APPROVED)

- Preview screen and main screen connected together. This ensured real-time previewing of formatting (e.g. similar to MS Word).
- Orientation of groups separated from formatting.



(format8.htm)

This is the final prototype and was implemented into the system (ReportEditFormatServlet) with a few very minors tweaks. This is a good example where the evolutionary approach to design can be very useful.

```
<head>  
<title>Generic Online Reporting (Report - Edit - Format)</title>  
<link href='styles.css' type=text/css rel=stylesheet>  
<meta http-equiv='Content-Type' content='text/html; charset=iso-8859-1'>  
</head>  
<body bgcolor='#FFFFFF' text='#000066'>  
<table width='100%' border='0' cellpadding='0' cellspacing='0'>  
  <tr bgcolor='#000000'>  
    <td><img src='images/lp.gif' width='1' height='1'></td>  
  </tr>  
  
  <script language="JavaScript">  
  
// declare 2 dimensional array to hold information about each line in report format  
// HORIZONTAL =  
// (0=font, 1=font_size, 2=bold, 3=italics, 4=underline,  
// 5=align, 6=t.line thickness, 7=b.line thickness,  
// 8=text_colour, 9=background_colour, 10=t.line colour, 11=b.line colour)  
// VERTICAL = each row in report  
  
lines_info = [  
  [5,7,0,0,0,1,1,1,5,0,0,0],  
  [5,2,0,0,0,1,0,9,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,1,0,0,0,0,0,0,1,0,0],  
  [5,2,0,0,0,0,0,0,1,1,0,0,3],  
  [5,2,0,0,0,0,0,9,0,1,0,0,0]  
];  
groups = [1,0,0,0,0,0];  
  
template_0 = [  
  [5,7,0,0,0,1,1,1,5,0,0,0],  
  [5,2,0,0,0,1,0,9,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,4,1,1,0,0,0,0,1,0,0,0],  
  [5,2,1,0,0,0,0,0,0,1,0,0],  
  [5,2,0,0,0,0,0,0,1,1,0,0,3],  
  [5,2,0,0,0,0,0,9,0,1,0,0,0]  
];  
template_groups_0 = [1,0,0,0,0,0];  
  
template_1 = [  
  [5,5,0,0,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,8,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,2,1,0,0,0,0,0,2,1,0,0,1],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,2,1,0,0,0,0,0,2,1,0,0,1],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,2,1,0,0,0,0,0,2,1,0,0,1],  
  [5,2,0,0,0,0,0,0,1,0,0,0],  
  [5,2,0,0,0,0,0,0,1,0,0,0]
```

```

[5,2,1,0,0,0,0,2,1,0,0,1],
[5,2,0,0,0,0,0,0,1,0,0,0],
[5,2,1,0,0,0,0,2,1,0,0,1],
[5,2,1,0,0,0,0,0,1,1,0,0,1],
[5,2,0,0,0,0,0,0,1,1,0,0,1],
[5,2,0,0,0,0,0,8,0,1,0,0,0]
];
template_groups_1 = [1,0,0,0,0,0];

// Veranda - causal (blues)
template_2 = [
[5,6,1,0,0,0,0,2,2,8,12,1,1],
[5,2,1,0,0,0,0,0,8,3,0,13,0],
[5,2,0,0,0,0,0,0,0,8,0,0,0],
[5,2,1,0,0,0,0,0,0,8,0,1,1],
[5,2,0,0,0,0,0,0,0,8,0,0,0],
[5,2,1,0,0,0,0,0,0,8,0,1,1],
[5,2,0,0,0,0,0,0,0,8,0,0,0],
[5,2,1,0,0,0,0,0,0,8,0,1,1],
[5,2,0,0,0,0,0,0,0,8,0,0,0],
[5,2,1,0,0,0,0,0,0,8,0,1,1],
[5,2,0,0,0,0,0,0,0,8,0,0,0],
[5,2,1,0,0,0,0,0,0,8,0,1,1],
[5,2,0,0,0,0,0,0,0,8,0,0,0],
[5,2,1,0,0,0,0,0,0,8,0,1,1],
[5,2,1,0,0,0,0,2,2,8,9,1,1],
[5,2,0,0,0,0,0,0,1,3,0,4,3],
[5,2,0,0,0,0,0,8,1,1,0,0,8]
];
template_groups_2 = [1,0,0,0,0,0];

// Times new roman - Forest greens
template_3 = [
[2,7,0,1,0,2,4,4,7,15,7,7],
[2,3,0,1,0,2,0,8,1,0,0,0],
[2,2,0,0,0,0,0,0,0,11,0,0,0],
[2,3,1,1,0,2,8,8,1,0,0,0],
[2,2,0,0,0,0,0,0,0,11,0,0,0],
[2,3,1,1,0,2,8,8,10,0,0,0],
[2,2,0,0,0,0,0,0,0,11,0,0,0],
[2,3,1,1,0,2,8,8,10,0,0,0],
[2,2,0,0,0,0,0,0,0,11,0,0,0],
[2,3,1,1,0,2,8,8,10,0,0,0],
[2,2,0,0,0,0,0,0,0,11,0,0,0],
[2,3,1,1,0,2,8,8,10,0,0,0],
[2,2,0,0,0,0,0,0,0,11,0,0,0],
[2,3,1,1,0,2,8,8,10,0,0,0],
[2,2,1,0,0,0,1,1,1,15,7,7],
[2,2,0,0,0,0,0,0,1,1,0,0,4],
[2,2,0,1,0,1,9,1,1,0,0,7]
];
template_groups_3 = [1,0,0,0,0,0];

// Times - cooperates navy
template_4 = [
[2,6,0,1,0,1,3,3,0,8,4,4],
[2,2,0,1,1,1,8,9,1,0,0,0],
[2,4,1,1,0,2,0,3,1,0,0,8],
[2,3,0,1,0,2,0,0,1,0,0,1],
[2,4,1,1,0,2,0,3,1,0,0,8],
[2,3,0,1,0,2,0,0,1,0,0,1],
[2,4,1,1,0,2,0,3,1,0,0,8],
[2,3,0,1,0,2,0,0,1,0,0,1],
[2,4,1,1,0,2,0,3,1,0,0,8],
[2,3,0,1,0,2,0,0,1,0,0,1],
[2,4,1,1,0,2,0,3,1,0,0,8],
[2,3,0,1,0,2,0,0,1,0,0,1],
[2,4,1,1,0,2,0,3,1,0,0,8],
[2,3,0,1,0,2,0,0,1,0,0,1],
[2,2,1,0,0,0,1,1,1,4,1,1],
[2,2,0,0,0,0,0,0,1,1,0,0,4],
[2,2,0,1,0,1,8,1,1,0,0,4]
];
template_groups_4 = [0,0,0,0,0,0];

// Courier - Greys
template_5 = [
[3,7,1,0,0,1,1,1,1,4,1,1],

