Hyperstructure Image Organiser

Submitted September 2005, in partial fulfilment of the condition of the award of the degree M.Sc in Information Technology

Joanne Louise Carter

School of Computer Science and Information Technology
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: _____

Date: ____ / ____ / ____

Abstract

This dissertation researches the application of hyperstructure to multimedia organisation, and aims to discuss the development of an image organiser which facilitates the use of transclusion. It includes discussion of previous technologies such as semistructure and relational databases, along with the problems experienced with both. Furthermore, it alludes to a similarity between the ideas of ZigZag and the mathematical theory of sets. This culminates in a discussion of a new area of hyperstructure theory called Dimensional Informatics, which attempts to provide a mathematical underpinning to this field. There is also analysis of the newly developed Mantra server, including both an evaluation and suggested improvements.

Acknowledgements

I would like to thank my supervisor, James Goulding, for his suggestions and comments throughout my dissertation. In addition, I would like to thank the members of the Mantra development team, and the ZigZag research group for their contributions in discussions and their shared knowledge.

I would also like to acknowledge my mother for her continued support throughout my education, and for her endless patience and invaluable proofreading.

Table of Contents

Table of Figures

Chapter 1: Introduction

Focusing on ZigZag, this project aims to research and build a photo organising system that facilitates transclusion.

The current filing system is modelled on the original notion of files, folders and desktops, adapted from the visions of Bush [7] and Engelbart [7]. Indeed, even databases are based on a paper system. Whilst universally accepted, these systems contain some fundamental flaws including multiple copies of files, versioning and imperfect knowledge ('nulls' or unknown data items). Many people have developed alternative systems to try to overcome these problems, including Norman [17], [18], Raskin [20], Shneiderman [21] and Nelson [13], [14]. One such system currently under development, called ZigZag™, deals with these problems through the use of transclusion and binary decomposition.

Additionally, due to competition among software companies it has become increasingly difficult to transfer files and information between applications. This means that numerous pieces of information are predisposed to be duplicated, such as names and contact details. This has been termed the "*application prison*" [12]. Ideally, ZigZag aims to help break out of this prison. This can be achieved through the use of applitudes or "*zones of purpose*" [13]. As all ZigZag clients share the same data structure, this means that data can be shared easily between applitudes. Moreover, information does not need to exist more than once as data can be cloned through the use of transclusion. The idea is the design of a hyperstructure-based computing environment which can be structured according to individual needs, rather than having to structure information around the applications provided by the computer.

1.1. Motivation

The objective is to research the application of hyperstructure to multimedia organisation, in part utilising the new Mantra server [32]. This server has been developed by the Web Technologies Group [37] of the University of Nottingham to implement ZigZag structure. It has been designed so that it can be expanded for use with any Dimensional Informatics based system (see 2.2 for discussion of *Dimensional Informatics*). An image organiser has been selected as one of the first Mantra clients to be developed due to its possibilities for integration with other clients, such as an address book or a personal organiser. Particular attention should be paid to making the design accessible to the widest range of people, including selecting the colours with the best contrast sensitivity, and providing both buttons and keyboard shortcuts.

Chapter 2: Semistructure and Hyperstructure

2.1. Overview of Semistructured Data

A vast amount of data is available today, especially on the Internet. This information can be structured in a multitude of different ways, ranging from relational databases to semistructured data to raw data, such as bitmaps or plain text. There are many reasons why data may be semistructured. The structure of the data could be irregular due to missing pieces of information, or due to data heterogeneity. Structure may also only exist partially because some data is external, or parts of data themselves lack structure, such as images [22]. The field of semistructured data aimed to address concerns with the weaknesses perceived in the relational database concept, as well as the realistic view that virtually no real-world data fits into a regular structure. The structure of data depends on a point of view. This view of the world may change, or it may differ depending on the context in which it is being viewed.

With semistructured data, the information that is normally associated with a database schema is contained within the data; this is termed *"self-describing"* [3]. This is convenient for conceptualising the World Wide Web, where data is varied and irregular and is in a constant state of flux. The Web can be viewed as a type of database, but it is not constrained by any schema. The users of the Web could take advantage of database support, particularly with respect to querying [10].

Semistructure is a flexible data format for data exchange between disparate databases, as even structured data can be considered a subset of semistructure. It represents the *"convergence of a number of lines of thinking and ways to represent and query data that does not fit with conventional data models"* [3]. The discipline was motivated by the need to bring new forms of

data into the ambit of conventional database technology, facilitating data exchange and transformation. Clearly, a database system that could easily accommodate irregular data and changes in structure would *"greatly facilitate the rapid integration of heterogeneous databases"* [10].

It was believed that semistructured data can *"most naturally be modelled as a collection of objects"* and each object may have any number of (possibly repeated) attributes, whose values can be other objects or atomic data [22]. There were a number of proposed models for semistructured data, including labelled graphs with nodes and edges (although with no provision for transclusion), XML structures, etc. Implementing a model for semistructured data required the rethinking of all aspects of information management systems, including *"storage management, indexing, query processing and optimisation, and user interfaces"* [10].

Representing data as a graph facilitates the discovery of new data to load into the structure, assists the integration of heterogeneous data and enables the structure to be easy to query without knowing the data type of the information. However, there exists a vulnerability to losing type information and optimization can be affected, causing it to be harder or nearly impossible due to the way the graph is structured [10]. The OEM (Object Exchange Model) data model can be represented as a labelled graph, with all the schema information contained in labels. One derivation of this model, LORE (Lightweight Object Repository) contains two types of objects: atomic units and collections of label-object pairs. Its query language, LOREL, employs a powerful form of navigation that utilises path expressions to traverse the graph inside the database engine, allowing users to retrieve and update data easily where there is no fixed, known structure [10].

LORE attempts to take advantage of structure where it exists, but also handles irregular data as gracefully as possible. No restrictions are imposed on the size or structure of atomic or complex data objects to accommodate semistructured data at the physical level. Additionally, the layout of objects on the disk has been customized to facilitate browsing and the processing of path expressions. LORE includes *DataGuides* to replace schemas, these provide a snapshot of the structure of the data and guide query optimisation. Furthermore, because one of the motivations

for using a semistructured data management system is to integrate data effortlessly from heterogeneous information sources (including the World Wide Web); LORE includes an external data manager to bring in data dynamically from external sources as needed during query execution [10].

Although the accepted representational format of semistructured data is an XML tree structure, this has proven to be unwieldy due to the lack of mathematical basis in this field. It is also virtually impossible to navigate and consequently hard to query. Managing semistructured data with tree structures can cause imbalance, and it can be difficult to compare data due to the complexity of the structure [22]. As a result of these problems, very little research has been done in this field in recent years and work on its original systems, such as LORE [10], has almost been abandoned.

2.2. Overview of ZigZag

Consider the basics of computer science. Data is represented by files with specific naming conventions. These files are organised in a hierarchical directory structure in a window-based operating system. Applications are used to access and modify files, and data can be transferred in a fashion between applications using the clipboard. Why is this? The computer is a mechanism by which we should be able to enact any world we choose. Limitless possibilities, yet we are stuck with a system that was designed for the optimisation of secretarial work. It should be about how we can get the effect we want with the resources that we have, not about recreating a paper based system [15].

Although it was born from an idea of how the world could be enacted, ZigZag can be analysed in terms of relational databases and set theory, bringing it under the ambit of *Dimensional Informatics* [24]. This is a new area of computing that aims to address the problems with relational databases and serve as a highly generalised information model. It also encompasses

decomposed relational databases [5], *Fenfire* [25] and possibly the *Semantic Web* [34]. ZigZag can be considered to be a single structure that represents a conglomeration of several existing structures, such as relational databases, lists, spreadsheets, etc.

The primary problem with databases is 'null' values. Unknown quantities are incompatible with computers, because of their inability to make a decision as to how the 'null' should be processed; does the information not exist, or is it merely not available at the moment? Relational databases are based on set theory, whose essence is the relation. At the heart of the relational model is Codd's *Information Principle* [4] that states that all information must be encoded as data values in relations. Relations are facts about the world, either true or false, and having an unknown value violates this theory on which relational databases are founded.

The following example illustrates a simple group of person relations, or records. This example highlights the problems with 'nulls' perfectly, as there are people who do not have middle names. Writing each of these relations mathematically, one can begin to see the problem; for example, {2, Neil, ?, Carter} is wrong [Figure 2.1], one cannot claim a statement is true if part of it is unknown. Several alternatives have been suggested, including having type 'nulls', but this will only lead further into a logical quandary. The only solution is to use normalisation to eliminate the 'null' values, and split the database into smaller tables. From there the structure can be normalised as far as possible, until full binary decomposition is achieved [Figure 2.2].

ID	First name	Middle name	Surname
1	Joanne	Louise	Carter
2	Neil	? (NULL)	Carter
3	Anne	Elizabeth	Clark

Figure 2.1: A table representing a simple person relation.

ID	First name
1	Joanne
2	Neil
3	Anne

ID	Middle name
1	Louise
3	Elizabeth

ID	Surname
1	Carter
2	Carter
3	Clark

Figure 2.2: Breakdown of the original table into a set of binary relations.

Only the values that are known have been entered.

There are now three relations with no unknown values, which could be queried using the JOIN operation. Everything is now consistent with mathematical theory. However, if queries were to be attempted with JOINs then a large quantity of processing power would be needed, especially with a large database, and the computer would grind to a halt. This is where ZigZag has the advantage as it makes use of binary relations; field names equating to ZigZag dimensions, which are decomposed and pre-joined in an orthogonal representation. Pivot cells, often represented by a primary or surrogate key, act as a central point to the entity. These cells are, by definition, multidimensional in nature and therefore visualisation is also far neater when represented via a graph [Figure 2.3].

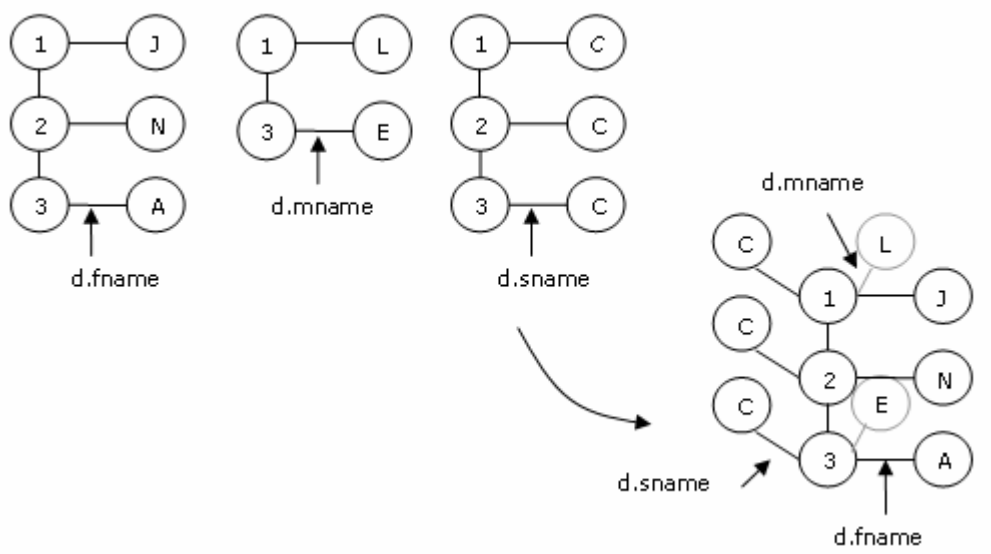


Figure 2.3: An illustration of how each binary relation can be represented as a graph, with nodes and edges. These can naturally be pre-joined to show three dimensions (or more).

The underlying assumption is that information can only be represented by binary relations in the world, and this denotes one of the primary axioms upon which Dimensional Informatics is based. This assumption makes the structure infinitely more flexible as data is already split up and there exist no assumptions as to what information is known. The flexibility of the structure also makes it more amenable to change. If it were to become more complicated then one cannot guarantee possessing more information. Being that one can never know that everything is known, this tries to represent the true nature of the world. The nature of the world is particularly relevant to the normalisation of relational databases, because normalisation is subjective. As illustrated, full binary decomposition is the result if a relational database is normalised entirely. One usually has to judge how far to normalise, especially as the world can change so that information that was once normalised may not be in the future.

The ZigZag structure can be represented as a tractable graph due to the rule that there may only be one posward and one negward connection to a cell (zzstructure) in any dimension [15]. This choice of 1-in-1-out makes it easy to navigate as one can always retrace the steps that have been made [9]. This is reminiscent of the information trails first proposed by Bush [7], the routes by which one finds information are as important as the information itself. Cells can contain any one type of data (text, images, sound, etc).

Dimensions are sets of information that can be related semantically or can be completely abstract. The standard syntax for a dimension is *d.something*, and three general dimensions are provided as standard, *d.1*, *d.2* and *d.3*. Together relations build up a dimension; for example, {Joanne, Neil, Anne} make up the name dimension (*d.name*), as they are related to each other. One can also have transitive relations, such as *d.parent*, where information can be inferred from the ordering of the cells. One can tune in and out of information with dimensions, like a radio. The system is built out of itself, dimensions themselves are cells; this has often been termed as "*turtles all the way down*" [36].

The same cell can exist in different contexts through the use of transclusion [14]. The resultant clones have their own dimension, *d.clone*, where the other contexts in which the cell exists are visible. If the information in one clone is changed, then the others will change to reflect it, as they are conceptually the same cell. Clones inherit from the rank head if attributes are not specified [16]. The deletion of the head of a rank of clones causes the next clone in the rank to assume the properties of the head; this ensures that there are no disconnected cells. This function will be particularly useful when dealing with the image organiser because, instead of there being multiple copies of the same image in different folders, images can be cloned. This could save on storage space and eliminate the problem of different versions existing.

The understanding of the Zigzag structure does depend to some extent on spatial awareness. Basically, ZigZag is a paradoxical space that leverages on familiar concepts such as the row, column, corner and wheel [16]. All views are legitimate and the structure can differ; its appearance depends on how it is presented to the user, or according to their preferences as to how it is to be displayed.

2.3. Overview of Mantra – Set Processing Language

It is important that a query language for such a structure be based upon set theory rather than SQL, to ensure that queries are performed quickly and efficiently. Mantra [32] is a general processing language created, at the University of Nottingham [37], for use with Dimensional Informatics structures, and as such it is not limited to ZigZag; in this context however, as ZigZag can be viewed with regards to set theory, it will be used with zzstructures. Furthermore, since the language is based on this mathematical theory, cells can be processed quickly and the structure can be traversed with ease. It also has roots in hypertext, due to its specificity to binary relations, and as such, it may be viewed as a hypertext or database query language, which may serve as a unifying influence between the two fields in the future. One of the original ZigZag implementations, gZZ [8], did not separate the structure of ZigZag from the presentation. This was judged to be a cumbersome approach to the issue, and the idea of a server with many clients was brought into fruition.

Zzstructures are uniquely identified through the use of a ZRI (ZigZag Resource Identifier), a cell address, which has since been re-named to simply NRI to maintain neutrality. Each unique identifier is made up of the server ID, the slice ID on the server and the cell ID (which is unique within the slice). Mantra is an SPL (Set Processing Language) used by clients to communicate with the Mantra server. The processing language takes in, and returns, a ZigZag structure, which consists of a set (a list) of NRIs. Query catenas consist of many statements connected together by the ‘|’ (pipe) symbol [Figure 2.4]. The resulting output set from each statement is passed on to the next statement as its input set; continuing until the final output set is reached.

```
> 0.0.0 | fetch all along "d.main" | create + along "d.penguins" | change
content = "hello"
```

Figure 2.4: An example mantra query.

Start with the first cell and get all the cells that exist in the main dimension. Then create new cells in the posward direction of each of those cells, along the dimension *d.penguins*. Change the content of all those cells to "hello".

The client sends the query as XML, using a specified set of Mantra tags and the server responds in the same way [Figure 2.5].

```
<mantra>
      <query ID="0">0.0.0 | fetch all along "d.main" | create + along
"d.penguins" | change content = "hello"
      </query>
</mantra >

<mantra version="0.1">
      <response ID="0" total_usecs="491" processing_usecs="373">
            <cell zri="0.0.10">d.main</cell>
            …
      </response>
</mantra >
```

Figure 2.5: A message from a ZigZag client to the Mantra server and its response.

This standardised output of queries allows it to be produced and parsed (or interpreted) by all programming languages. This use of XML contrasts with its original use with semistructured information, in models like LORE [10], as it is underpinned by mathematical theory; and as such is optimized for this particular purpose.

Chapter 3: ZigZag Structure and the Mantra Server

3.1. ZigZag Structure

Due to the novelty of this area of research there have been no formal specifications produced about how dimensions should be set up. This can cause a great deal of confusion over what a dimension is, and can overcomplicate ZigZag structure. A dimension can be considered a context of information, or a link between different units of information; a way of connecting related cells.