```

```

[3,2,0,0,0,1,0,8,1,0,0,0],
[3,2,0,0,0,0,0,0,1,0,0,0],
[3,4,1,0,0,0,0,0,1,0,0,0],
[3,2,0,0,0,0,0,0,0,8,0,0],
[3,2,0,0,0,0,0,0,0,8,0,0],
[3,2,0,0,0,0,0,0,0,8,0,0],
[3,2,0,0,0,0,0,0,0,8,0,0],
[3,2,0,0,0,0,0,0,0,8,0,0],
[3,2,0,0,0,0,0,0,0,8,0,0],
[3,2,0,0,0,0,0,0,0,8,0,0],
[3,2,1,0,0,0,0,0,0,8,0,0],
[3,2,0,0,0,0,0,0,0,8,0,0],
[3,2,1,0,0,0,0,0,0,1,0,0],
[3,2,0,0,0,0,0,0,1,1,0,0,4],
[3,2,0,0,0,0,8,0,1,0,0,0]
];
template_groups_5 = [1,0,0,0,0,0];

// Swinging 60s!
template_6 = [
[1,7,1,1,1,1,5,5,7,12,10,10],
[1,2,1,0,0,1,0,8,11,15,0,15],
[1,2,0,0,0,0,0,0,1,0,0,0],
[1,4,1,0,0,0,0,0,5,15,0,0],
[1,2,0,0,0,0,0,0,1,0,0,0],
[1,4,1,0,0,0,0,0,5,15,0,0],
[1,2,0,0,0,0,0,0,1,0,0,0],
[1,4,1,0,0,0,0,0,5,15,0,0],
[1,2,0,0,0,0,0,0,1,0,0,0],
[1,4,1,0,0,0,0,0,5,15,0,0],
[1,2,0,0,0,0,0,0,1,0,0,0],
[1,4,1,0,0,0,0,0,5,15,0,0],
[1,2,0,0,0,0,0,0,1,0,0,0],
[1,4,1,0,0,0,0,0,5,15,0,0],
[1,2,1,0,0,0,2,2,8,9,1,1],
[1,2,1,0,0,0,0,1,6,13,0,1],
[1,2,1,1,1,0,8,5,7,12,12,10]
];
template_groups_6 = [1,0,0,0,0,0];

function resetFromTemplate () {
    index = document.mainForm.TEMPLATES.selectedIndex;
    temp_lines = eval("template_" + index);
    temp_groups = eval("template_groups_" + index);
    for (i = 0; i < 15; i++) // go through each line
        for (j = 0; j < 12; j++) // go through each element
            lines_info[i][j] = temp_lines[i][j];
    for (g = 0; g < 6; g++)
        groups[g] = temp_groups[g];
    focus = -1;
    updateHiddenField();
    refreshGroups();
}

// variable to hold which element is in focus
// -1 = nothing
// 0, 1, ..., x = page elements
// and variable to hold which colour element is in focus.

var focus=-1; // default focus (not focused on anything)
var numOfGroups = 1;
var colourFocus=0; // focus of which colour is selected.

// This is the little function for updating the right selection for each colour
function showColour() {
    var index = lines_info[document.mainForm.COLOURS.selectedIndex];
    document.mainForm.COLOUR[index].checked = true;
}

// If a textbox was clicked then you must update the font settings etc.
function lineFocus(line) {
    focus = line; // declare which line is in focus.

    document.mainForm.FONT.selectedIndex = lines_info[focus][0]; // font
    document.mainForm.FONT_SIZE.selectedIndex = lines_info[focus][1]; // font size

```

```

        if (lines_info[focus][2]== 1)                                // bold
            document.mainForm.BOLD.checked = true;
        else
            document.mainForm.BOLD.checked = false;
        if (lines_info[focus][3]== 1)                                // italics
            document.mainForm.ITALICS.checked = true;
        else
            document.mainForm.ITALICS.checked = false;
        if (lines_info[focus][4]== 1)                                // underline
            document.mainForm.UNDERLINE.checked = true;
        else
            document.mainForm.UNDERLINE.checked = false;
        document.mainForm.ALIGN[lines_info[focus][5]].checked = true;    // align
        document.mainForm.LINE_TOP.selectedIndex = lines_info[focus][6];    // lines top and
        ...
        document.mainForm.LINE_BOTTOM.selectedIndex = lines_info[focus][7]; // ... bottom
        refreshColours();
    }

    // methods to respond to setting a font, font size, bold, align, italics,
    // alignment, lines and various colours. Each of these up the array
    function setFont() {
        if (focus >= 0) {
            lines_info[focus][0] = document.mainForm.FONT.selectedIndex;
            updateHiddenField();
        }
    }
    function setFontSize() {
        if (focus >= 0) {
            lines_info[focus][1] = document.mainForm.FONT_SIZE.selectedIndex;
            updateHiddenField();
        }
    }
    function setBold() {
        if (focus >= 0) {
            if (document.mainForm.BOLD.checked == true)
                lines_info[focus][2] = 1;
            else
                lines_info[focus][2] = 0;
            updateHiddenField();
        }
    }
    function setItalics() {
        if (focus >= 0) {
            if (document.mainForm.ITALICS.checked == true)
                lines_info[focus][3] = 1;
            else
                lines_info[focus][3] = 0;
            updateHiddenField();
        }
    }
    function setUnderLine() {
        if (focus >= 0) {
            if (document.mainForm.UNDERLINE.checked == true)
                lines_info[focus][4] = 1;
            else
                lines_info[focus][4] = 0;
            updateHiddenField();
        }
    }
    function setAlign(align) {
        if (focus >= 0) {
            lines_info[focus][5] = align;
            updateHiddenField();
        }
    }
    function setColourChooser() {
        if (focus >= 0) {
            colourFocus=document.mainForm.CURCOLOUR.selectedIndex;
            refreshColours();
            updateHiddenField();
        }
    }