With earlier forms of ZigZag like gZZ [8], there was a dimension called *d.main*. However, it was very difficult to conceptualise what this dimension actually meant, and which cells it should contain. This has been discarded with the Mantra server in favour of a *system* cell. The *system* cell contains (i.e: is connected to along *d.contains*) the slices, types and dimensions that are needed in the structure [Figure 3.1]; and in addition, the *home* cell, contains the structure and the cells which relate to the different clients. These cells are connected along *d.set*. The cell which represents zzImage will be contained in the *home* cell; this will then contain the images, tags etc. The system cells will be hidden from the user, who will be restricted to the zzImage cells; except when information from other clients is needed (see discussion of *Zones of Purpose* in 3.2.11).

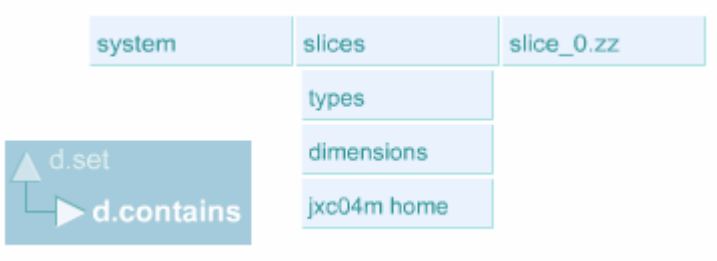


Figure 3.1: The ZigZag system cells

These two dimensions, *d.contains* and *d.set* are the main dimensions in the structure [Figure 3.2], acting to create a hierarchical structure, which can be thought of in terms of folders. The set replaces the idea of the wheel of cells, which has been used previously with ZigZag structures, as it represents more accurately the disordered nature of the cells. The terminology 'set' is more in sync with ZigZag's affinity with Set Theory (see previous discussion of *Dimensional Informatics* in 2.2).

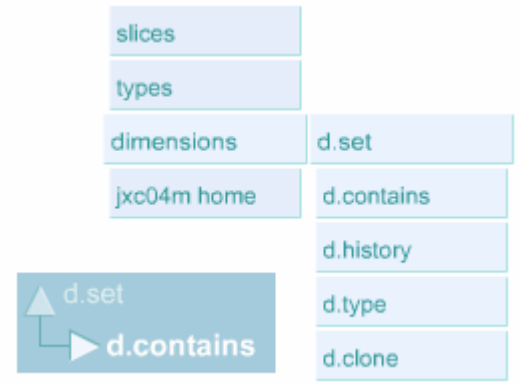


Figure 3.2: The current default dimensions (*d.history* is inactive)

The understanding of how the ZigZag structure works is hinged on one's comprehension of dimensions. Any misconceptions at that stage can cause complications further on, and this is why it is important to grasp them initially. Once the concept of the dimension is understood, then the idea of ZigZag can come together.

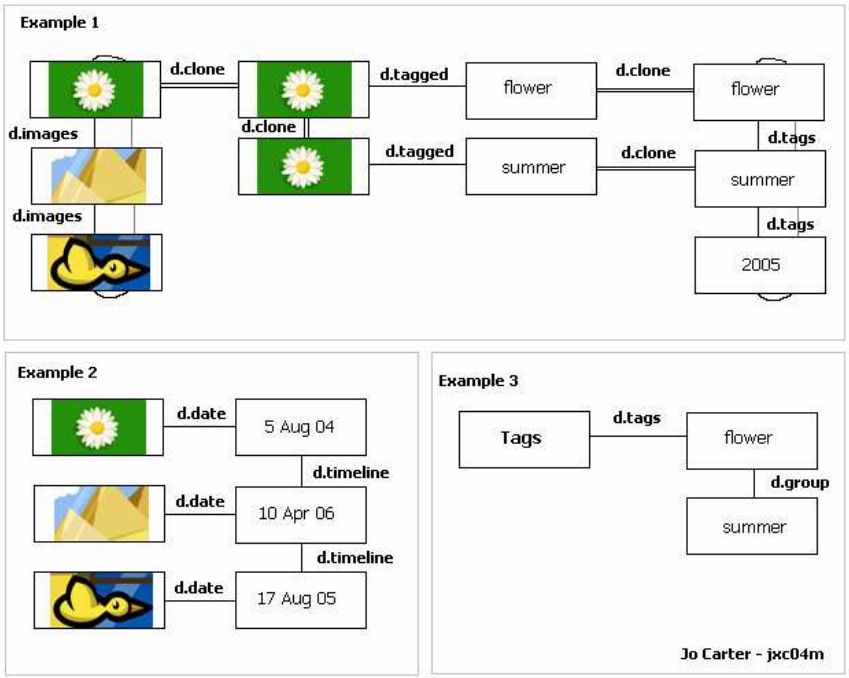


Figure 3.3: Example structure for zzImage.

As illustrated [Figure 3.3], it is very easy to become confused about how to organise the cells. This can result in an unwieldy ZigZag structure, excessive cloning and poor use of resources. This also introduces multiple redundant dimensions, which are poorly optimised for use in any situation. A consistent approach is needed, which can be used by multiple clients. This can also provide a basis for any ZigZag structure. The dimensions d.contains and d.set are part of this solution. The proposed structure for zzImage [Figure 3.4] makes use of a new dimension, d.ordered, which is discussed later (see discussion of *d.ordered* in 3.2.4).

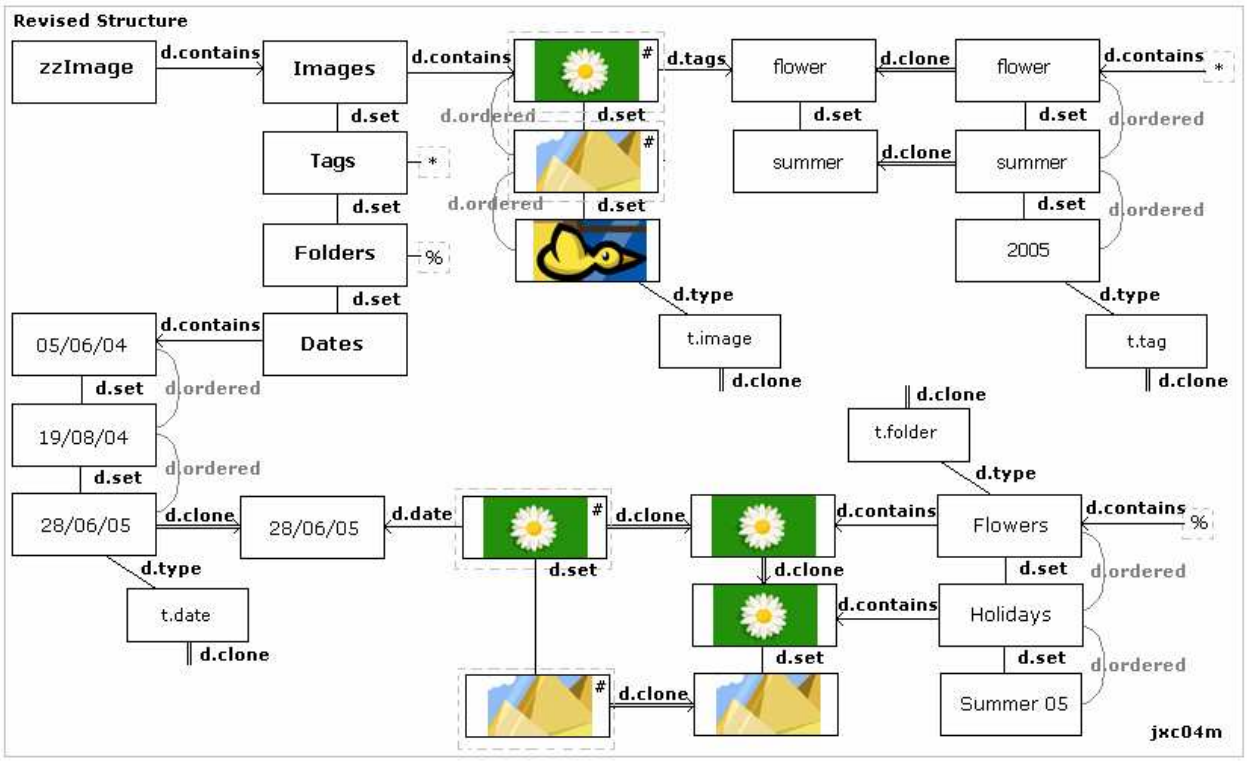


Figure 3.4: Revised structure for zzImage, incorporating *d.set*, *d.contains* and *d.ordered*

The arrows indicate the 'posward' direction. The dotted grey squares and symbols indicate where cells have been duplicated temporarily to facilitate the demonstration of the structure

This revised structure aims to make the most efficient use of transclusion. Clones must be used because ZigZag cells can only have one 'posward', and one 'negward' connection along any dimension; i.e: cells can only belong to one set at a time, if one wants to connect a cell to a different set then cloning must be considered. This is because connecting a cell to another cell along *d.set* would sever any previous connections the cell had along a different *d.set*.

Once the structure has been mapped out, it can be created on the Mantra server [Figure 3.5].

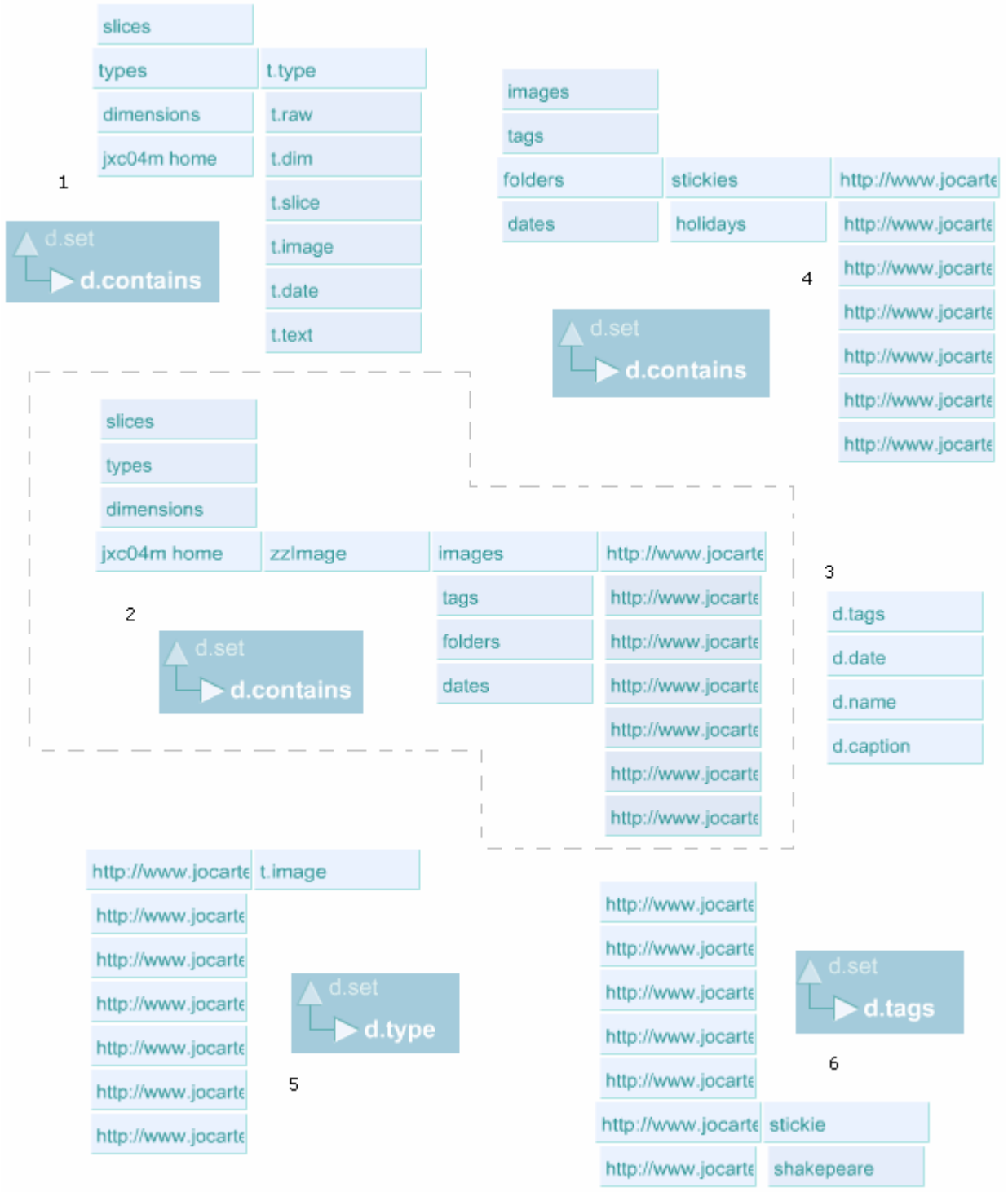


Figure 3.5: The structure from Figure 3.4 represented via the MantraF client

1: The cell types; 2: zzImage home cell and images; 3: zzImage's extra dimensions; 4: Folders with cloned images; 5: Image type; and 6: Image tags. Not represented here are examples of *d.name* and *d.caption*.

3.2. The Mantra Server

3.2.1. Introduction

"*The Mantra Set Processing Language is designed for the client-server communication of databases that use Dimensional Informatics (DI) rather than the relational model of information structure (such as ZigZag).*" Currently the only implementation is the MantraD server, which is under development in the University of Nottingham School of Computer Science and Information Technology [1].

Owing to the fact that the Mantra server is still in the initial stages (version 1.0), no more than the basic commands have been implemented so far. These work well to illustrate the basic principles of Mantra; however, there are some shortcomings and problems, which are discussed in later sections. Additionally, there are several supplementary ideas and commands that should be taken into consideration.

3.2.2. Current Commands

Only the basic commands have been implemented so far. These are illustrated in Figure 3.6.

```
save slice <number>
reset system
<cell_set> | fetch [all/this to prev/next to tail/head/-1 to -3/etc] along
"d.something"
<cell_set> | create [+/-] along "d.something"
<cell_set> | change content = <string>
<cell_set> | delete
<cell_set> | insert [+/-] <target_cell> along "d.something"
<cell_set> | sever [+/] along "d.something"
gather [dims/types]
resolve [all/content/mode/created]
```

Figure 3.6: The current Mantra query commands

3.2.3. Cloning

The current process of cloning in Mantra is rather cumbersome, involving the creation of a new cell along the appropriate dimension, which is then connected to the original cell along d.clone [Figure 3.7].

```
> 0.0.34 | create + along "d.set" | insert + 0.0.6 along "d.clone" |
resolve
```

Figure 3.7: Example of current method of cloning using Mantra

This example creates a new cell after cell 34 along *d.set*; this empty cell is then inserted after cell 6 along *d.clone*. The new cell is now a clone of cell 6

Mantra correctly interprets these cells as clones, which means that if the content of one cell is changed the other cells will reflect that change. Additionally, `resolve mode` correctly returns 'clone' rather than normal. It seems that any cell that is inserted along d.clone has its mode changed to 'clone'.

This process could be improved greatly by adding a simple `clone` command [Figure 3.8], which would provide an easy method of creating clones and reduce the number of potential errors.

```
> 0.0.34 | clone 0.0.6 + along "d.set" | resolve all
```

Figure 3.8: Example of the proposed 'clone' command

Clones should retain their type, i.e: take the type from the head cell of the rank; moreover, one should be able to follow the other dimensional connections, if the option to do so is turned on. However, currently with Mantra no inheritance for clones has yet been instigated.

3.2.4. Ordering Cells

Although the new dimensions *d.set* and *d.contains* provide a good framework for ZigZag, *d.set* is unordered. Cells are added without regard to whether they should hold a particular position along the dimension, with the premise that all the cells are connected to each other, and to the main cell along *d.contains*. If one needed to have cells in a specific order, then these would need to be fetched, ordered manually and then added to another dimension in order; additionally, this would have to be repeated every time a new cell was added.

Mantra is in need of another default dimension called *d.ordered* along which cells can be ordered alphanumerically; this would also involve the development of an additional command, `order`, to turn a complicated client-side process into a simple Mantra command [Figure 3.9]. Cells could then be connected in any order along one dimension, *d.set*; and then connected in alphanumeric order along another dimension, *d.ordered*. This could be extended to `order +` or `order -` depending on whether ascending or descending ordering was required.

```
> 0.0.34 | fetch all along "d.set" | order along "d.ordered" | resolve
```

Figure 3.9: Example of the proposed 'order' command

Naturally, when one begins talking about ordering, there are several things that must be clarified. Firstly, how does one determine order? Is it merely alphanumerical, numbers then letters? If so, where does punctuation come in? What about capital and lowercase letters? Alternatively, is there a need to consider lexical and linguistic implications? For instance, in Lithuanian, the letter 'y' comes between the letters 'i' and 'k'; and in Swedish the letters 'v' and 'w' are the same thing. There are rules and libraries for Unicode string sorting, but that only makes sense if the language of a particular cell is known [2]. This should not be a problem; however, there does seem to be a need for a mechanism to determine the language of any cell (see discussion of *Types* in 3.2.5).

A language's writing system will determine what influences the sort order of the language. For example, a sort order for Russian would be based on Cyrillic letters and possibly diacritics, but a sort order for Japanese might be based on the number of strokes it takes to draw a character. Linguistic sort orders are different from the Unicode code point order. A sorting element (such as a character) can be the combination of more than one Unicode code point. For example,

U+0B95 + U+0BCD + U+0BB7 = Tamil Ksha, which is a single sorting element in Tamil [2]. Additionally, languages that use the same script often have different linguistic sort orders. With regards to linguistic comparisons it could become quite difficult and would possibly be too complicated for this command; and thus if needed should be considered client-side.