```

```

}
function setLineTop()
{
    if (focus >= 0) {
        lines_info[focus][6] = document.mainForm.LINE_TOP.selectedIndex;
        updateHiddenField();
    }
}
function setLineBottom()
{
    if (focus >= 0) {
        lines_info[focus][7] = document.mainForm.LINE_BOTTOM.selectedIndex;
        updateHiddenField();
    }
}
function setColour(colour) {
    if (focus >= 0) {        // CURCOLOUR
        lines_info[focus][8 + colourFocus] = colour;
        updateHiddenField();
    }
}

// updates the corect column for colour
function updateColourChooser(item) {
    if (focus >= 0) {
        colourFocus=item;
        document.mainForm.CURCOLOUR[item].selected = true;
        refreshColours();
    }
}
// refreshes colours
function refreshColours() {
    if (focus >= 0) {
        document.mainForm.COLOUR[lines_info[focus][8 + colourFocus]].checked = true;    //
    }
}
// update group.
function setGroup (index, value){
    groups[index] = value;
    updateHiddenField();
}

// refreshs groups depending on array values.
function refreshGroups (){
    for (i = 0; i < numOfGroups; i++)    // show each group
        eval("document.mainForm.GROUP_" + i + "[groups[" + i + "]].checked = true");
    document.mainForm.GROUP_5[groups[5]].checked = true;    // detail (always there)
}

// It updates the hidden field so the server can read all the
// array data for each line (fonts, size, etc )
function updateHiddenField() {
    showPreview();    // display update to screen
    var temp = "";
    for (i = 0; i < 15; i++) {
        for (j = 0; j < 12; j++)    // go through each line
            temp += lines_info[i][j] + ", ";    // go through each element
    }
    for (g = 0; g < 6; g++)
        temp += groups [g] + ", ";
    document.mainForm.ARRAY_DATA.value=temp;
}
// This function updates the preview screen depending on the values of the arrays.
function showPreview() {
//=====
// Constant array data
//=====

    colours = [
        "FFFFFF", "000000", "666666", "999999", "CCCCCC", "FF0000", "0000FF", "009900",
        "000099", "99CCFF", "990099", "FF9900", "FFFFCC", "CCCCFF", "CCFFFF", "CCFFCC"
    ]

```

```

];
fonts = [
    "Times New Roman, Times", "Arial", "Times New Roman, Times", "Courier New",
    "Georgia", "Verdana, Arial"
];
align = ["left", "middle", "right"];
textTop = ["Example Title", "This is an example Sub Title of a report."];
textBottom = "Example Footer...";
sData = "Data";
sLabel = "Label";
sGroup = "Group";
sDetail="Detail";

//=====
// Start creating html String
//=====

line = 0;
html =
    "<table cellpadding='0' cellspacing='0' width='100%'>\n";

// do first 2 lines - Title & Sub Title
for (line = 0; line < 2; line++)
    html +=
        addLine (line, 6) +
        "<tr align=" + align[lines_info[line][5]] + " bgColor='" +
colours[lines_info[line][9]] + "'>" +
        "<td onClick=\"lineFocus(" + line + ");\">" +
        fontStart (line) + textTop[line] + fontEnd (line) +
        "</td></tr>\n" +
        addLine (line, 7);

// create group html (if there is any)
for (g = 0; g < numOfGroups; g++) {
    line = 2 + (g * 2);
    if (groups[g] == 0)
        html +=
            addLine (line, 6) +
            "<tr align=" + align[lines_info[line][5]] + " bgColor='" +
colours[lines_info[line][9]] + "'>\n" +
            "<td onClick=\"lineFocus(" + line + ");\">\n" +
            fontStart (line) + sGroup + " " + (g+1) + " - " + sLabel + fontEnd (line) +
            "</td></tr>\n" +
            addLine (line, 7);
    line++;
    html +=
        addLine (line, 6) +
        "<tr align=" + align[lines_info[line][5]] + " bgColor='" +
colours[lines_info[line][9]] + "'>\n" +
        "<td onClick=\"lineFocus(" + line + ");\">\n" +
        fontStart (line) + sGroup + " " + (g+1) + " - ";
    if (groups[g] == 0)
        html += sData;
    else {
        if (groups [g] == 1)
            html += sLabel + ": ";
        html += sData;
    }
    html +=
        fontEnd (line) +
        "</td></tr>\n" +
        addLine (line, 7);
}

// create html for detail
line = 12;
if (groups[5] == 0)
    html +=
        addLine (line, 6) +

```



```

    "<tr align=" + align[lines_info[line][5]] + " bgColor='" +
colours[lines_info[line][9]] + "'>\n" +
    "<td onClick=\"lineFocus(" + line + ");\">\n" +
    fontStart (line) + sDetail + " - " + sLabel + fontEnd (line) +
    "</td></tr>\n" +
    addLine (line, 7);
    line++;
    html +=
    addLine (line, 6) +
    "<tr align=" + align[lines_info[line][5]] + " bgColor='" +
colours[lines_info[line][9]] + "'>\n" +
    "<td onClick=\"lineFocus(" + line + ");\">\n" +
    fontStart (line) + sDetail + " - ";
    if (groups[5] == 0)
        html += sData;
    else {
        if (groups [5] == 1)
            html += sLabel + ": ";
        html += sData;
    }
    html +=
    fontEnd (line) +
    "</td></tr>\n" +
    addLine (line, 7);

    // Create rest
    line++;
    html +=
    addLine (line, 6) +
    "<tr align=" + align[lines_info[line][5]] + " bgColor='" +
colours[lines_info[line][9]] + "'>\n" +
    "<td onClick=\"lineFocus(" + line + ");\">\n" +
    fontStart (line) + textBottom + fontEnd (line) +
    "</td></tr>\n" +
    addLine (line, 7);

    // finish and update DIVs
    html += "</table>\n";
    previewHtml.innerHTML=html;

//=====
// addLine

//=====

function addLine (line, tOrB) {
    var heights=[1,2,4,6,10,15];
    index = lines_info[line][tOrB];
    if (index > 0 && index < 7) {
        return (
            "<tr bgColor='" + colours[lines_info[line][(tOrB + 4)]] +
            "' height='" + heights[lines_info[line][tOrB] -1] + "'>" +
            "<td><img src='images/lp.gif' width='1' height='1'></td></tr>\n"
        );
    }
    else if (index > 7 && index < 12) {
        index = lines_info[line][tOrB];
        temp = "";
        for (i = 7; i < index; i++)
            temp += "<tr bgColor='" + colours[lines_info[line][(tOrB + 4)]] +
            "'><td>&nbsp;</td></tr>\n";
        return temp;
    }
    else return "";
}

//=====
// fontStart

//=====

```

```

function fontStart(line) {
    fontHtml = "";
    if (lines_info[line][2] == 1) fontHtml += "<b>";
    if (lines_info[line][3] == 1) fontHtml += "<i>";
    if (lines_info[line][4] == 1) fontHtml += "<u>";
    fontHtml +=
        "<font color='" + colours[lines_info[line][8]] +
        "' face='" + fonts[lines_info[line][0]] + "' " +
        "size = '" + lines_info[line][1] + "'>";

    return fontHtml;
}

//=====
// fontEnd

//=====

function fontEnd(line) {
    fontHtml = "";
    if (lines_info[line][4] == 1) fontHtml += "</u>";
    if (lines_info[line][3] == 1) fontHtml += "</i>";
    if (lines_info[line][2] == 1) fontHtml += "</b>";
    fontHtml += "</font>";

    return fontHtml;
}
}

</script>
<form method='post' action='' name='mainForm'>
    <tr>
        <td bgcolor="#CCCCCC">
            <table border="0" cellspacing="1" cellpadding="0" bgcolor="#999999">
                <tr bgcolor="#CCCCCC">
                    <td colspan="2" nowrap><font color="#FFFFFF" face="Verdana, Arial,
Helvetica, sans-serif" size="2"><b><font color="#CCCCCC" face="Verdana, Arial,
Helvetica, sans-serif" size="2">&nbsp;</font><font
color="#000000">Font</font></b></font><font size="2" face="Verdana, Arial, Helvetica,
sans-serif">
                        <select name="FONT" onChange="setFont();">
                            <option value="0" selected>Default Font</option>
                            <option value="1">Arial</option>
                            <option value="2">Times New Roman</option>
                            <option value="3">Courier New</option>
                            <option value="4">Georgia</option>
                            <option value="5">Veranda</option>
                        </select>
                        <select name="FONT_SIZE" onChange="setFontSize();">
                            <option value="2" selected>None</option>
                            <option value="1">1</option>
                            <option value="2">2</option>
                            <option value="3">3</option>
                            <option value="4">4</option>
                            <option value="5">5</option>
                            <option value="6">6</option>
                            <option value="7">7</option>
                        </select>
                    </font></td>
                    <td align="center" nowrap><font color="#FFFFFF" face="Verdana, Arial,
Helvetica, sans-serif" size="2"><b><font color="#CCCCCC" face="Verdana, Arial,
Helvetica, sans-serif" size="2">&nbsp;</font><font
color="#000000">Lines</font></b></font>
                    </td>
                <tr>
                    <td colspan="2" nowrap>
                        <select onChange="setColourChooser();" name="CURCOLOUR" size="3">
                            <option value="0" selected>Text</option>
                            <option value="1">Background</option>
                            <option value="2">Line - Top</option>
                            <option value="3">Line - Bottom</option>
                        </select>
                    </td>
                </tr>
            </table>
        </td>
    </tr>

```

```

        </select>
      </td>
    <td rowspan="2" nowrap>
      <table border="0" cellspacing="1" cellpadding="0" bgcolor="#000000">
        <tr bgcolor="#FFFFFF" align="center" valign="middle">
          <td bgcolor="#FFFFFF" width="25" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(0);">
          </td>
          <td bgcolor="#000000" width="25" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(1);">
          </td>
          <td bgcolor="#666666" width="25" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(2);">
          </td>
          <td bgcolor="#999999" width="25" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(3);">
          </td>
          <td width="25" bgcolor="#CCCCCC" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(4);">
          </td>
          <td width="25" bgcolor="#FF0000" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(5);">
          </td>
          <td width="25" bgcolor="#0000FF" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(6);">
          </td>
          <td width="25" bgcolor="#009900" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(7);">
          </td>
        </tr>
        <tr>
          <td width="25" bgcolor="#000099" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(8);">
          </td>
          <td bgcolor="#99CCFF" width="25" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(9);">
          </td>
          <td bgcolor="#990099" width="25" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(10);">
          </td>
          <td width="25" bgcolor="#FF9900" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(11);">
          </td>
          <td width="25" bgcolor="#FFFFCC" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(12);">
          </td>
          <td width="25" bgcolor="#CCCCFF" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(13);">
          </td>
          <td width="25" bgcolor="#CCFFFF" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(14);">
          </td>
          <td width="25" bgcolor="#CCFFCC" height="25">
            <input type="radio" name="COLOUR" onClick="setColour(15);">
          </td>
        </tr>
      </table>
    </td>
  </tr>
  <tr bgcolor="#CCCCCC">
    <td width="77" nowrap>
      <table border="0" cellspacing="0" cellpadding="3">
        <tr>
          <td><b><font color="#000000" face="Verdana, Arial, Helvetica, sans-serif" size="2">B</font></b></td>
          <td><b><font color="#000000" face="Verdana, Arial, Helvetica, sans-serif" size="2">
            <input type="checkbox" name="BOLD" value="1" onClick="setBold();">
            </font></b></td>
          <td><font color="#000000" face="Verdana, Arial, Helvetica, sans-serif" size="2"><i>I</i></font>