3.2.5. Typing

Currently, if a type is assigned to a cell then the appropriate type's cell has to be cloned and connected to the new cell along *d.type* [Figure 3.10]. This is a long-winded process, which needs to be simplified. This is especially pertinent because the current method does not really give a cell a type, so type-specific operations are made more complicated.

```
> 0.0.40 | create + along "d.set" | change content = "hello.png" | create
+ along "d.type" | insert + 0.0.38 along "d.clone"
```

Figure 3.10: Example of the current method of typing, the creation of a *t.image* cell

There are a multitude of ways that this could be accomplished; however, to be consistent with the current commands (such as `change content = ""`), something like `change type = "t.image"` would be appropriate; or possibly as an extension of `create`; for example, `create + along "d.set" type = "t.image"`. When this command was executed, the server would need to check whether the specific type existed and either, error if it did not or default to the 'raw' type.

Types could be created through a `create type` command, and if a type does not exist then an error would be returned. Furthermore, the `create` command could also be extended to `create dim`, and perhaps even `create slice`. This would help to ensure that cells are created with ease, and that excessive cloning is minimised; furthermore, this would limit the number of cells that are created that should not be normal cells (i.e: dimension cells, type cells, etc.). The modes of a cell could possibly be expanded here, to give the choice of normal, clone, dim, type or slice.

The addition of this command would necessitate the expansion of the `resolve` command to include cell types (`resolve type`). The type could then be retrieved as an XML attribute [Figure 3.11].

```
<cell nri="0.0.40" type="t.image"/>
```

Figure 3.11: Example of XML response from Mantra server after 'resolve type'

It would be preferable to return the type as text rather then a cell identifier, owing to the fact that the system cells may change so that a table of reference could not be constructed easily. In addition, it is more useful to have the type returned as text because the text is more likely to be desired by the client. If the cell identifier of the type were needed, then it would be easier to retrieve it as required. Alternatively, one may be able to retrieve the cell identifier automatically with a command like `t.image | resolve`, thus keeping it server-side and providing another Mantra standard. However, regarding that command, it does seem to make more sense to be consistent with current paradigms, returning a cell identifier and using `0.0.9 | resolve` to get the type's name instead.

As was mentioned previously, types could play a large role in ordering cells (see discussion of *Ordering* in 3.2.4). The *t.text* type could be sub-typed to include different languages (i.e: *t.fr_text*, *t.gb_text*), which would effect how the content of the cell was dealt with by the `order` command; and also by any other commands that may need to compare cell content. Types are also very important with regards to cell content (see discussion on *Cell Content* in 3.2.6), as the type of a cell can affect how Mantra deals with it, and how it is displayed in a client. For example, if a cell was used to store a music file, then if the file were dealt with as text then it would be unreadable to humans; however, add a *t.music* type, and the program knows that it is a music file (this could also be sub-typed to deal with different music formats, such as *t.mp3*, *t.wma*, etc).

3.2.6. Cell Content

In its present format the Mantra server only allows cells to contain text data. This is obviously limiting due to the vast amount of data that could be stored in a cell; from images and files, to whole films; acting in a similar manner to blobs in databases. For example; the zzImage client is limited to a cell containing a physical address for the file (which was in turn limited previously by the inability of cells to contain more than basic punctuation, so that characters such as ':' and '/' would cause an error; this has since been resolved).

With the proper introduction of typing and the expansion of possibility for cell content, then a cell could contain an actual image file. This would obviously eliminate the possibility of a file not being found, due to it being moved or deleted, as the files would be stored on the server itself; furthermore, it would assist the Mantra server in becoming a self-contained system.

3.2.7. Type Specific Commands

An expanded implementation of types, and cell content, opens up the possibility of type specific commands. This is one thing which could be very useful depending on the type of client. For example, the image organiser could use a `create image` command, which would add a cell to 'Images' which is of type *t.image*, and could contain an actual image (or the data that makes up an image). These sorts of type specific commands could simplify and reduce the length of queries.

One other possibility is having commands contained in cells. This could be used for commands that are typed frequently, such as `fetch all` along specific dimensions. Envisaged are two possible syntaxes, either `execute 0.0.35` or `0.0.194 | execute 0.0.35`. The first is proposed for a command where one does not need a starting cell or where the starting cell is always the same, and therefore can be included in the command; i.e: `gather dims`, or perhaps `create image`. The second would act on the starting cell or cells provided before the `execute` command. Once `execute` was called the Mantra server would be able to retrieve the Mantra query syntax from the cell specified and execute it. Commands would have to be created in a different way to an ordinary cell to ensure that they were of correct syntax; perhaps through `create command`.

3.2.8. Filtering

At the present time any filtering of cells must be conducted client side; this includes searching for cells with specific content. A server-side `filter` command is needed to reduce the number of elements in a set by filtering them through a conditional statement (i.e: = =, !=, <, >, etc). This filtering may potentially be extended from covering purely cell content, to properties and attributes [Figure 3.12].

```
> 0.0.34 | fetch all along "d.set" | filter content < 10 | filter d.tags =
"stickies"
```

Figure 3.12: Example of the 'filter' command

However, the same considerations that applied to ordering must be applied also to these conditions, with relation to linguistic and lexical comparisons (see discussion on *Ordering* in 3.2.4). Additionally, in order to filter properly along dimensions such as *d.tags* (including *d.set*), then the ambit of dimensions need to be extended (see discussion of *Dimensions* in 3.2.9).

3.2.9. Dimensions

Dimensions are ways in which connections between different pieces of information can be represented. One can use dimensions to interpret these relations; however, what may be evident about a connection to the human eye, is not to a computer. For example, in *d.ordered* it can be said that one item of data falls before (or after) another item of data; from that one can infer that an item of data three cells up also falls before that item of data [Figure 3.13]. A computer cannot deduce this about a dimension without further information about it; the computer is limited to what it knows about the 'posward' and 'negward' connections.

{1, 2, 3, 4}

(1, 2) (2, 3) (3, 4)

(1, 3) (2, 4) (1, 4)

Figure 3.13: An example of a transitive set (ordered)

We can say 1 is less than 2 etc; and can infer 1 is less than 3, etc, because it is a transitive set. If it was also reflexive (less than or equal to) then we can say 1 is less than or equal to 1, etc.

There are many of these types of relations, taken from Set Theory [35], which could aid a computer in interpreting dimensions. These would have to be represented as attributes of a dimension [Figure 3.14], and the server could refer to a dimension's attributes to see how cells should be interpreted within a relation. Adding attributes to a dimension also allows concepts such as multiple names, limits to content, etc.

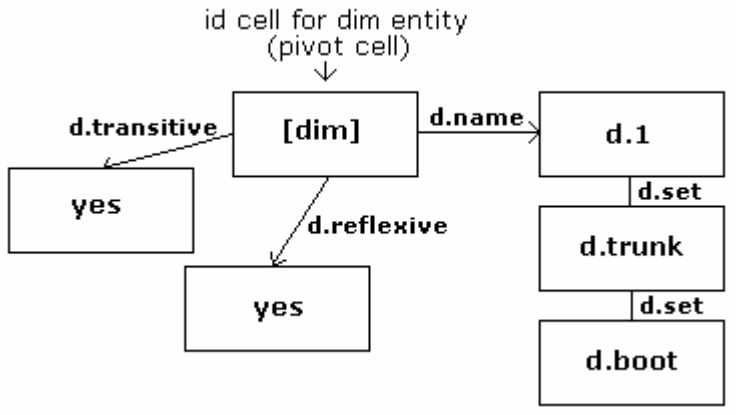


Figure 3.14: A diagram of a dim designator showing its relevant properties

Furthermore, if dimensional attributes were implemented then it could have welcome impact on the relationship between *d.contains* and *d.set*. One is able to infer that, because cell 2 is connected to cell 1 along *d.contains* and cell 3 along *d.set*, cell 3 is also connected to cell 1 along *d.contains*. However, with the current set-up a computer cannot deduce this, so a query must combine several commands to get all the cells contained in a set [Figure 3.15]. There could be just one command like `fetch all +1 along "d.contains"`, or perhaps `fetch all along "d.contains", "d.set"`.

```
> 0.0.33 | fetch +1 along "d.contains" | fetch all along "d.set" | resolve
```

Figure 3.15: The current Mantra query required to fetch all the cells in a set

If 0.0.33 is the cell for 'Images', then this command fetches the first image along *d.contains* and then all the other images from there along *d.set*.

This would also be extremely useful to use to preserve the head cell of a set of cells. Currently with Mantra if a cell is deleted from the centre of a set the set 'heals' itself and dimensional links are preserved. However, if the head cell of a set, the one connected along *d.contains*, is deleted the rest of the set is orphaned; this is contrary to the theory of ZigZag [16]. Using dimensional attributes it may be possible to preserve the head cell of a rank by passing the *d.contains* link to the next cell down. Even if this was not possible, it should be implemented in some way server-side.

One problem that has been experienced with dimensions, which relates to a previous discussion on cell content (see 3.2.6), is that although dimension cells can be created with any character, they can then only be referred to by their NRI if they do [Figure 3.16]. The name of a dimension should only be able to contain letters [Figure 3.17] and the underscore (_) character, in addition

to the '*d.*'; however, the current version of Mantra does not recognise dimension names that contain an underscore [Figure 3.18], and this can cause problems when NRIs are not standard to a dimension.

```
0.0.6 -> d.%^&*
0.0.56 | fetch all along 0.0.6
```

Figure 3.16: Dimensions created with non-standard characters can only be referred to by their cell NRI

```
0.0.7 -> d.contains
0.0.56 | fetch all along 0.0.7 OR 0.0.56 | fetch all along "d.contains"
```

Figure 3.17: Dimensions created with alphabetical characters can be referred to by name or NRI

```
0.0.8 -> d.author_name
0.0.56 | fetch all along 0.0.8 OR 0.0.56 | fetch all along "d.author_name"
```

Figure 3.18: Dimensions created with alphabetical characters and the underscore should be able to be referred to by both name and NRI

3.2.10. History

One of the previous forms of ZigZag, gZZ [8], had a default dimension called *d.history*. This dimension contained every cell created in the order they were created, which meant that if a cell's only other dimensional connection was severed then the cell could also be accessed along this dimension. This has not yet been implemented on the Mantra server, and currently one must add every cell manually to *d.history* if it is needed; however, once in place it could prove useful.

In relation to this dimension, several new commands could be developed. For example, `retrieve last`, which could return the last cell added. This command would be especially useful if the cell had been created and its dimensional connections severed accidentally. It could also be expanded to include the last x cells, the last dimension created, the last type, etc.

3.2.11. Zones of Purpose

When developing a client for use with the Mantra server it is important to think about what structure is needed. The systems cells should be hidden from the regular user, and only the 'Home' cell with the relevant client-specific cells would be viewable. Nevertheless, one of the main ideas behind ZigZag was to eliminate the *"application prison"* [12]; with this in mind, interaction between clients is important. Each client has a cell in home, from that cell they can create their structure; this is termed a *"Zone of Purpose"* (ZOP) [13].

However, there may be dimensional relationships between ZOPs; for example, an image organiser may contain photos of people, these photos could contain links along *d.people* to their names, and from that point tie in to an address book which contained their contact details, etc. Another connection could be between the dates the photos were taken, and a diary or organiser client, so links could be formed to events that occurred on a particular day. This type of structure

ensures that there is no redundancy, and that files and pieces of information can be shared between *"applitudes"* [13] without any problems, creating a wholly interlinked system.

3.2.12. Slices

A slice is a pseudo-grouping of cells existing on a server [15], where each cell lives in one slice and one only. To the user the slices may even be invisible, such as the system cells. Slices have been devised mainly with the purpose of memory management, but are part of where the cell currently exists. A cell on the Mantra server is identified by the server ID, the slice ID and the cell ID, giving it an address; i.e: 0.0.23 is cell 23 on slice 0 of server 0 (server 0 is equivalent to localhost).

Slices can be used to contain demonstrations of ZigZag structure, whereby to view the demonstration one merely has to load in the relevant slice. Slices can also be used to separate the system cells, which could either not be edited or only available to add dimensions, types, etc through the `create` command. Different clients will also be able to access their unique structures which exist on separate slices, or even over several slices. Dimensional connections are even possible across slices, allowing connectivity between information usually accessed by different clients; though it is important to save both slices at the same time to ensure they stay synchronised.

Slices have not yet been fully implemented on the Mantra server, and the existing set-up means that slices must be saved after each `create` or `insert` command to ensure that data is not lost [Figure 3.19], which can become tedious.

```
> 0.0.36 | fetch +1 along "d.contains" | fetch tail along "d.set" | create
+ along "d.set" | change content = "2005" | save slice 0 | resolve
```

Figure 3.19: Slices must be saved to ensure that recently created cells are not lost.

Slices could be saved automatically, and this would function effectively at the moment whilst the server is only acting on one slice. However, this would introduce problems once multiple slices were involved, meaning that the client would possibly have to save manually after each query. The client should be able to specify the slice it was working on; especially if cells were needed to be kept on separate slices, or demonstration slices were being created. On the other hand, if one were working with a structure that spanned several slices, then automatic saving of those slices would be useful to ensure they stayed synchronised.

There are potentially other problems related to the multiplicity of slices, including the synchronisation aspect of interaction between slices. If dimensional links are allowed across slices, then measures must be taken to ensure that they are preserved. If one slice were to be corrupted, or simply loaded out of memory, then relationships between cells would be lost and the integrity of the structure compromised. It certainly needs a great deal more consideration before it could be fully implemented.

Chapter 4: Design

4.1. Introduction

With the advent of affordable digital cameras, people are storing a plethora of digital photos and other images on their home computers. These images are most commonly organised into some kind of hierarchical directory structure, though this in itself in not ideal due to the fact that many images tend to fall under more than one category.

There are currently numerous good photo management programs available, such as iPhoto for the Mac [29] and Picasa for Windows [33]. These implement various ways to organise images, including tags, virtual folders, calendar views etc. Additionally, they also have developed into image editors, offering options such as reducing red-eye, cropping and changing the image contrast. There are several features that can be implemented ideally by ZigZag through an image organiser client, or through linking to other ZigZag clients:

* **Views** – due to ZigZag's immense flexibility, the creation of new views and the customization of the interface should be straightforward. Users will merely have to change the dimensions being displayed to get a whole new view, and there should be provision to save customized views. The default starting position will be the *home* cell, which should contain a set of menu-like cells that state what information is contained within the client, e.g: Images, Tags, Folders, and Dates etc. These cells will then contain a set of images, tags, etc depending on their contents.

There should also be the provision to change easily the display of the client, such as switching between a flat view with thumbnails view that is reminiscent of a Windows environment

organiser, a slideshow, or a ZigZag cell type environment. Tabs could also provide further functionality, allowing users to navigate with ease between the various categories.

* **Timeline** – with the use of a ZigZag dimension (e.g: *d.ordered*), it would be simple to organise all the images in the order they were created or modified. This could also be subject to customisation by the user, who should be able to create a custom timeline according to dates they enter for images rather than those stored by the computer.

* **Calendar** – images could also be grouped and viewed by date in a calendar dimension. Additionally, this could be used with an ordered dimension within a particular date to illustrate the progression of time.

* **Tags** – through the use of transclusion, tags can be used to group the images in several different virtual 'folders'. A search algorithm could also be implemented here. There are limitless possibilities for views here as well: tabbed, virtual folders, ZigZag cells, etc.

* **Collections** – collections of images can easily be created, without having to create new tags. These collections could represent relationships between images, or merely be used to group images together. Collections could also be used in collaboration with groups (mentioned below), or be created simply to group images together temporarily for a particular purpose.

* **EXIF (Exchangeable Image File) data** – information about the images can be stored alongside them; including author, copyright, rating, captions etc. This can also be used to sort images in different dimensions. Furthermore, EXIF data should be able to be stored with the image so that, in the event of images being transferred to a computer that does not use the ZigZag

framework, the information about the file will be preserved. EXIF data should be able to be viewed and edited easily in both a flat form view, or in a ZigZag set.

* **Emailing images** – the image organiser client can link with an email client to send images via email. This emphasises the facility in which data can be exchanged between ZigZag applitudes.

* **People** – the client should also be able to link to an address book or personal organiser client, connecting the people in the images with their contact information. This should begin to eliminate the need for information to be duplicated due to incompatibility between applications, and inevitably free up disk storage space.

Dimensions (such as *d.person*) could be used to display the image, linking to the names and contact details of the people in the image, and additionally displaying further images that also contain those people. Groups of people could also be created if multitudes of images contain the same people, or if a particular group repeats over a number of images.