```

```

</td>
<td><font color="#000000" face="Verdana, Arial, Helvetica, sans-serif"
size="2"><b>
      <input type="checkbox" name="ITALICS" value="1"
onClick="setItalics();">
      </b></font></td>
      <td><font color="#000000" face="Verdana, Arial, Helvetica, sans-serif"
size="2"><u>U</u></font>
      </td>
      <td><font color="#000000" face="Verdana, Arial, Helvetica, sans-serif"
size="2"><b>
      <input type="checkbox" name="UNDERLINE" value="1"
onClick="setUnderLine();">
      </b></font></td>
    </tr>
  </table>
</td>
<td nowrap>
  <table border="0" cellspacing="0" cellpadding="3">
    <tr>
      <td></td>
      <td>
        <input type="radio" name="ALIGN" value="left"
onClick="setAlign(0);">
      </td>
      <td></td>
      <td>
        <input type="radio" name="ALIGN" value="middle"
onClick="setAlign(1);">
      </td>
      <td></td>
      <td>
        <input type="radio" name="ALIGN" value="right"
onClick="setAlign(2);">
      </td>
    </tr>
  </table>
  <td nowrap> <font color="#FFFFFF" face="Verdana, Arial, Helvetica, sans-
serif" size="2"><font color="#CCCCCC" face="Verdana, Arial, Helvetica, sans-serif"
size="2">&nbsp;</font><font color="#000000">Top:</font></font>
    <select name="LINE_TOP" onFocus="updateColourChooser(2);"
onChange="setLineTop();">
      <option value="0" selected>-</option>
      <option value="1">1</option>
      <option value="2">2</option>
      <option value="3">4</option>
      <option value="4">6</option>
      <option value="5">10</option>
      <option value="6">15</option>
      <option value="7">-</option>
      <option value="8">S1</option>
      <option value="9">S2</option>
      <option value="10">S3</option>
      <option value="11">S4</option>
    </select>
    <font color="#000000" face="Verdana, Arial, Helvetica, sans-serif"
size="2">Bottom:</font>
    <select name="LINE_BOTTOM" onFocus="updateColourChooser(3);"
onChange="setLineBottom();">
      <option value="0" selected>-</option>
      <option value="1">1</option>
      <option value="2">2</option>
      <option value="3">4</option>
      <option value="4">6</option>
      <option value="5">10</option>
      <option value="6">15</option>
      <option value="7">-</option>
      <option value="8">S1</option>
      <option value="9">S2</option>

```

```

        <option value="10">S3</option>
        <option value="11">S4</option>
    </select>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<tr>
    <td bgcolor="#000000"></td>
</tr>
<tr>
    <td>&nbsp;</td>
</tr>
<tr>
<td>
    <table border="0" cellspacing="1" cellpadding="0" bgcolor="#000000">
        <tr>
            <td>
                <table width="700" border="0" cellspacing="1" cellpadding="2"
bgcolor="#999999">
                    <tr align="center">
                        <td colspan="2" bgcolor="#000099"><font color="#FFFFFF" face="Verdana,
Arial, Helvetica, sans-serif" size="2"><b>
FORMAT (Click on a line to format it)</b></font></td>
                    </tr>
                    <tr>
                        <td bgcolor="#ffffff" colspan="2">
                            <div id=previewHtml></div>
                        </td>
                    </tr>
                    <tr align="center">
                        <td width="350" bgcolor="#CCCCCC" align="right"><font face="Verdana,
Arial, Helvetica, sans-serif" size="2" color="#000000"><b>GROUP
1 - <font color="#0000FF">COUNTY.COUNTY_DESC</font>:</b></font><font
size="2" face="Verdana, Arial, Helvetica, sans-serif">
</font></td>
                        <td align="left" bgcolor="#CCCCCC"><font face="Verdana, Arial,
Helvetica, sans-serif" size="2" color="#000000">
                            <input type="radio" name="GROUP_0" onClick="setGroup(0,0);">

                            <input type="radio" name="GROUP_0" value="1"
onClick="setGroup(0,1);">
                            
                            <input type="radio" name="GROUP_0" value="1"
onClick="setGroup(0,2);">
                            </font></td>
                    </tr>
                    <tr align="center">
                        <td bgcolor="#CCCCCC" width="350" align="right"><font face="Verdana,
Arial, Helvetica, sans-serif" size="2" color="#000000"><b><font
color="#009900">DETAIL</font><font face="Verdana, Arial, Helvetica, sans-serif" size="2"
color="#000000"><b>:</b></font><font size="2" face="Verdana, Arial, Helvetica, sans-
serif">
</font></b></font></td>
                        <td bgcolor="#CCCCCC" align="left"><font face="Verdana, Arial,
Helvetica, sans-serif" size="2" color="#000000">
                            <input type="radio" name="GROUP_5" value="0"
onClick="setGroup(5,0);">
                            
                            <input type="radio" name="GROUP_5" value="1"
onClick="setGroup(5,1);">
                            
                            <input type="radio" name="GROUP_5" value="1"
onClick="setGroup(5,2);">
                            </font></td>
                    </tr>
                    <tr>
                        <td colspan="2" bgcolor="#CCCCCC" ><font color="#000000"
face="Verdana, Arial" size="2"><b>Page:</b>