4.2. Flash

"In a world where most digital experiences fall flat, the Macromedia Flash Platform offers something different. It is a lightweight, cross-platform runtime that can be used not just for rich media, but also for enterprise applications, communications, and mobile applications. No matter how it is used, we designed the Flash Platform so that organizations of all kinds can deliver the most effective experiences imagination will allow." [26]

Similar to Java, Macromedia Flash [26] is a dynamic streaming technology that allows animations, interactive forms, games and other features to be embedded in web pages, or it can be

used as a stand-alone application. It is a bandwidth friendly and browser independent vector-graphic animation technology; and as long as different browsers are equipped with the necessary plug-ins, Flash animations will look the same. Macromedia Flash Player is a well known and trustworthy plug-in that users should feel comfortable installing; it is freely available to download and is compatible with any system.

Flash was chosen to create the prototype because it is cross-platform compatible, and also usable on the internet. Additionally, it is possible to create a user-friendly interface without an enormous amount of coding, leaving the designer free to concentrate on connectivity and features. Flash MX with Flash Player 6 has also introduced some new accessibility features which make Flash movies accessible to screen readers, such as JAWS [30], and Window-Eyes [39]. The beauty of creating the interface in Flash is that the Flash Player can be maximised, which allows for the increase in size of the buttons and fonts.

4.3. Colour

When considering the design of the interface, particular attention was paid to the needs of people with disabilities such as colour-deficiency and partial sight. A questionnaire was devised to elicit preferred colour combinations of text and background, and font type and size, the full results of which can be found in Appendix C (see 4.7 for a discussion of the *Questionnaire*).

One must ensure that there is a clear contrast between the colour of the text and the colour of the background. It was found that the preferred colour combination for people with normal vision (using corrective spectacles if necessary) is black text on a white background. However, for those with vision problems, a white background can cause too much glare which reduces the visibility of the text. By using a pale coloured background such as grey or light blue, this reduces the glare and increases the contrast [Appendix C].

Additionally, large light coloured text on a dark background also provides good contrast; however, this is a less desirable colour combination for normal sighted people. To enable people with all levels of visual ability to use the interface comfortably, a pale background was chosen along with bold black text. To provide a selection depending on colour preferences, alternative versions of the client have been produced in different colours; and in future versions there are plans to add an option to select a custom background and text colour. This would add more versatility to the design and increase accessibility.

Negative contrast (dark text on a light background) provides a higher contrast sensitivity than positive contrast (light text on a dark background); this is because of certain asymmetries in the visual processing system; and additionally, the tendency of white letters to bleed into the dark background (this is called irradiation) [23 – Ch4]. In addition to the combination of colours, there are many other factors which can affect contrast sensitivity (CS); these include: illumination, spatial frequency (the width of light separating two dark lines), the resolution of the eye itself (level of vision), or movement (which is not applicable in this case) [Figure 4.1].

Variable	Effect	Example
Reduced contrast	Reduced visibility	Black print on grey
Reduced illumination	Reduced contrast sensitivity	Reading a map in poor conditions
Polarity	Black on white better than white on black	Designing viewgraphs
Spatial frequency	Optimum CS at 3 C/D (cycles/degrees - the number of light/dark pairs in 1 degree of visual angle)	Ideal size of text font given viewing distance
Visual accommodation	CS	Map reading during night driving

Figure 4.1: Some variables that affect contrast and visibility.

4.4. Colour Deficiency

There are several types of colour deficiency (also called *Daltonism*), which can affect different people in varying degrees [Figure 4.2]. There are four main types; three that affect the primary wavelengths of light: red, green and blue, and total colour blindness (achromatic).

Condition		**Proportion (%)**	
		Male	**Female**
Protanopia	Red-blind	1.0	0.01
Protanomaly	Red-weak	1.0	0.03
Deuteranopia	Green-blind	1.0	0.01
Deuteranomaly	Green-weak	5.0	0.35
Tritanopia	Blue-blind	Very small	Very small
Tritanomaly	Blue-weak	Very small	Very small
Achromatopsia	Total	Minute	Minute

Figure 4.2: Occurrence of colour vision deficiencies in the UK

The UK has one of the highest proportions of colour-blind people, along with Central Europe and Central North America [11 - page 38].

Colour deficiency genes are located on the X chromosome, which means that males are more likely to be affected by it and females are more likely to be carriers [Figure 4.3]; it can also be brought on by age, or as the result of an accident or illness. In fact, 5% of the population have an acquired defect as serious as the 8% with a congenital defect [11 - page 55].

X	Y	Males (observed)	(%)
R G		Normal	92
R* G		Red-deficient	2
R G*		Green-deficient	6

X	X	Females (calculated)		(%)
RG	RG	Normal	Non-carrier	84.7
RG	R*G	Normal	Red-deficiency carrier	3.7
RG	RG*	Normal	Green-deficiency carrier	11
R* G	R* G	Red-deficient	Red-deficiency carrier	0.04
R G*	R G*	Green-deficient	Green-deficiency carrier	0.36
R* G	R G*	Normal (red-weak poss)	Red & green-deficiency carrier	0.24

Figure 4.3: Possible gene distribution on 23rd chromosome

8% of males carry a deficient gene, as do 16% of females. There is a 50% chance of a son of a woman carrier being colour-blind, and 100% if she carries both defective genes (with a 50/50 chance of either colour). The colour-weak gene is dominant over the colour-blind gene; but is itself dominated by the normal gene [11 - page 49].

With colour deficiency, or the milder form of colour weakness, there is a lack or displacement of one of the three colour receptor mechanisms, which causes an inability to see certain colours; and confusion between colours arises. This means that colours that would usually be thought of as contrasting with each other can actually look the same. This can cause problems in everyday life, such as distinguishing traffic and taillights when driving; and can also affect people in industry and the forces, hence restricting the range of occupations available to those with colour problems [11 - Ch 7].

This has a serious impact on design considerations, so it is important that the colours used on the user interface have enough contrast at any wavelength. There are several programs available that can simulate an interface as seen by a person with a colour deficiency. A variety of colours have been selected to illustrate the differences, these can be seen in [Figure 4.4]. The twenty-seven normal colours have been produced by all combinations of 0, 50 and 100% levels of red, green and blue primaries. They have then been transformed to show their appearance to colour-blind users [11 - page 213].

The diagram shows that the safest colours to use are black, white and grey; this is why these were selected as the initial colours for the interface. The light blue colour is seen as a pinkish shade by those with a red or green colour deficiency; however, it still retains sufficient contrast with black to justify its viability as a background colour.

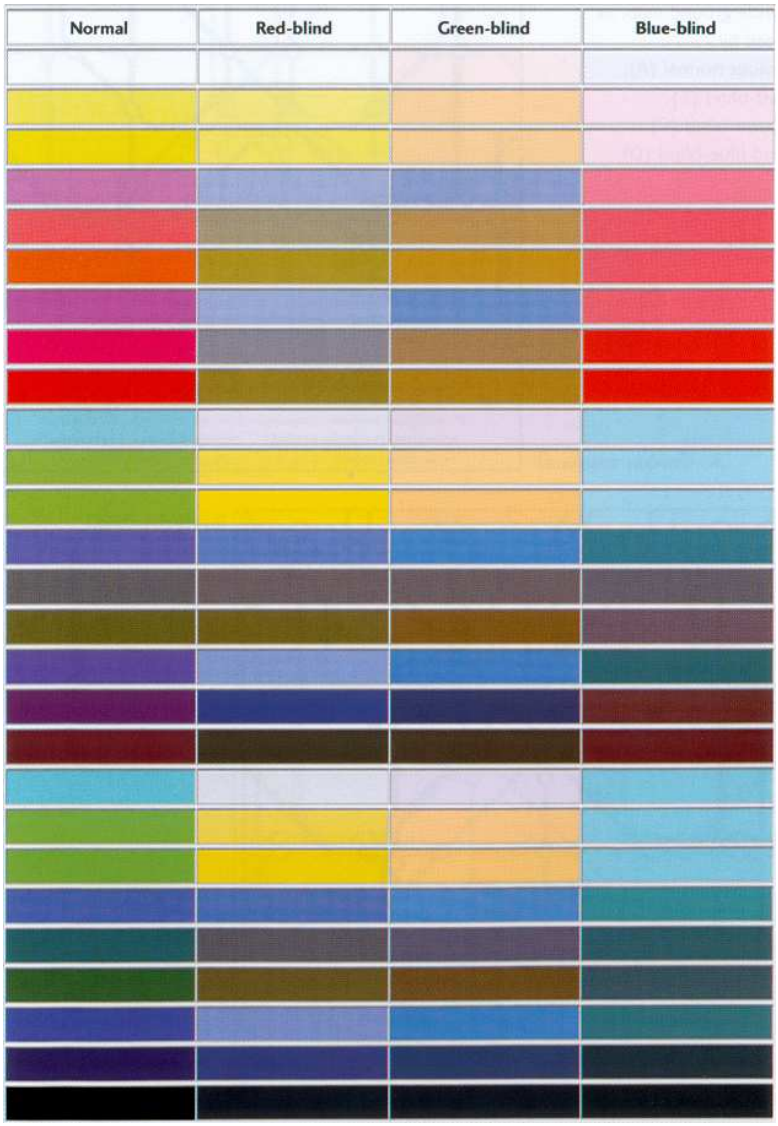


Figure 4.4: Websafe selection (Normal – Red-blind – Green-blind – Blue-blind)

4.5. Fonts and Buttons

The font Verdana at a size of 12 points has been used in the interface as the questionnaire showed it to be the preferred font of the majority of those with normal vision and partial sight. The questionnaire also indicated that the interface should be coded to include key shortcuts for all buttons. This ensures that persons with motor control problems have no issues clicking smaller buttons. Having key shortcuts also enables those with screen readers to activate buttons without many problems.

Key shortcuts were originally coded with multiple keys (e.g: Ctrl + 7) [28]; however when the listeners were added they did not function as expected, perhaps due to incompatibility between Flash MX and Flash MX 2004; and time constraints did not allow more extensive testing. As a consequence, the listeners were removed and the simple one key press shortcut was kept. This did mean that shortcut keys could not be added to the 'Add Image' page due to the fact that this interfered with typing in the image information. However, with more research into the key listeners, they should be able to be suitably modified for use in future versions. The key shortcuts are detailed in the Help file supplied with zzImage, which is accessible via the 'Help' button; and additionally, in the User Manual [Appendix A].

4.6. Accessibility

As mentioned previously (see discussion of *Flash* in 4.2) the most recent versions of Flash and the Flash Player have included more functions to increase the accessibility of Flash movies. Using this feature, alongside sensible coding and limited use of animated buttons, the interface can be made available to the widest range of users. While it is quite popular to include motion in movies as part of transitions and loading sequences, it is important that these animations settle to

a static screen once the page loads. For people with learning disabilities, motion on the screen can be distracting and might even make other elements unreadable [27]. Additionally, content that continuously loops can confuse screen readers into thinking that new content is available and they keep returning to that section of the page. Keyboard shortcuts make buttons more accessible and larger buttons mean that the target area for the mouse point is larger.

Furthermore, the Web Accessibility site recommends the following measures should be taken [38]:

Hearing disabilities

- Provide synchronized captions for any audio that conveys content

Photo epilepsy

- Remove strobing content that flashes between 2 and 55 times per second

Motor disabilities

- Ensure the Flash content is keyboard accessible
- Do not require fine motor skills

Cognitive disabilities

- Give users control over time sensitive content
- Provide easy to use controls and navigation schemes
- Be consistent
- Use the clearest, simplest language appropriate to the content

Low vision

- Provide plenty of contrast
- Allow the Flash content to scale to a larger size

Blindness

- Ensure screen reader accessibility or provide an accessible alternative
- Ensure keyboard accessibility
- Do not interfere with screen reader audio or keyboard commands
- Provide textual equivalents for all non-text elements that convey content or provide a function.

4.7. Questionnaire Design

Questionnaires are a well-established means of collecting data or opinions. A certain amount of skill is needed to make sure that the design of the questionnaire is such that the data collected is appropriate, and that it can be analysed effectively. There has to be a balance between open and closed questions in order to elicit the relevant information.

Online questionnaires are becoming increasingly common because they are effective in reaching a large audience quickly and easily [19 - Ch13]. Web based questionnaires can provide immediate data validation and can enforce various rules about questions such as, 'select only one of the following'; this reduces the amount of time needed for data analysis. One drawback of a web-based questionnaire is obtaining a random sample of respondents; though this is sometimes a willing compromise when a rapid response is needed.

The questionnaire (which can be viewed in Appendix C), used to obtain data mainly about colour preferences and fonts, was posted online. It was designed firstly on paper, and went through many revisions before being publicised. An online journal site [31] was used to host the questionnaire, since it has a tool which enables questionnaires to be generated with a minimum of effort, and tallies the results of any closed questions automatically. The only drawback with using this tool was that the questionnaire was limited to fifteen questions; however, this proved to be sufficient in this instance.

The aim of the questionnaire was to elicit responses from a wide range of people with varying visual abilities. The advantage of using the journaling site meant that the questionnaire could be advertised in the relevant communities so that a sufficient sample of answers could be obtained. Special attention was paid to attract those people with reduced vision, and those who had various colour problems; additionally, a sample of people with normal vision was obtained for comparison.

4.7.1. The Questions

The questionnaire was designed to try and provide a balance between the need for data that would be quick to analyse, and the need for open answers that may cover eventualities that had not been anticipated. Participants were encouraged to add any further data or explanations in the comment box provided, or via email. Additional questions could have been added; however, it was decided that enough data could be gathered from the original questionnaire and there was no need for any supplementary questions.

The design of the questions was such that the data obtained could be analysed quickly so that various conclusions could be drawn. There were some instances where it was suggested that a question could have been formatted differently, such as allowing the respondents to choose more than one interface colour, or displaying the font in the actual typeface. However, these were minor, and had been considered previously.

The problem with allowing participants to select more than one interface meant that some might select them all, whereas the decision presented to them forced them to make a conscious choice between the designs. Furthermore, the fonts were supposed to be displayed in the appropriate typeface; however, a problem with the questionnaire generator caused this formatting to be lost.

4.7.2. The Results

There were discussions regarding the opinion that the interface contained too many areas of focus with having the sidebar buttons split into two groups; however, this positioning was also deliberate as it is important to separate out areas of interest so the eye does not become crowded. It is essential to keep the view button where it is, due to the fact that the eye automatically starts

reading at the top left corner [19], and changing the view type would be one of the first actions a user would execute.

Several people made comments with regards to the positioning of the buttons; they were of the opinion that the bottom row of navigation buttons should be relocated to the top left. However, there was disagreement, and it was decided that the placement was deliberate as it groups the entire set of navigation buttons together out of the way. They should not become the main focus of the eye, as they are secondary. The main buttons are grouped in the bottom left, which do not immediately draw the eye, but they are placed logically as the screen is read from left to right.

The results obtained from the questionnaire supported conclusions drawn previously, and sparked some new ideas which were incorporated into the design. Although not everyone answered every question, a sufficient sample was obtained for analysis. Additional analysis was performed on the answers of those with reduced vision; those registered as legally blind and those with colour problems. These supplementary results also correlated with those of the whole, thus supporting the conclusions drawn.

Chapter 5: Implementation and Evaluation

A full code listing and a User Manual can be found in Appendices D and A respectively.

5.1. Connecting to the Mantra Server

Code written for the MantraF client was adapted (with permission) from Flash MX 2004 to Flash MX for use with the zzImage client. This enabled the client to make use of the query and query stack classes, which provided a quick and easy method of sending messages to, and receiving them from, the Mantra server by opening an XML socket [Figure 5.2]. Once messages are received then they can be executed by the function that requested the information. The client connects to the server on a specified port, which identifies the user and links them to their pre-defined structure [Figure 5.1]. There are currently no other security measures on the server, so users are separated only by the port they access.

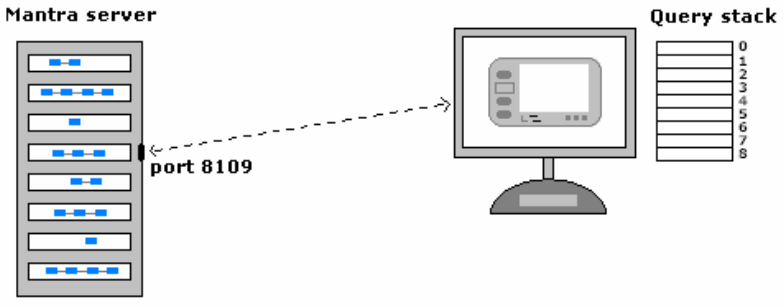


Figure 5.1: The client accesses the Mantra server on the specified port

```
function server_connect(ip, port)
{
      xsock = new XMLSocket();
      xsock.onData = process_xml_message;
      xsock.connect(ip, port);
}
```

Figure 5.2: The server_connect function

The client sets up, and connects to, an XML socket, which will send received data to the specified function.

When a message is sent by the client, the query stack stores the query ID alongside a function pointer to the function that will process the XML. When the client receives a message from the server it searches through the query stack to find the corresponding query ID. It then calls the linked function and sends the XML content to it; along with any extra parameters it may need [Figure 5.3]. By simply parsing the XML received from the server, the contents and addresses of cells can be extracted for display.

```
function process_query_response(xml_doc)
{
      var response_id = xml_doc.firstChild.firstChild.attributes.id;
      var query_match = query_store.find_entry(response_id);

      if (query_match == null)
      {
            trace("error: matching query for response cannot be found");
```

```
        }
        else
        {
                query_store.remove_entry(response_id);
                query_match.function_name(xml_doc, query_match.param);
        }
}
```

Figure 5.3: The process_query_response function

Once the previous function has checked that the message is a server response, then the response can be processed. The function gets the response ID from the XML message. It then searches through the query stack to find the matching query ID. When the match has been found it removes the entry, and sends the XML document and any necessary parameters to the function which needs the data. The function is stored as a function pointer, which is a link to the function so it can be sent with the appropriate parameters.

5.2. The Slideshow

The Normal View displays the images it receives as a slideshow. When the client is connected to the server, then the images can be loaded by clicking the "Load Images" button. When this occurs, the client contacts the server and asks it to retrieve all the cells it currently has connected to the 'images' cell [Figure 5.4].

```
function load_images() {
      var dim_query = '0.0.34 | fetch tail along "d.contains" | fetch all
                                    along "d.set" | resolve';
      _root.submit_query(dim_query, createSlideshow);
      p = 0;
}
```

Figure 5.4: Fetch all images, and send the result to the createSlideshow function

The addresses of the images are then retrieved from the XML and pushed into an array [Figure 5.5]. This array is then stepped through and loaded into a movie clip in a slideshow format [Figure 5.6]. Users can step through the images using the arrow buttons, or using the left and right arrow keys on their keyboard. A caption for the image is displayed underneath, alongside the position of the image in the array.

```
function createSlideshow(xml_doc)
{
      xmlNode = xml_doc.firstChild.firstChild.childNodes;
      image = [];
      address = [];
      total = xmlNode.length;
      for (var i = 0; i<total; i++)
      {
            image[i] = xmlNode[i].firstChild;
            address[i] = xmlNode[i].attributes.nri;
      }
```

```
        firstImage();
}
```

Figure 5.5: The response from the server is sent to the createSlideshow function

The addresses and contents of the cell are extracted and stored in an array. The address can be used to gather the other information about the image needed for the sidebar, and also the caption.

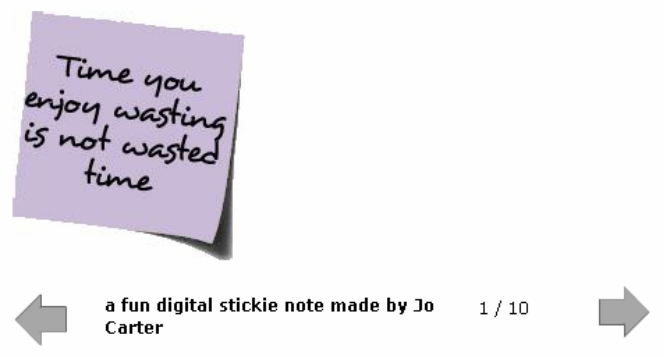


Figure 5.6: The images are loaded into the slideshow

The only drawback with this method is that Flash does not support .png or .gif format, which means that only .jpg type images can be displayed. Although the 'Back' and 'Goto' buttons have not been coded in this prototype, the 'Home' button returns the slideshow to the first image.

One major problem that was experienced with the slideshow was asynchronous loading of the images, which resulted in a show-one skip-one type progression; the progression even occurred

when the 'Home' button was pressed, when the images were reloaded [Figure 5.7] and was mirrored by the thumbnail image.

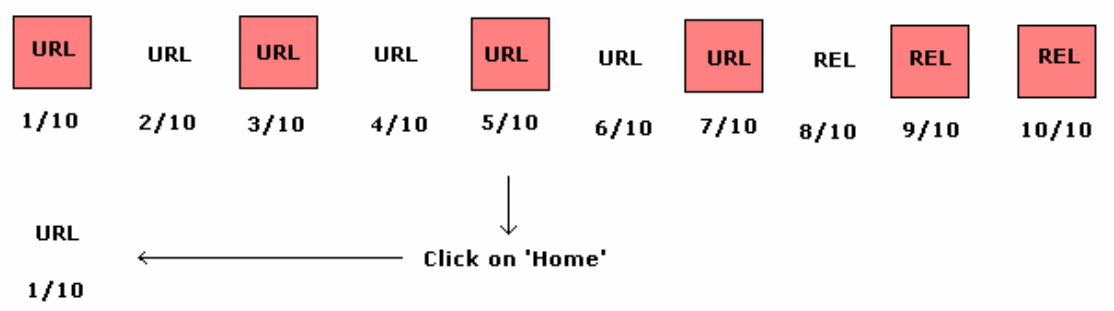


Figure 5.7: The show-one, skip-one progression pattern repeats infinitely

URL indicates where an image is loading from an URL, and REL indicates where an image is loading from the computer where the client is located (the images are in the same folder).

The problem was suspected to be related to the speed of the connection to the Mantra server. This was tested with both URLs and relative paths to images, and the results of this were inconclusive. Any images using a relative path that were directly after the image loaded from an URL would load in the show-one skip-one pattern; however, within their own range they would all display, as would the next one from the URL [Figure 5.7].

After extensive research it was discovered that this problem was caused by the fact that Flash was caching the images and there was a simple solution to prevent this from happening; the movie clip must be unloaded before a new image can be loaded in [Figure 5.8], and the same with the thumbnail clip. The images now all display, without any problems.

```
unloadMovie(picture);
loadMovie(image[p], picture);
```

Figure 5.8: A simple solution to the show-one, skip-one problem

5.3. The Sidebar

The sidebar on both views displays a thumbnail version of the image that is currently selected or being rolled over (if an image is selected), including extra information about the images such as the tags, the name, etc [Figure 5.9].

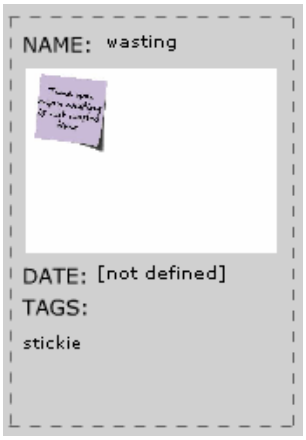


Figure 5.9: A thumbnail version of the image is displayed on the sidebar

It is propagated by the `on_focus()` function, which records the current cell's NRI and executes the first query. The queries and their respective functions are executed consecutively. This can result in a slight time lag in the text being propagated to the respective text boxes, and this is somewhat noticeable if there is a slow connection between the client and the Mantra server. The respective functions check whether each property of the image has been defined, and if not then it displays [not defined].

If there are no tags defined then the query which is sent to the server can end up returning the entire set of images instead of the set of tags. To avoid the image addresses being output in the place of the tags, the set needs to be cycled through and the NRIs compared to the current image's NRI. If the NRI is in the set, then the tags have not yet been defined [Figure 5.10].

```
function showTags(xml_doc)
{
	xmlNode5 = xml_doc.firstChild.firstChild.childNodes;
	var no_tags = false;
	var _total = xmlNode5.length;

	for (var i = 0; i < _total; i++)
	{
		if (xmlNode5[i].attributes.nri == current_cell)
		{
			no_tags = true;
		}
	}

	if (no_tags == true)
	{
		image_tags.text = "[no tags]";
	}
	else
```

```
	{
		var tags;
		tags = xmlNode5[0].firstChild;
		for (var i = 1; i < _total; i++)
		{
			tags = tags + ", " + xmlNode5[i].firstChild;
		}
		image_tags.text = tags;
	}
}
```

Figure 5.10: The showTags function

5.4. The ZigZag View

The ZigZag View displays the ZigZag structure as a series of connected nodes. This code has also been adapted (with permission) for Flash MX from the MantraF client. Once connected to the server, the client queries it to obtain the cells, and then creates an instance of a graphic (node_container) to represent the cell [Figure 5.11]. The client constructs the ZigZag structure by querying the server for every related cell within six cells of the 'Home' cell. If the specified range does not exist, then the function merely skips over it.

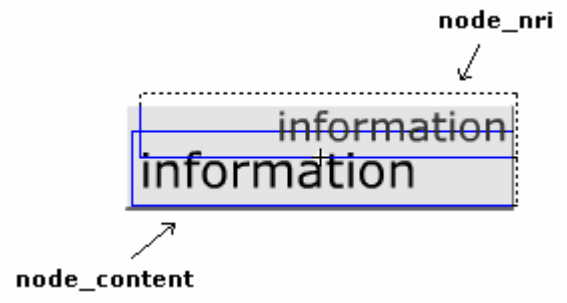


Figure 5.11: An instance of node_container

```
update_nodes[i].mc = nodes_mc.attachMovie("node_container",
                                          "node_container_"+i,
                                          (30+i));
```

Figure 5.12: The code to attach the movie clip

The code attaches the movie clips, gives it a unique name, and specifies the level into which it needs to be loaded.

Once all the cells have been retrieved then the `update()` function constructs the cell movie clips, making sure that each one is loaded onto a different level [Figure 5.12], and displays them on the screen [Figure 5.13]. To prevent the cell positions having to be recalculated every time the interface changes, the cells are attached to another movie clip called `nodes_mc`; this is then the only thing which has to be repositioned. The 'Home' button centres the structure back on the

zzImage home cell; this is very useful if one has become disoriented navigating through a complicated structure.

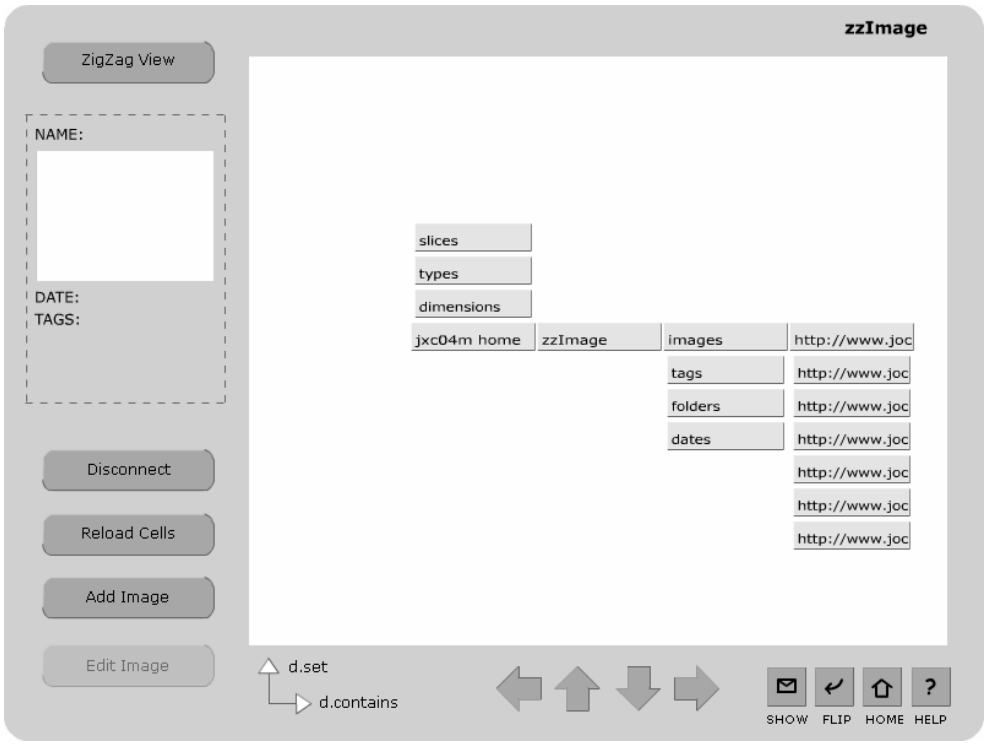


Figure 5.13: The ZigZag view, after the cells have been loaded

The cell's addresses can be shown or hidden by clicking the 'Show' button [Figure 5.14].

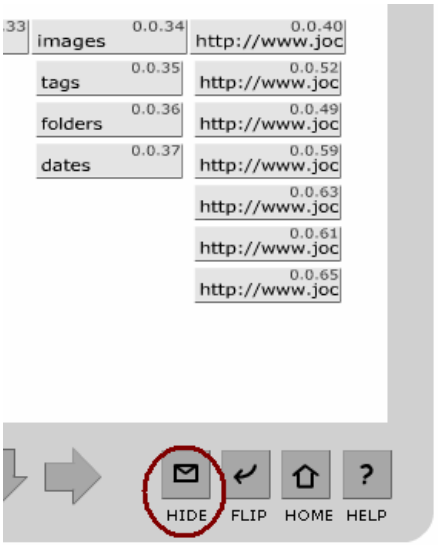


Figure 5.14: The cell addresses can be toggled on and off by the 'Show/Hide' button

The dimension lists are pop-ups, which are created by using a movie clip; these play when an invisible button is clicked (or the shortcut key is pressed) [Figure 5.15]. The same button also closes the pop-up, or one can also use the close button. The pop-up contains a list box that is propagated through the use of a useful Mantra query (`gather dims`). A dimension can be selected, and when the pop-up is closed the selection is stored in a variable and displayed.

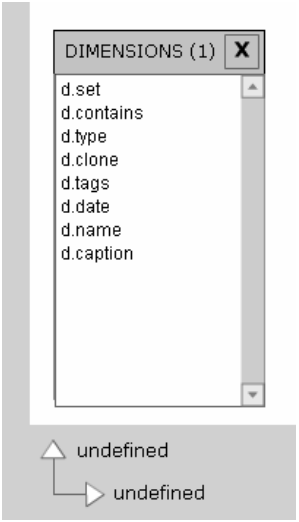


Figure 5.15: The dimension list pop-up

A 'Flip Dims' button swaps the selected dimensions around. This rotates the structure 90^{o} and gives the user a different viewpoint; with complex structures in particular, this option can prove fortuitous.

Some problems were experienced in the conversion of the code from Flash MX 2004 to Flash MX due to the fact that classes are declared in a different manner, which means that some code will need to be altered drastically. Code that worked perfectly with Flash MX 2004 caused massive problems with Flash MX. One of the initial problems was with loading the ZigZag cells, as only half of them were loading. After extensive testing and debugging it was discovered that the problem was due to the `splice(match, 1)` command and that splicing the array caused only part of it to be processed, displaying only half of the cells. Once this line of code was removed the problem was solved.

One major problem experienced was the difference between MX and MX 2004 classes. MX 2004 uses Java like classes declared through the `class` command, containing the various functions; these classes can then be instantiated. MX requires one to create a function with the name of the class, then create prototype functions, which are then attached to the 'class' function. This means that all variables that are needed in that class need to be assigned inside the 'class' function; even if they are not passed to the constructor [Figure 5.16].

```
function mantra_node(_nri,
                 _content,
                 _target_xpos,
                 _target_ypos,
                 _target_height,
                 _target_width)
{
      this.nri = _nri;
      this.content = _content;
      this.target_xpos = _target_xpos;
      this.target_ypos = _target_ypos;
      this.target_height = _target_height;
      this.target_width = _target_width;
      this.mc = new MovieClip;
      this.current_xpos = 0;
      this.current_ypos = 0;
      this.current_height = 0;
      this.current_width = 0;
}
```

Figure 5.16: The mantra_node class in Flash MX

A disappointing drawback of the conversion of the code from Flash MX 2004 to Flash MX was that the cell sensors ceased to function. The sensor was designed to detect when a cell was clicked and to redraw the layout in order to centre on that cell. With further testing it should be possible to include this functionality in subsequent versions.

5.5. Adding an Image

The 'Add Image' screen follows the same format as the other screens. The user has the option of entering a location for the image, giving it a name and a caption [Figure 5.17].

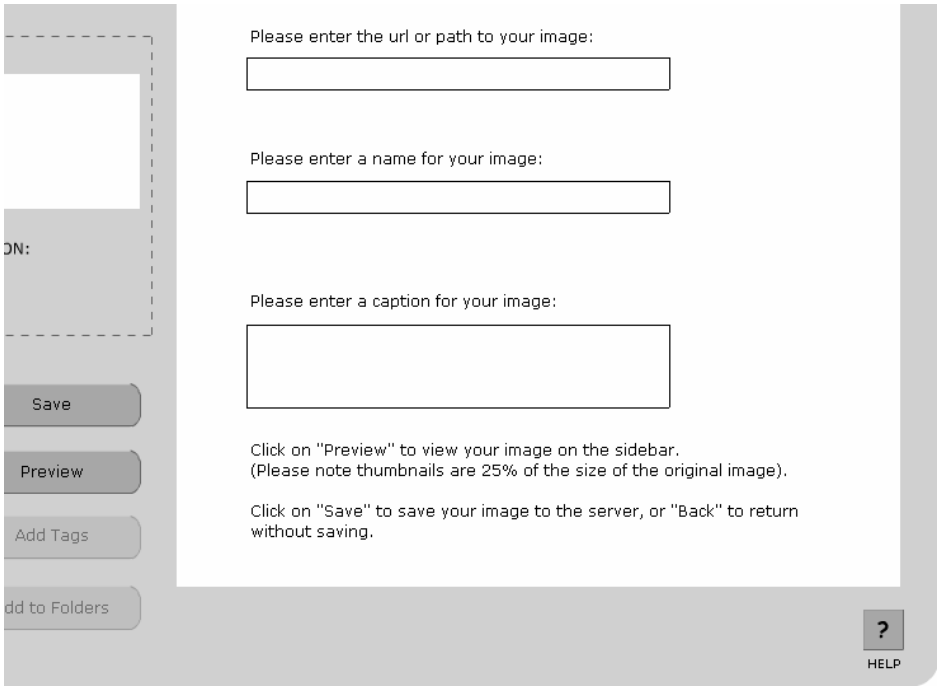


Figure 5.17: The 'Add Image' page

There is a choice of saving the image directly, going back to the previous screen (either the ZigZag or the Normal view, depending on where the user started) without saving, or previewing the image.