```

```



```

Appendix C – Graphics

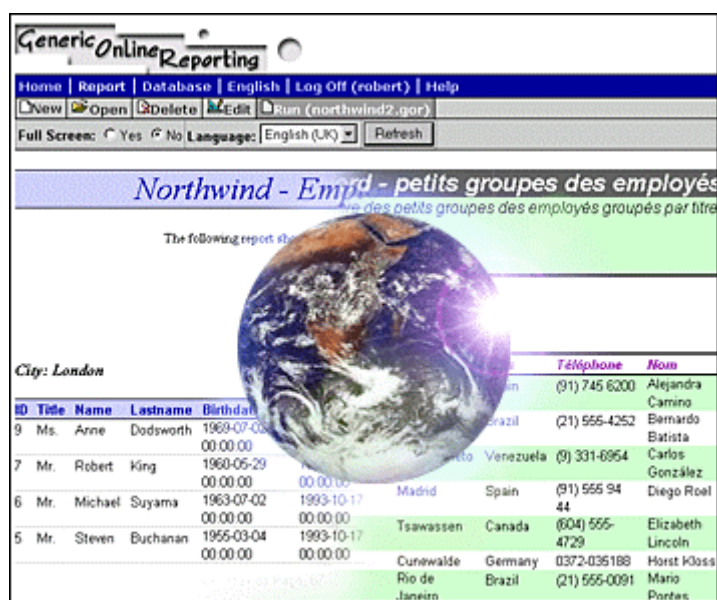
There are a total of 33 images used in the entire system. The use of images in any web system must be done in a conservative manner. The majority of these images (31) are tiny icons (about 1K in size). This ensures each page downloads in the quickest time possible.

Of the two other images, one is the title image, which is quite plain and appears in every page of the system. This image doesn't need to be reloaded as most browsers cache images. The image was created in Adobe Photoshop, which is an excellent program for creating and manipulating images.












title.gif (7k)

The other image is from the “Home” section of the system and it is the most complex. It was also created in Photoshop using two reports blended together with a picture of planet earth (taken from internet – <http://images.google.com>) on top. A single pixel black border was added with a small shadow to set it off.



home_image.gif (44k)

The following table show all of the icons in the system.

Icon	File Name	Description
	lp.gif	Special 1x1 transparent pixel used for spacing purposes.
	flag_britian.gif	Flag icons used to simplify the selection of an interface language. These were found at the mySQL database web site (http://www.mysql.com) in the file download section.
	flag_france.gif	
	flag_spain.gif	
	flag_germany.gif	
	flag_usa.gif	
	icon_align_left.gif	These are used in the format section for selecting the alignment of text. These icons are from MS Word.
	icon_align_middle.gif	
	icon_align_right.gif	
	icon_database_add.gif	These icons are associated with the database sections of the system. These icons are from MS Word.
	icon_database_delete.gif	
	icon_database_open.gif	
	icon_edit.gif	These icons are associated with the report - edit section of the system. These icons are mostly from MS Word and MS Access. Icons which were handcrafted are icon_edit_test.gif and icon_edit_format.gif.
	icon_edit_format.gif	
	icon_edit_groups.gif	
	icon_edit_labels.gif	
	icon_edit_query.gif	
	icon_edit_relations.gif	
	icon_edit_tables.gif	
	icon_edit_test.gif	
	icon_format_data.gif	These icons are used in the format section to show how a group is displayed in a report. These icons were all handcrafted.
	icon_format_ss.gif	
	icon_format_tb.gif	
	icon_new.gif	These are standard icons used in most windows applications. They are for creating / opening reports.
	icon_open.gif	
	icon_page_100.gif	These are used in the format section for selecting the page orientation of a report. These are from MS Word (icon_page_100.gif was handcrafted).
	icon_page_landscape.gif	
	icon_page_portrait.gif	
	icon_printer.gif	From MS Word
	icon_report_delete.gif	Handcrafted.
	icon_run.gif	Handcrafted.

Appendix D – Evaluation Guide

To evaluate this system please follow the following guide.

Section A – Create ODBC connection to the database northwind.mdb

1. Click on the “Data Sources (ODBC)” icon.
2. Click on System DSN – Add – Microsoft Access Driver – Finish.
3. Name the Data Source as “gor_northwind”. The location of the database is c:\databases\northwind.dbd.
4. For advanced users. An optional step is giving the database connection a user name and password.

Section B – Logon to the system

1. Try to logon onto the system using a false user name and password.
2. Logon onto the system using your username and password.
3. Read the Home section to give yourself an idea of what the system is intended to do.
4. NOTE: If you need help the Quick Help sections at the bottom of each page is a good starting point.

Section C – Create a new database connection file

1. Click on Database – Add.
2. Name the File Name as “gor_northwind”.
3. Name the Connection Name as “gor_northwind” (Data Source from Section A).
4. Fill in the other parts and click OK. If the message “Database connection was successful!” appears then the database connection file has been created.
5. Check the tables and columns in the database connection file by clicking on Database – Open.

Section D – Create a new report file

1. Click on Report – New.
2. Select “gor_northwind.dbd” from the list, name the report file and click OK.
3. Try and run the report (Report – Run) and you should see 3 errors.

Section E – Edit Report (Create query)

The first part of editing a report is to create a query. This involves using the first 3 sections of the Report – Edit section (Tables, Relations and Query). Emulate the following Access query using the system (Figure 1).