The preview loads a thumbnail version of the image in the sidebar, and displays the name and caption on either end. The quality of the thumbnail is mediocre at best, due to the fact that the movie clip displays it at a quarter of the original size; however, it provides enough of a view of the image for the user to check that the image is correct. If the path is incorrect, then the image will not load [Figure 5.18] and it will enable the user to re-enter a correct address without adding incorrect data to the cell. Potentially in the next version there should be an error message displayed if the image cannot be located. This ability to preview reduces the amount of inaccurate data being added, and reduces the amount of queries being sent to the server. The aim of this is to reduce user stress, and repetitive re-entry, therefore increasing user satisfaction.

```
> Error opening URL
file:///G|/University/MSc%20IT/Dissertation/Image%20Viewer/test.JPG
```

Figure 5.18: Error shown in debugging console

This error occurs when an image that does not exist is attempted to be previewed.

The user is given visual feedback, in the form of a greyed out ‘Saved’ button, to show that an image has been saved successfully.

The user will also have the chance, in future releases, to add tags to the image and put the image in virtual folders from this page. As these functions require the use of the `clone` command, which has not yet been fully implemented and currently requires a lengthy query, they are planned for the next version. It is envisaged that the user will only be able to add existing tags

and folders to an image, and a supplementary screen will be provided to add new tags and folders; this functionality is planned for forthcoming versions.

5.6. Changes made during implementation

Not all features were implemented in the prototype, due to the fact that the Mantra server is in the early stages. This means that all the necessary commands have not yet been executed server-side; and also some commands are more complicated than necessary (see 3.2 for discussion of *Mantra*). In addition, some changes have been made to the initial design as it was created in Flash. The tabbed interface of the Normal View was impractical to implement and was therefore replaced with a pop-up menu choice (currently unavailable); which made it more consistent with the design of the ZigZag View.

The original idea of a thumbnail view was modified to become a thumbnail of the picture on the sidebar, and a slideshow of images. In future versions it may be preferable to add thumbnails to the navigation as the number of images increases. The addition of tags and virtual folders, and the navigation through them, has not been implemented due to lack of suitable commands.

The storage of EXIF information with the image has not been included in this prototype because the cells just contain a link to the image rather than the image itself. Once ZigZag cells are able to contain images then the EXIF information will be obtained, which means that less data concerning the image will need to be entered.

5.7. Testing

A testing plan was devised to test the speed of various functions. The `trace()` function was used to output the milliseconds of the current time, and these times were compared to obtain an average 'loading' time for each function. The speed test was conducted several times to get an average result. The speed tests were conducted over Broadband (ADSL) at a speed of 1.1 Mbps between the hours of 11:00pm and 11:30pm. Please note that speeds may vary depending on connection and server traffic.

The ensuing test plan was followed, obtaining reassuring results [Figure 5.19].

- Connect to server by clicking the 'Connect' button
- Click the 'Load Cells' button – took an average of 192 milliseconds to construct and display the cells, including parsing the XML (16 cells displayed).
- Click the 'Flip Dims' button – took an average of 115 milliseconds to construct and display the cells, including parsing the XML (4 cells displayed).
- Click on the up dim and choose "*d.contains*" from the Dimensions pop-up – took an average of 104 milliseconds from sending the query to the Dimensions being populated in the list box. It then took an average of 120 milliseconds to construct and display the cells, including parsing the XML (16 cells displayed).
- Disconnect from server by clicking the 'Disconnect' button

- Switch to Normal View by clicking on the 'ZigZag View' button
- Connect to server by clicking the 'Connect' button
- Click the 'Load Images' button – it took an average of 57 milliseconds to obtain and parse the XML ready for displaying in the slideshow. It then took a further 212 milliseconds (on

average) to obtain and display the cell information on the sidebar – it has been noted that this procedure can seem to take a while if there is a slow connection.

- Click the 'Next' arrow button, or hit the right arrow key – it took an average of 198 milliseconds to display the information in the sidebar, including the caption under the image.
- Click the 'Add Image' button
- Enter image details
- Click the 'Preview' button – it took less than 1 millisecond to display the preview.
- Click the 'Save Image' button – it took an average of 266 milliseconds to save the image and the other details entered.
- Click the 'Back' button
- Disconnect from server by clicking the 'Disconnect' button

Figure 5.19: Test plan and results

Continuous testing and debugging were conducted throughout the lifespan of the prototype to ensure that all problems were detected and corrected as soon as they occurred, in order to prevent conflicts between areas of the code. Code was added in small sections and each component was tested on its own before being added to the entire program. Each time a new component was added the program was assessed to verify that no conflicts had arisen. Throughout the testing various bugs were discovered, which have been discussed previously, and the majority of these were able to be rectified.

5.8. Conclusions

The idea behind hyperstructure is that the user, rather than the application developer, should be in charge of how their data is organised. Computers should help us with organizing our lives, rather than making them more difficult. Hyperstructure, in particular ZigZag, allows us to structure our systems around the things that we care about and to express the relationships between them. Hyperstructure can be of particular use in the organisation of multimedia, such as images. This type of data is very amenable to interpretation, and there is no simple way to define the connections between different pieces of information as relationships between items of data can overlap.

The expectations of users and their reliance on technology have grown faster than the quality of user-interfaces, leading to much frustration on the part of the user [21]. Some of this frustration stems from the fact that the needs of the user differ from the needs of the designer. In particular, the contrasting need of the designer to show off their skills and their abilities in new technologies, as opposed to giving consideration to the dissatisfaction that the user experiences when faced with learning how to use yet another new complex product that still does not have all the functionality that they would wish for.

The objective is to help the user to understand better the relationships between the items in their lives and to provide them with the information that they need, in the right contexts, rather than having to open a different application or search for a file. The aim is to help them to organise their ideas and thoughts, and to structure everything else in their computer around them. The system should be based on our knowledge of those things that are inherent to the way our minds and bodies work, and on the interaction between humans and their artefacts; starting with a model of how we work as opposed to a model of how machines work.

Based on the idea of ZigZag [14] with foundations in Dimensional Informatics [24], the Mantra server [32] allows the developer to implement hyperstructure in such a way that they can offer a

variety of different views to a user to display their information. Through the use of applitude clients that connect to the server, information can be shared between "*zones of purpose*" [13], not confined to an "*application prison*" [12]. Moreover, duplication of data items will become redundant with the use of transclusion.

References

[1] Brailsford, T.J. "*The Mantra Bible*". University of Nottingham, August 2005. [Restricted distribution, available via: http://www.cs.nott.ac.uk/~tjb/mantra-docs/mantra.html, distribution policy at: http://www.cs.nott.ac.uk/~tjb/mantra-docs.html].

[2] Brailsford, T.J. and Goulding, J.O. (Personal communication). "*ZQL 0.0.2 - A new discussion of concepts*". January 2005. Available in the www.hypertexture.com Developer's forum at: http://avexa128.valuehost.co.uk/ZZ/forum/viewtopic.php?t=81 [Restricted access, see above for details].

[3] Buneman, P. "*Tutorial: Semistructured data*". Proceeding of ACM Symposium of Principles of Database Systems, 117-121 ACM, New York 1997

[4] Codd, E. F. "*Extending the relational database model to capture more meaning*". ACM Trans on Database Systems, 4, 4, 394-397 ACM 1979

[5] Connolly, T. and Begg, C. "*Database Systems: A Practical Approach to Design, Implementation and Management*". Addison-Wesley 2005

[6] Fallenstein B. and Lukka T.J. "*Hyperstructure: Computers built around things that you care about*". 2004. Available via: http://fenfire.org/manuscripts/2004/hyperstructure/

[7] Hughes. "*Dust or Magic: Secrets of a Successful Multimedia Design*". Addison-Wesley 2000

[8] Lukka, T.J. and Ervast, K. "*GZigZag – a Platform for Cybertext Experiments*". In: CyberText Yearbook 2000, Eds Eskelinen, M and Koskimaa, R. Eastgate Systems, MA 2000. Available via: http://gzigzag.sourceforge.net/ct/ct.html

[9] McGuffin, M. and Schraefel, M. "*A Comparison of Hyperstructures: Zzstructures, mSpaces and Polyarchies*". Proceedings of ACM Conference on Hypertext and Hypermedia, 153-162 ACM 2004.

[10] McHugh, J., Abiteboul, S., Goldman, R., Quass, D. and Widom, J. "*Lore: A Database Management System for Semistructured Data*". SIGMOD Record, 26, 3, September 1997

[11] McIntyre D. "*Colour Blindness: Causes and Effects*". Dalton Publishing 2002

[12] Moore, A. and Brailsford, T. "*Unified Hyperstructures for Bioinformatics: Escaping the Application Prison*". Journal of Digital Information, 5, 1, Article 254, May 2004.
Also available via, http://jodi.ecs.soton.ac.uk/Articles/v05/i01/Moore/

[13] Moore, A., Goulding, J., Brailsford, T. and Ashman, A. "*Practical Applitudes: Case Studies of Applications of the ZigZag Hypermedia System*". Proceedings of ACM Conference on Hypertext and Hypermedia, ACM 2004

[14] Nelson, T.H. "*ZigZag Tutorial*". 1999.
Available via: http://www.xanadu.net/zigzag/tutorial/ZZwelcome.html

[15] Nelson, T.H. "*Cosmology for a different computer universe: data model, mechanisms, virtual machine and visualisation infrastructure*". Journal of Digital Information, 5, 1, Article 298, July 2004

[16] Nelson, T.H. (Personal communication). "*ZigZag, Floating World and transclusion*". Talk held at the University of Nottingham, on 14th July 2005.

[17] Norman, D. "*The Design of Everyday Things*". Basic 2002

[18] Norman, D. "*Emotional Design: Why We Love (Or Hate) Everyday Things*". Basic 2004

[19] Preece, J., Rogers, Y., and Sharp, H. "*Interaction Design: Beyond Human-Computer Interaction*", Wiley & Sons 2002.

[20] Raskin, J. "*The Humane Interface: New Directions for Designing Interactive Systems*". Addison Wesley 2000

[21] Shneiderman, B. "*Designing the User Interface: Strategies for Effective Human-Computer Interaction*". Addison Wesley 2004

[22] Suciu, D. "*Management of Semistructured Data*". SIGMOD Record, 26, 4 December 1997

[23] Wickens, C.D., Lee, J.D., Liu, Y. and Becker, S.E.G. "*An Introduction to Human Factors Engineering*". 2nd Edition, Pearson Education Ltd. 2004.

[24] "*Dimensional Informatics*", Wikipedia, (this page has since been deleted - please see http://www.hypertexture.com for more information), http://en.wikipedia.org/wiki/Dimensional_informatics

[25] Fenfire Project Homepage, http://savannah.nongnu.org/projects/fenfire/

[26] Flash (Macromedia), http://www.macromedia.com/platform/

[27] Flash (Macromedia), "*Accessibility and Macromedia Flash MX 2004 Animation*", available from: http://www.macromedia.com/macromedia/accessibility/features/flash/animation.html

[28] Flash MX 2004 (Macromedia) LiveDocs, "*Specifying advanced accessibility options for screen readers*", available from:

http://livedocs.macromedia.com/flash/mx2004/main_7_2/wwhelp/wwhimpl/common/html/wwhelp.htm?context=Flash_MX_2004&file=00000565.html

[29] iPhoto 5, Apple Mac, http://www.apple.com/ilife/iphoto/

[30] JAWS Screen Reader, http://www.hj.com/fs_products/software_jaws.asp

[31] LiveJournal, http://www.livejournal.com/

[32] Mantra discussions on Hypertexture web forums.
Available via: http://www.hypertexture.com, open community forums currently hosted at:
http://avexa128.valuehost.co.uk/ZZ/forum/viewforum.php?f=1

[33] Picasa 2, Google, http://picasa.google.com/index.html

[34] "*The Semantic Web*", W3C, http://www.w3.org/2001/sw/Activity

[35] "*Set Theory*", Wikipedia, (September 2005), http://en.wikipedia.org/wiki/Binary_relation

[36] "*Turtles all the way down*", Wikipedia, (September 2005),
http://en.wikipedia.org/wiki/Turtles_all_the_way_down

[37] University of Nottingham, Web Technologies Research Group,
http://www.cs.nott.ac.uk/Research/wtg/

[38] WebAIM (Web Accessibility in Mind), "*Creating Accessible Macromedia Flash Content*", October 2004. Available from: http://www.webaim.org/techniques/flash/

[39] Window-Eyes Screen Reader, http://www.gwmicro.com/

NB: All websites were accessed on September 13th 2005 to ensure that content had not changed.

Appendix A: User Manual

zzImage - the ZigZag/Mantra Image Organiser

A Quick Introduction

A.1. Keyboard Shortcuts

The following keyboard shortcuts can be used in zzImage.

NB: Please note that keyboard shortcuts are not enabled on the 'Add Image' page at this time.

'C' -> Connect to/Disconnect from the server

'L' -> Load the cells/images

'A' -> Add an image

'H' -> Return to the home cell/first image

'V' -> Switch between views

'P' -> Load the Help file

A.1.1. ZigZag view commands:

'F' -> Flip the Dimensions

'S' -> Show/Hide cell addresses

'1' -> Show/Hide the Up Dimension pop-up

'2' -> Show/Hide the Across Dimension pop-up

A.1.2. Normal view commands:

Left/Right -> Navigate through the images

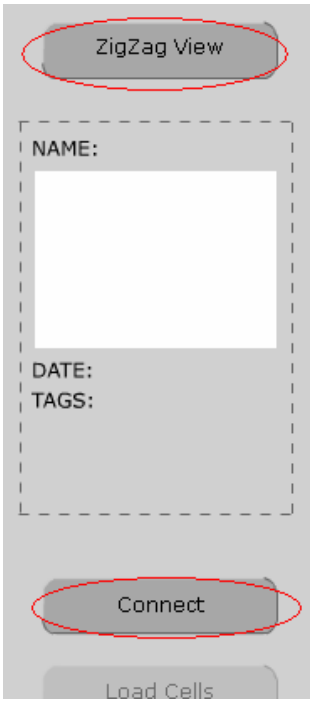
If you have any problems, suggestions or questions then please email me: jxc04m@cs.nott.ac.uk.

A.2. Loading the Image Organiser

Double click on the zzImage file to start the image organiser. Once the Splash screen (pictured below) has loaded, you can click anywhere on it to continue.

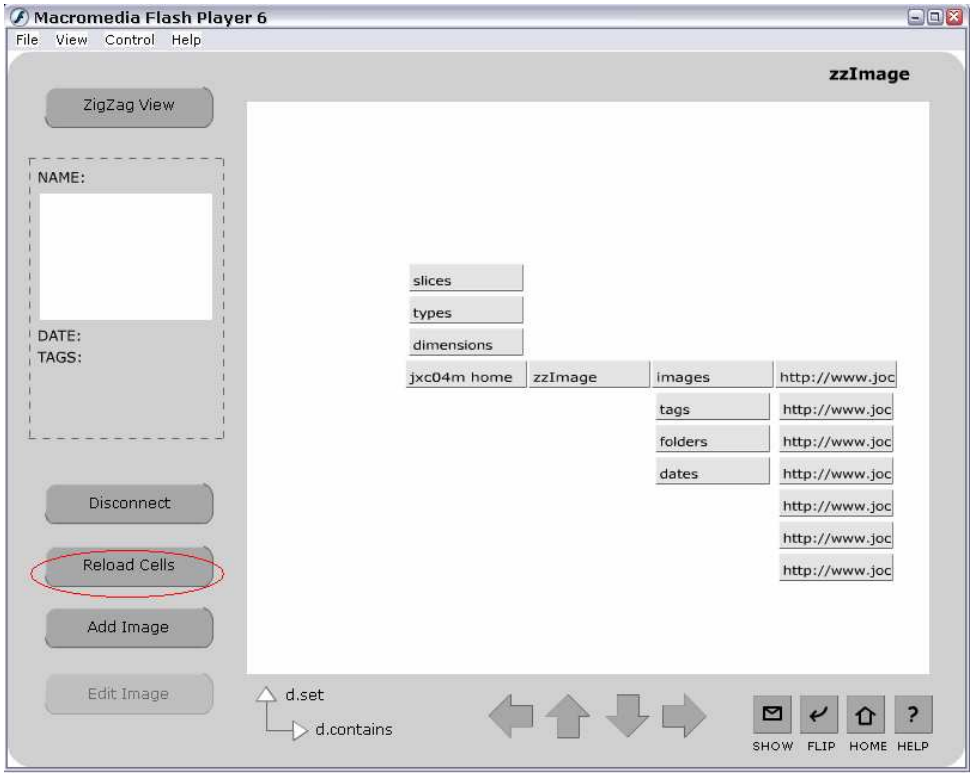
A.3. Connecting to the Server

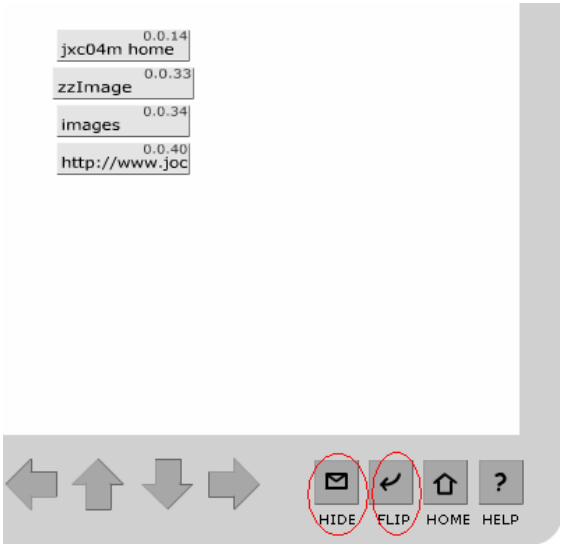
Once the screen has loaded you can either click the 'Connect' button to connect to the Mantra Server, or you can switch between views.
NB: You need to be connected to the server to proceed any further.



A.4. The ZigZag View

The images can be loaded as ZigZag cells by clicking on the 'Load Cells' button.

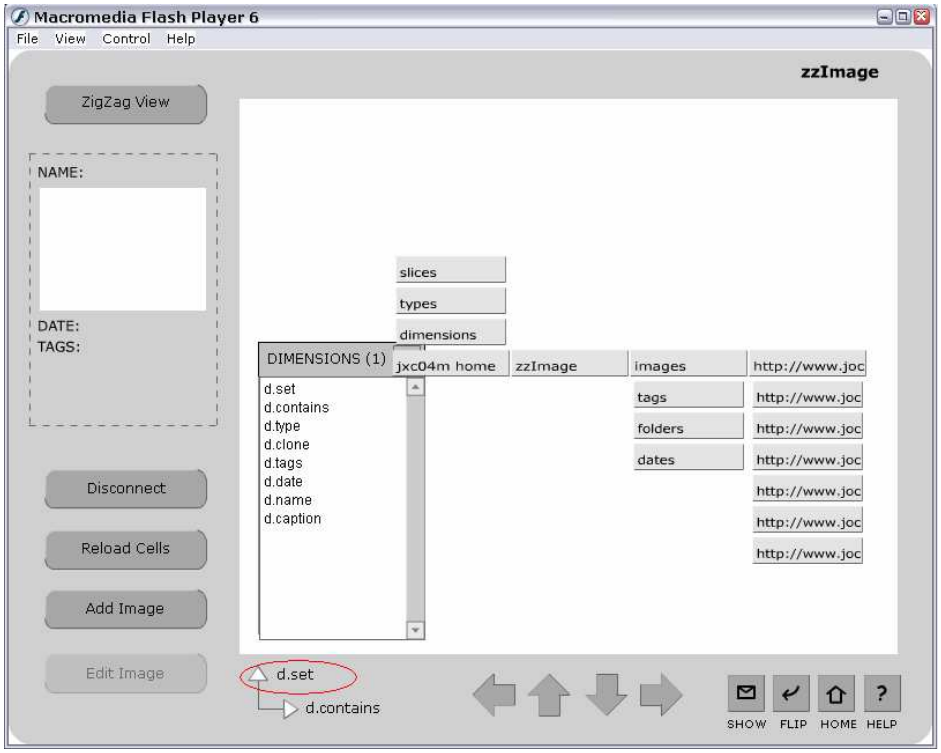




You can show or hide the cell's addresses by clicking on the envelope button 'Show/Hide'.

You can also flip the dimensions to view the structure in a different way.

You may also select different dimensions by opening the Dimension pop-up. You can do this either by clicking on the name of the dimension, or by the keyboard shortcut.

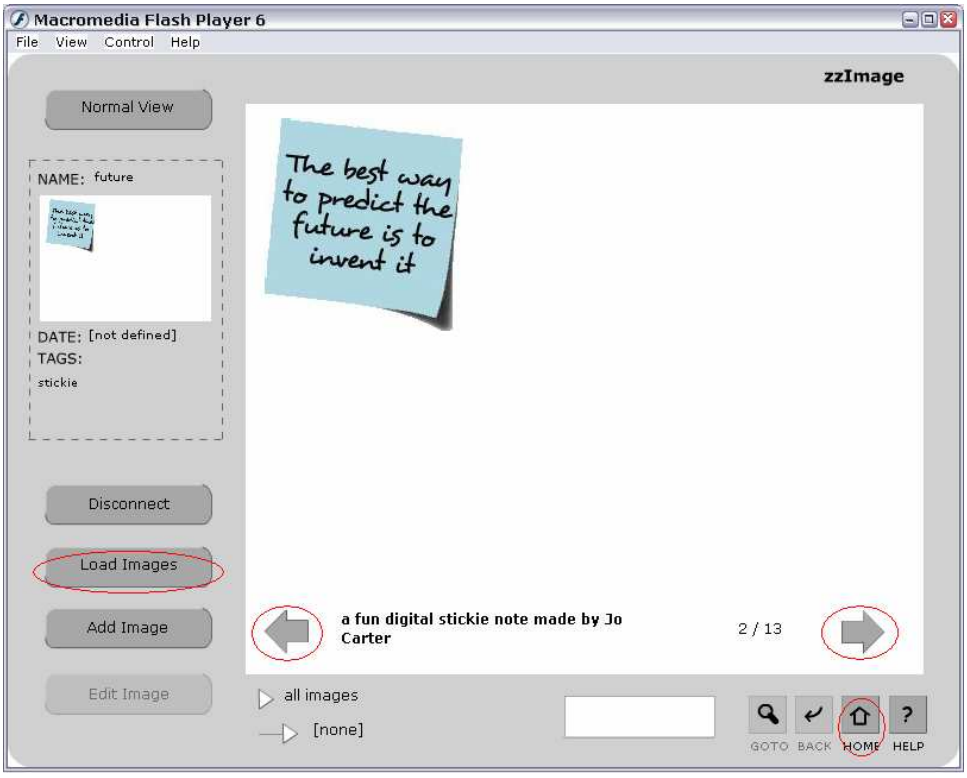


Once you have chosen a dimension you can confirm your choice by clicking on the close button (X), the dimension name, or by using the keyboard shortcut. The cells will then be redrawn to match your selection.

A.5. The Normal View

If you want to view your images in a different way, you can switch to the Normal View. You do this by clicking on the top left button (or by using the keyboard shortcut), this toggles between views.

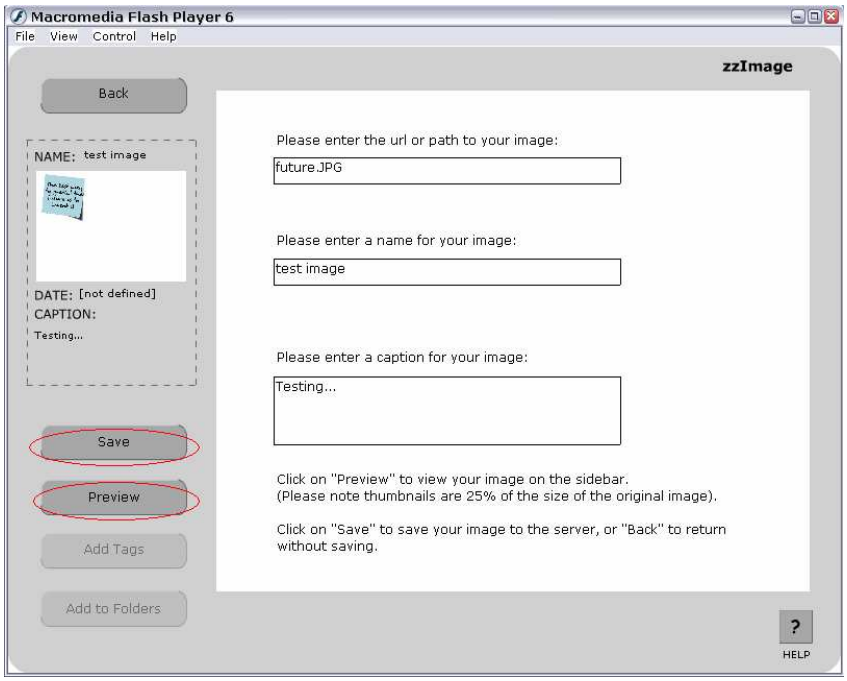
With the Normal View you can load your images into a slideshow by clicking on the 'Load Images' button.



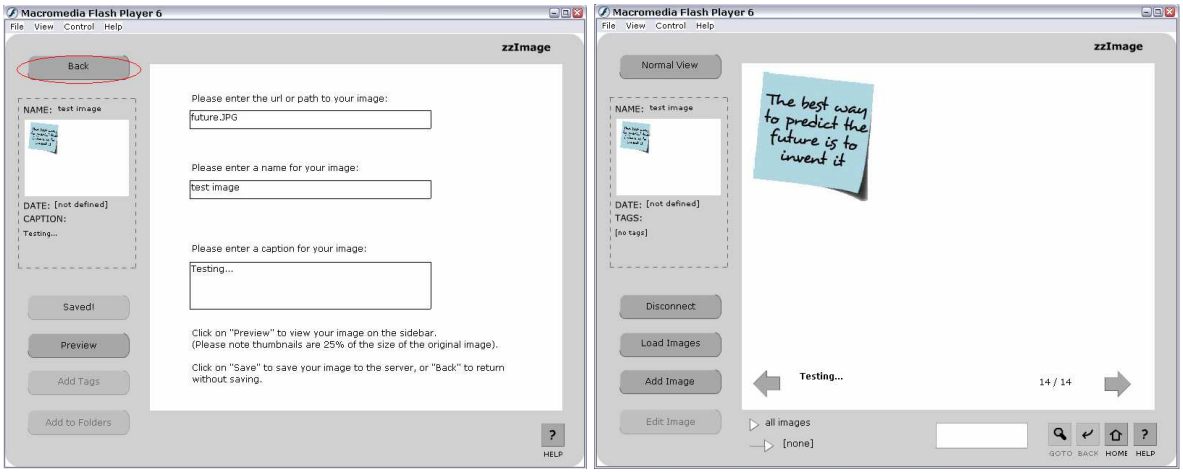
You can navigate through your images by using the arrow buttons or the arrow keys on your keyboard. If at any point you wish to return to the first image, you can do this by clicking on 'Home'.

A.6. Adding Images

Images can be added from either view, once you are connected to the server, by clicking on the 'Add Image' button. On this screen you can enter some of the details of your image.



You then can preview the image to check that you have the address correct, if it is correct then a thumbnail of your image will appear on the sidebar. Once you have confirmed all the details, you can 'Save' the image. You can go back to the view you were on at any point by clicking 'Back'. Please note that your image will not be saved if you do this.



Once your image has been saved, the saved button will grey out and display the text 'Saved!'. You can then go back to your previous view and re-load the images or cells. You will then be able to see the image you just added.

Appendix B: The CD

B.1. Files on the CD

B.1.1. Code files

zzImagev1.4.fla
zzImagev1.4b.fla

B.1.2. Flash Player files

zzImagev1.4.swf
zzImagev1.4b.swf (blue version)

B.1.3. Windows Projector files

zzImagev1.4.exe
zzImagev1.4b.exe

B.1.4. Other files

Image files for demonstration
Readme file
PDF User Manual
PDF version of Dissertation with Appendices

5.9. System Requirements

Flash Player 6+, or a Flash-enabled Internet Browser

Appendix C: Vision Questionnaire – zzImage

I am developing an image organiser client (you can see the interface I designed below) for my dissertation, which will connect to a "database" type server that is being researched at my university (This is not being designed for use on the Internet at this time). I am conscious of vision related issues that could apply to my interface design and wish to get the opinions of a wide range of people with relation to these issues.

The client is to be developed in Flash (because that was the easiest method of doing a prototype for me. The other options would have been C++ or Java.), and will be viewed via Macromedia's Flash Player. This does mean that it can be maximised to increase font size, etc. What I am interested in gathering would be the optimal colours, fonts and font sizes to use to accommodate the widest audience. In future versions, it could be expanded to include personal preferences, which would further help. Please also feel free to add any further comments via this entry, or email me.

Thank you very much for your time, it is most appreciated.

Note: All results will be kept anonymous, and no names will be mentioned in my write-up. Please try and answer all questions as honestly as possible.

Question 1: I have read and understood the above information and am willing to help

105 people answered the questionnaire

Question 2: How would you describe your vision?

Normal - 19
Normal with corrective spectacles – 57
Slightly reduced vision – 18
Low vision - 8
Partially sighted - 2
Blind - 1

Question 3: Which eye(s) does this affect?

Left eye only - 0
Left eye predominantly - 15
Right eye only - 1
Right eye predominantly - 15
Both eyes equally - 68

Question 4: Are you registered as legally blind?

Yes - 6
No - 91

Question 5: Do you have any colour vision problems?

No - 98
Yes, red colour-weak
Yes, red colour-blind
Yes, green colour-weak

Yes, green colour-blind
Yes, blue colour-weak - 1
Yes, blue colour-blind
Yes, total colour-blind - 2
Yes, but I don't know which type - 3

Question 6: Which colour combinations do you find make surfing the internet and using applications easier? Please indicate which colour is the text (t) and which colour is the background (b).

No preference - 3

Black on white - 54
Dark on light - 21
Black on light grey - 5
White on black - 20

Light on dark - 7
Black on purple - 4
Black on light - 4
Blue on white - 1
Bright on black - 1
Light on black - 2

Dark on white - 2
White on dark - 5
Red on black - 1
Black on blue - 3
Black on cream - 2

Number of people answered - 96

Comments:

ADG357-- (t) - Dark colour, preferably | (b) a lighter, contrasting colour. I rather, however, see a nicely designed/formatted/cleaned page with ugly colours than just an ugly page.
JMP582-- I find bright colours to draw my eyes, but not to be easy to read. Thus, bright colours work best for diagrams or illustrations. For text and background, I prefer moderate colours with very strong contrast. I do not have specific colour preferences.
CFK948 -- Black text on white background, or white text on black background. However, anything that is nicely contrasting works as well. Bright colours (canary yellow, cyan, etc.) do not work.

OJS474 -- I like it kept simple. I prefer plain black and white combinations, but it is interchangeable between text and background. In truth, colour is not as important to me as the light-to-dark contrast being clear.
ITD422 -- It is not particular colours that make it easier - it is the amount of contrast between text and background - high contrast is generally easier to read

Question 7: Which colour combinations do you find make surfing the internet and using applications more difficult? Please indicate which colour is the text (t) and which colour is the background (b).

No preference - 1
Light on bright - 2
Red on black - 10
Dark on dark - 14
Light on light - 15
Bright on bright - 19
Light on dark - 5
White on black - 9
Light on white - 4
Purple on pink - 2
Red on blue - 8
Bright on white - 10
Yellow on red - 3
Bright on black - 3
Red on green - 4
Red on yellow - 4
Blue on yellow - 2
Black on white - 2
Bright on light - 2
White on dark - 2
Red on white - 2
Dark on bright - 2

Number of people answered - 95

Comments:

JMP369 -- Bright yellows or greens (b); dark background with small, low contrast text, especially red (t) on black (b)
JMP471 -- Yellow and Red, Blue and Green...generally any light/light or dark/dark combination
SVY159 -- White background, any bright colours, and similar colours together (like dark blue text on a light blue background).

SVY372 -- White background is the worst. Black background is usually bad as any text that is clear enough to see is usually a bit burning.

SVY483 -- Pure white (b) give me headaches. Pure black (b) can also be problematic when paired with the wrong text colours. Yellow (t) on white (b). Bright green (t or b) and red (t or b), and any other t/b colour combinations that cause the text to "blink" when viewed.

SVY837 -- Any combination of green and yellow, blue and purple or anything else next to one another on the colour wheel.

BEH567 -- Black (b), white (t). I have a great deal of trouble with black or dark backgrounds on websites and applications and usually use the Firefox web developer toolbar to switch off colours if I come across a dark website.

CFK159 -- Usually anything lighter than black text makes it hard

LQU231 -- Grey. I have a hard time with Grey (B) / Black or White (T). I also have a hard time with bright backgrounds - i.e. orange or yellow

NRW789 -- Anything that is across the colour palette from each other. (i.e: red and blue, green and orange)

OTX654 -- Black backgrounds and most any colour of text makes it difficult for me; it seems like the text seems to "swim".

ZIQ975 -- If both the t and b are around the same light level (both dark colours, and both light colours) and also if the t and b are the same colour but different shades.

OJS474 -- Any colours other than black and white when involving text can be difficult, mostly because not enough people know how to make the contrast strong or clear enough. So black and white is a simple default to stick to.