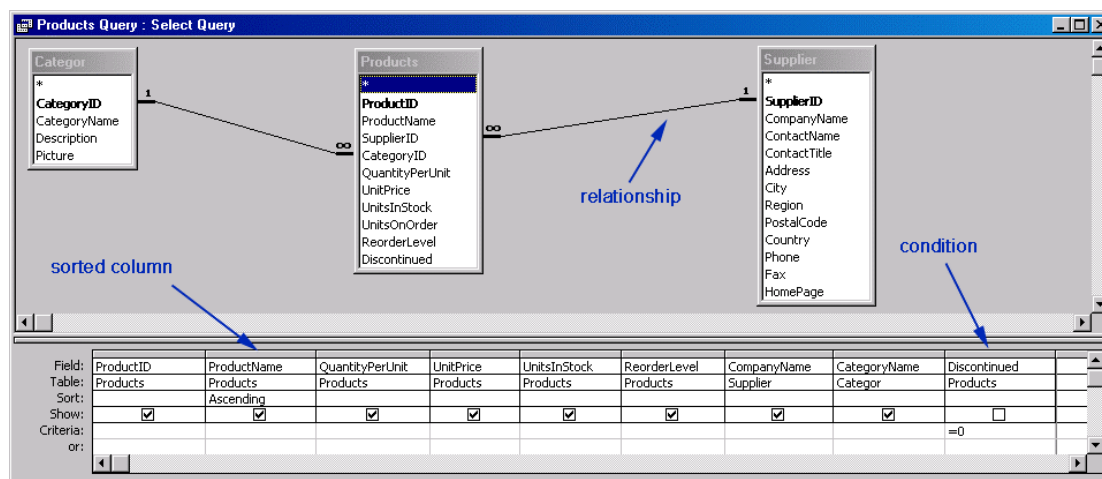


Figure 1

1. Click on Report – Edit – Tables and add the tables Products, Categories and Suppliers and give appropriate aliases.
2. Click on Report – Edit – Relations and create relationships (lines between tables in Figure 1) between the tables e.g. Products.SupplierID = Supplier.SupplierID).
3. Click on Report – Edit – Query and add all the columns and conditions in Figure 1.
4. To test a query click on Report – Test. If the query has been completed correctly then it should return 69 rows in total.
5. Once the query has been set up successfully, click on Report – Edit – Groups and create a group using the Category.CategoryName column.

Section F – Edit Report (Labels)

1. Click on Report – Edit – Labels and create a new set of English (UK) labels.
2. Set the title to “Northwind”, Subtitle to “The following is a list of all currently available products grouped by Category.” and Footer to “www.northwind.com”.
3. Give all the data fields to appropriate labels. e.g. give Products.ProductID the label “ID”.
5. For advanced users. If you know another language apart from English then create another set of labels in that language.

Section F – Edit Report (Format)

The report should be able to run now so click on Report – Run to see it. When you run it you should see that it is quite plain with the colours being simply black and white. The last and final step in creating a report is to format it with various colours, fonts, lines, etc. Click on Report – Edit – Format and try and create a report similar to the following diagram (Figure 2).

Northwind						
The following is a list of all currently available products.						
Category: Beverages						
ID	Name	Quantity	Price	Stock	Reorder Level	Company
1	Chai	10 boxes x 20 bags	18.0000	39	10	Exotic Liquids
2	Chang	24 - 12 oz bottles	19.0000	17	25	Exotic Liquids
39	Chartreuse verte	750 cc per bottle	18.0000	69	5	Aux joyeux ecclésiastiques
38	Côte de Blaye	12 - 75 cl bottles	263.5000	17	15	Aux joyeux ecclésiastiques
43	Ipoh Coffee	16 - 500 g tins	46.0000	17	25	Leka Trading
76	Lakkalikööri	500 ml	18.0000	57	20	Karkki Oy
67	Laughing Lumberjack Lager	24 - 12 oz bottles	14.0000	52	10	Bigfoot Breweries

Figure 2

Section G – Final Steps

Now that a report has been successfully created, complete the following to end this evaluation.

1. Try the report in full screen mode.
2. Run the report in different languages (if other languages have been specified).
3. Click on Report – Delete and select your report file and delete it off the server.
4. Click on Database – Delete and select “gor_northwind.dbd” and delete it off the server.
5. Change the language of the user interface (click on English) and do a quick browse through the system.

You have successfully completed this evaluation of the system. The final step is to complete the evaluation questionnaire. Thanks very much for taking part in this evaluation.

Appendix E – Completed Evaluation Forms

Eight people took part in the evaluation of the system. The following 16 pages are the completed evaluation forms.

Appendix F – Accompanying CD

The accompany CD with this report has five directories. Each of these directories contain the following:

1. \document – This directory contains two files. The “End Of Semester 2 Report.doc” file which is this report and the “Source Code.doc” file a list of all the source code of the system.
2. \generic_online_reporting – This directory contains all the classes, HTML, images and files need to run the system on a web server. The following is the readme.txt file which shows a user how to install the system:

```

-----
G E N E R I C       O N L I N E       R E P O R T I N G
-----

INSTALLATION GUIDE

Follow the following steps to install the system.

1.  Ensure that a webserver that is capable of running
    Java Servlets is installed and running properly.
    The Java JDK version must be at least 1.3.

2.  Copy the gor directory the root directory of the
    HTML pages. i.e. If using the Apache Web server and
    the html pages are stored on the "c:\Apache\htdocs\"
    directory place the gor directory under it to make:

    "c:\Apache\htdocs\gor\"

3.  Copy the CONTENTS of the servlets directory into the
    servlets directory on the web server. For example
    Apache JServe:

    "c:\Apache JServ 1.1.2\servlets\"

4.  Change the following String in the Site.java file
    (if necessary) to the location of the servlets
    directory on the web server and compile it.

    public static final String DIRECTORY_OF_SERVLETS =
        "c:/apps/Apache JServ 1.1.2/servlets/";

    Move the compiled site.class file into the servlets
    directory with the other classes.

    NOTE: All the source code to the system is stored on the

5.  The web server should need restarted.

```

6. Create an ODBC connection to the Northwind database which comes with Microsoft Access. Name the system DSN as "northwind". (OPTIONAL STEP)
7. To test the system copy the following into a browser on the server: (OPTIONAL STEP)

`http://localhost/servlet/Report?report=northwind.gor&label=0`

 If a report displays then the system is working.
8. Access to the system is now possible using the following web address ...

`http://localhost/servlet/LogonServlet`

 ... using the following username and password.

 Username: admin
 Password: admin
9. To create new usernames and passwords run the `UserNameAndPassword.class` file and copy the generated String into the `\servlets\files\system\users.dat` file.

 i.e. `java UserNameAndPassword`

 If any problems with the above steps then contact Robert Marks at the below email addresss.

Email to Robert Marks at
marksie531@yahoo.com (recommended) or
marks-r@ulst.ac.uk

3. `\html_format_prototype` – This directory has all the HTML files for the eight different prototypes discussed in Appendix A.

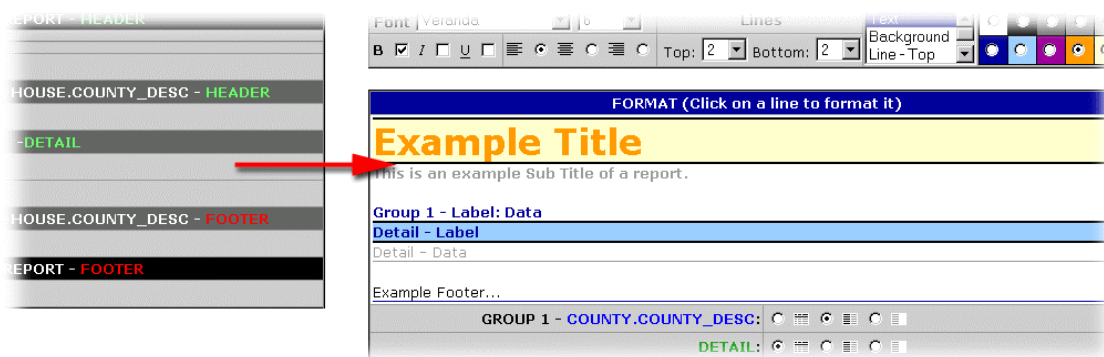


Figure 1

4. \java_doc – A very handy feature of the Java JDK is the ability to generate HTML documents from comments in Java source files such as the following:

```

/** Returns the country of the locale (e.g. "gb")
 * @param num      0 to number of interface locales - 1
 * @return         Country
 */

```

The generated HTML files use frames with a navigation bar on the left for all the classes. The frame on the right is the contents of that particular class. This means of browsing the system code is especially recommended for anyone who does not want to see page after page of code (Figure 2). To use open the index.html using an Internet Browser.

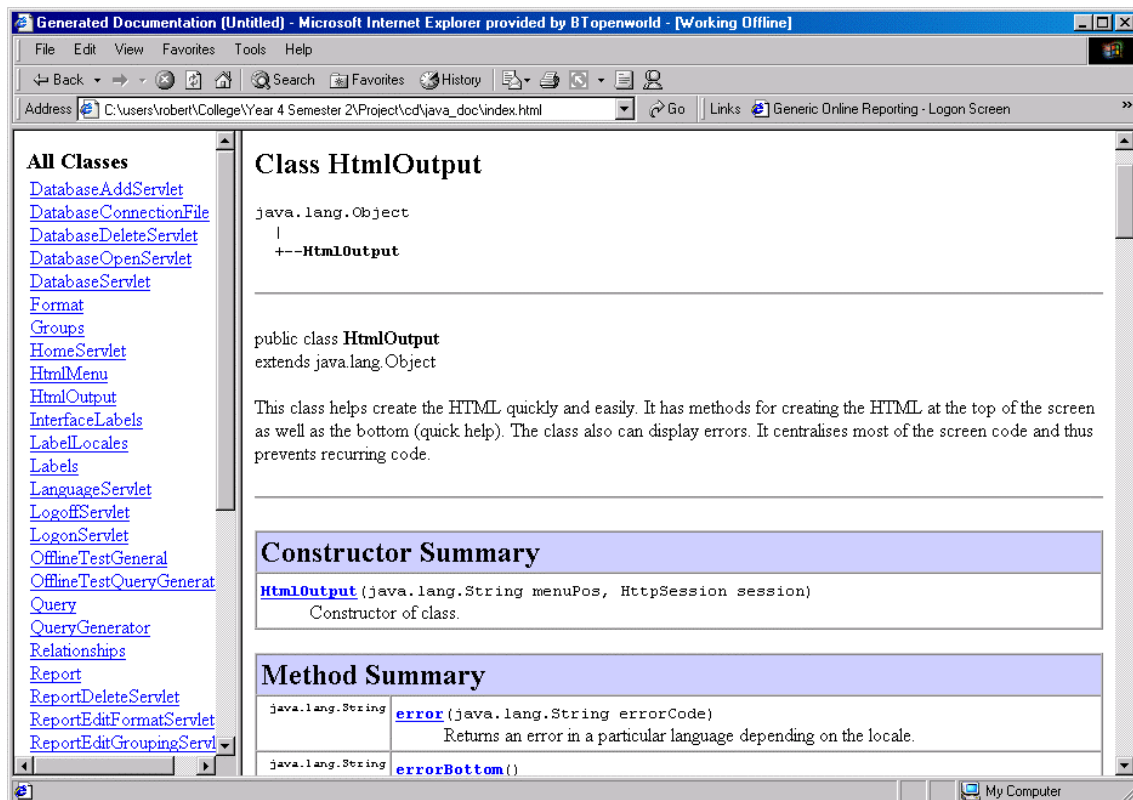


Figure 2

5. \java_source_code – The last directory on the CD holds the complete Java source code for all class used in the system. There is 41 in total.