UON287 -- Medium hues are harder to distinguish. Picture backgrounds are evil.

ANT184 -- Any non-contrasting colours. The one I see most often is blue/purple on blue/purple.

Question 8: Which is your preferred colour combination when you have the option to customise an interface/page as you wish? Please indicate which colour is the text (t) and which colour is the background (b).

No preference - 1

Black on yellow - 1
Black on pale blue - 7
Black on white - 39
White on dark - 5
White on black - 19
Black on purple - 5
Cream on brown - 1
Dark on light - 14
Black on light - 6
Green on black - 3
Light on dark - 6
White on bright - 1
Black on pink - 2
Green on white - 3
Light on black - 4
Blue on white - 2
Black on green - 1
Yellow on dark blue - 1
Bright on black - 1
Pink on brown - 1

Number of people answered - 98

Comments:

JMP582 -- I like muted purples, greens, and cream colours for backgrounds, with text either black, white, or a very dark colour to create contrast.
SVY372 -- I have had good luck with soft blues, greens, and reds as background with dark text on top of them.
MSC099 -- Backgrounds- light soft colours - e.g: cream, green; text- strong contrast - dark brown
BEH567 -- White (b), black (t). Really, though, I choose those colours only because they require the least thought. Another, more aesthetically pleasing combination would probably be even better, but my main priority is not to give myself a headache from reading.

Partially sighted people:

No preference - 1

Black on white - 8
Dark on light - 4
Black on purple - 1

White on black - 6
Light on black - 1
Black on light - 2
White on dark - 1
Pink on brown - 1

Light on dark - 3
Black on pink - 1
Green on white - 1

Legally blind people:

Black on white - 3
Dark on light - 1
White on black - 2

Colour blind people:

Dark on light - 1
White on black - 1
Light on black - 1
Black on white - 1

Question 9: Please choose from the fonts listed below, which you find to be the easiest to read or which you prefer. (Apologies if any of these are not available on the Mac – I only have a PC).

- Verdana - 39
- Arial - 27
- Times New Roman - 10
- Courier New - 1
- Tahoma - 9
- Comic Sans MS - 16

Partially sighted:	*Legally blind people:*	*Colour blind people:*
Verdana - 14	Verdana – 10	Verdana – 1
Arial - 6	Arial - 4	Arial - 4
Times New Roman - 2	Times New Roman – 0	Times New Roman - 0
Courier New - 0	Courier New - 0	Courier New - 0
Tahoma - 1	Tahoma - 0	Tahoma - 0
Comic Sans MS - 5	Comic Sans MS - 1	Comic Sans MS - 0

Question 10: At what font size? (The font used in the interfaces below for the buttons and text labels is size 12 Verdana.)

Size 10 - 21
Size 12 - 65
Size 14 - 12
Size 16 - 3
Size 18 or larger - 2

Partially sighted:	*Legally blind people:*	*Colour blind people:*
Size 10 - 4	Size 10 - 0	Size 10 - 1
Size 12 - 14	Size 12 – 2	Size 12 - 2
Size 14 - 6	Size 14 - 0	Size 14 – 1
Size 16 - 2	Size 16 - 1	Size 16 - 0
Size 18 or larger - 2	Size 18 or larger - 2	Size 18 or larger - 1

Question 11: Do you prefer buttons or key shortcuts?

Buttons (via the mouse) - 26

Keyboard shortcuts (especially if they are small buttons) - 9

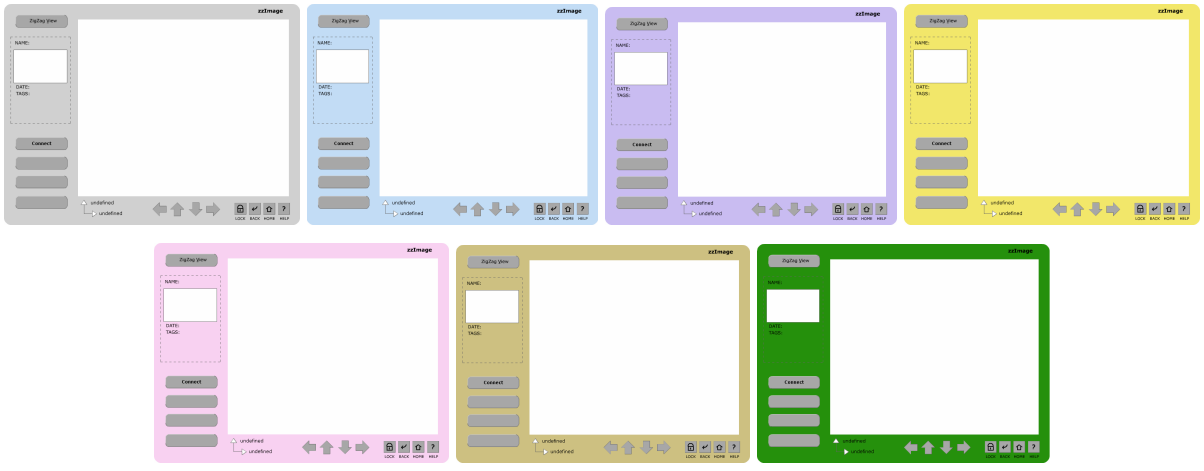
I use both - 55

I don't really mind - 15

Partially sighted:	*Legally blind people:*	*Colour blind people:*
Buttons - 5	Buttons - 1	Buttons - 1
Keyboard - 3	Keyboard - 2	Keyboard - 1
Both - 17	Both - 2	Both - 3
No preference - 4	No preference - 1	No preference - 1

Question 12: Out of the following interfaces, which to you is more usable (with regards to colour contrast, font size, font type, legibility)? (NB: Larger versions were supplied)

Note: I will change the button colours to a more appropriate shade of the background (as with the grey) on the other interfaces; these are just mock-ups.



Grey - 23
Blue - 38
Purple - 17
Yellow - 3
Pink - 12
Brown - 1
Green - 11

Partially sighted	*Legally blind people:*	*Colour blind people:*
Grey - 5	Grey - 1	Grey - 1
Blue - 7	Blue - 2	Blue - 2
Purple - 5	Purple – 1	Purple - 1
Yellow - 1	Yellow – 0	Yellow – 1
Pink - 4	Pink – 0	Pink – 0
Brown - 0	Brown - 0	Brown - 0
Green - 5	Green - 1	Green - 0

Question 13: Out of the previous interfaces, which did you prefer?

Grey - 20
Blue - 39
Purple - 22
Yellow - 0
Pink - 10
Brown - 6
Green - 7

Partially sighted:	*Legally blind people:*	*Colour blind people:*
Grey - 4	Blue – 2	Grey - 1
Blue - 6	Grey - 0	Blue - 2
Purple - 6	Purple – 1	Purple – 1
Yellow - 0	Yellow - 0	Yellow – 0
Pink - 6	Pink – 1	Pink - 0
Brown - 1	Brown - 0	Brown – 1
Green - 3	Green - 1	Green - 0

Question 14: Aside from the colours, what are your opinions of the above interface?

ADG135 -- Looks good, decent sized buttons

ADG357 -- I am a sucker for the MacLook (i.e brushed metallic, clean look)

ADG579 -- I would prefer the function buttons (below the major window) to be above it and to the left.

ADG681 -- I do not care about the colours, but the background/text on the buttons is not high enough contrast

ADG824 -- I think the navigation buttons should be at the top left instead of bottom right.

ADG924 -- Using Zoom text at a magnification of 1.5x, I could only read 2 of the words

JMP147 -- Use JAWS for screen reading...as long as alt-text is included I am fine

JMP369 -- Simple and attractive

JMP471 -- Very handy...though I do want to see it in action...I also want to see what the other buttons do...

MSC905-- Very simple, a little bland, but very easy to read, easy on the eyes, and easy to navigate

JMP582 -- The green is too dark to have black text on it. The yellow is too bright - I would not be able to focus on the window contents. The brown is simply an unappealing colour. The others are all workable, though the grey is a little alienating.

JMP693 -- Very good - navigation buttons in bottom right may be better suited at top left to fall inline with most common applications, and to be picky, maybe the 'ZigZag view' button could be dropped below the dashed square, creating a single group of buttons

SVY261 -- Simple and easy to use/understand.

SVY372 -- It is reasonably uncluttered.

SVY837 -- I thought the contrast between the button text and button could have been greater. Dark grey and black are not the best, in my opinion, for the visually impaired. I would suggest light grey and black.

BEH567 -- It looks nice and simple, uncluttered, clean, etc. :)

BEH912 -- They are neat and not too cluttered with buttons

CFK483 -- Easy to read, nice size buttons

CFK615 -- Rather plain, but easy to navigate

LQU342 -- The blue and yellow interfaces make nice contrasts

LQU453 -- I liked it, it looks easy enough to use and also has a visually attractive setup

MSC110 -- It is a nice interface, well put together!

LQU786 -- It is set up so everything is easily accessed.

LQU918 -- Pretty basic, but very easy to read.

NRW123 -- Very user friendly

NRW345 -- Easy to understand

NRW456 -- Clear, easy to read

NRW678 -- Nice and simple. I approve.

OTX876 -- Easy to use, clearly laid out.

OTX654 -- The grey has little contrast. The brown is a hideous shade, and the yellow is too bright. I liked the other shades. Although, with how my monitor is set, I like the green the best.

OTX219 -- Simplistic, looks very Bauhaus.

OTX198 -- It seems basic enough. I do not know what the buttons mean, but I doubt it would take much explanation. Are "tags" used to search images by keywords (i.e., a picture of a Monet would have "impressionism", etc as its tags)? That would be good.

ZIQ642 -- I like it, easy to understand

ZIQ429 -- Clear and concise; very simple and user-friendly

ZIQ297 -- I liked the font sizes and legibility

OJS474 -- It is simple and fairly straightforward. I like it, overall. Keep it simple is my biggest rule of thumb, personally. I am not seeing much difference in the interfaces except in colours so I am not sure I understand what you are asking, beyond colour.
OJS278 -- Blue and pink are the easiest to read/work with
UON287 -- I like how plain they are. I prefer plain interfaces to very 'artistic' and complicated ones.
UON465 -- I like how the font lies on the purple interface...it is nice and readable.
ITD422 -- The pastel shades are less harsh on the eyes than the bolder ones.
ANT184 -- The places with non-anti-aliased fonts ("Connect," "undefined") were jarring. (NB: Fixed the anti-aliasing)

Question 15: Do you have any other comments you would like to add? Do you have any questions you think I should have asked?

ADG357 -- Colours are what make a page unique; It is hard to say "I like $foo and $bar"; for me it depends on what the actual page looks like when put together.

I do not think your font size question is fair. With Tahoma (the font I usually set my browser to) I have 12pt for text areas but 10px for the actual display. Also, if I were to use Times New Roman, I'd set it at 12, so if you're looking for information to say "Okay 50 people picked Times New Roman" but "Oh everyone reads their pages at 14px" that's not fair unless you can say "Oh, but the 50 who picked Times New Roman view it at 10px". (NB: I have checked this).

JMP147 -- There are no questions about accessibility for totally blind people using screen readers

JMP582 -- For me the question is less /which/ colour than how colour is used.

I agree that font size is pretty arbitrary since each typeface varies. In addition, the question of typeface (font) is problematic for me because it begs the question, "For /what/?" I like Comic Sans, which I checked off, but only for informal, familiar kinds of writing, and not for large

blocks of text. For something to appear literate and professional, I think Time New Roman is best, although you have to keep an eye on it to make sure it is not too small. For more technical writing, one generally wants a sans-serif typeface: I like Arial better than Verdana. I do not know Tahoma at all and so could not compare.

About colour: although my vision isn't perfect, my main limitation when it comes to viewing images is not my actual eyesight but the fact that I have ADD and have difficulty focusing. That means that really bright colours need to be used sparingly for me: they are best suited to images which are important to the content of the page (because I /will/ look at them, whether I want to or not) but not for either text or backgrounds. Contrast is extremely important; otherwise my brain is convinced that my eyes cannot see properly.

In addition, I find colour-coding to be an extremely important tool. Having all things that belong in one category in one colour, and changing the colour of the background or the text or both to mark a change in subject matter is very helpful to me.

JMP693 -- Just to add on the other one, I would find it easier if direction buttons were put in 2 or 3 rows like on the keyboard arrows or numpad arrows, nice design!

SVY372 -- I just do not know fonts well enough to know which I like, but Sans Serif seems to be better, I think in general. Oh, and I have general decreased colour vision across the spectrum, but nothing specific. I mix up what colour things are sometimes.

SVY726 -- For clarification, I picked 10pt for email. On a webpage I would probably go a bit bigger.

BEH567 -- I actually prefer a font size of 11px. ;)

CFK726 -- White backgrounds make text difficult to read because it is like staring at a light bulb all the time.

LQU918 -- Large bold type is easy to read.

OTX198 -- The yellow interface was too bright; the brown one was ugly; the green one was too low-contrast. (Make the greys lighter and the green darker. A nice forest green is a good comfortable shade.)

OJS712 -- As far as serif/sans fonts and readability are concerned, it is a toss-up for me. On paper, serif is usually best. On screen, serif is sometimes best, but at small sizes, it can be worse than sans. It varies by font. Tahoma /Verdana are all right at reasonable sizes

I am the one (!) who picked the brown interface. I know it does not look great with the current grey buttons, but I happen to find the brown/black combination rather soothing to my eyes.

As far as fonts go, if I see something written in Comic Sans, I start conjuring up ideas of how terribly misguided and incompetent the person responsible for the use of the font must be. I am probably wrong, but that is what it makes me think. Someone once asked me what I had against it (she thought "it looks fun!"). I think it might have looked fun the first few times I saw it, before it became the universal "fun" font. Honestly, I simply have a hard time paying attention to things written in Comic Sans because I have seen it misused so many times, trying to bring about a sense of frivolity to situations where it really is not appropriate. You will never catch me using Comic Sans on anything.

I have some personal issues with Arial as well. At certain point sizes, it looks really, really nice. At others, it does not. I just typed up a test sentence in Arial and increased the font point by point. It does some really funky things between point sizes, to the point where at times it looks like a different typeface. That, and there is the issue with its history as a Helvetica knockoff...

OJS474 -- Colours/complex design should be minimal to keep the buttons, etc. more visible/in focus. Strong colours should be reserved for things like error messages. Simple & straightforward are key to user-friendliness!

ITD239 -- The only other question I would ask is why a particular colour combination or setup is preferred. For me, it's 'traditional' to see black on white: I am more comfortable with it. As to the issue of blue v. any other colour, it's 'cooler' on my eyes.

ANT184 -- Note that fonts in bitmaps appear as different sizes. I use 90 dpi at home and 75 dpi at work, but expect 10 pt. to be the same physical size on both.

I have slightly odd colour vision. I am not colour-blind, but my blue vision is unusual. As far as I can tell, and I have not tested this formally, my blue receptors respond to a very slightly higher frequency than normal. (Alternative hypothesis: my corneas are slightly more transmissive at higher frequencies than usual. I have not thought of a good way to test the difference.) In practical terms, here are several effects, none of them very noticeable:

There is a narrow band of colours that others see as blue-green that I see as grey. It is a very small range of colours, but I had a closet in a dorm room that was painted this shade. It annoyed me considerably. Occasionally I will see very faint purple patterns in flowers that others see as pure white.

A friend of mine was experimenting with fluorescent glazes. He was showing samples of them under a long-wave UV light. While everyone could see the light itself as purple (cheap long-wave filters are never very sharp), I saw its non-specular reflection in the ceramic substrate he was testing the glazes on as a very bright royal purple, where everyone else saw a faint reddish purple. This was the clear, repeatable effect that convinced me I was not imagining or exaggerating things.

The effect is inherited; my mother sees things the same way.

Appendix D: Code Listing

D.1. Actionscript from Actions Layer, Frame 1, Scene 1 (Preloader)

```
var bytesloaded = Number(getBytesLoaded());
var bytestotal = Number(getBytesTotal());

if (bytesloaded >= bytestotal)
{
      gotoAndStop(6);
}

var per = bytesloaded/bytestotal*100;
setProperty("_root.bar", _xscale, per);
var percent_t = math.round(per)+"%";
```

D.2. Actionscript from Connection Layer, Scene 2

```
// The following script is adapted for Flash MX and zzImage by Jo Carter
// From comms_engine.as, query.as and query_stack.as (Flash MX 2004)
// Original code copyright (c) 2005, James Goulding, Chris Coleman
// Licensed under the Apache License, version 2.0
//           http://www.apache.org/licenses/LICENSE-2.0

var query_counter = 0;
var query_store;
var connected = false;
```

```
var connect_msg;
var is_connected;
var normal = false;


// Ensure that connection status is passed between pages
if (is_connected == true)
{
      connected = true;
}

// Query class
function query(_query_id,
                    _function_name,
                    _param)
{
      this.query_id = _query_id;
      this.function_name = _function_name;
      this.param = _param;
}

// Query stack class
function query_stack()
{
      trace("creating query stack");
      this.queries = new Array();
}

// Add a query to the stack
query_stack.prototype.add_entry = function(query_id,
                                           function_name,
                                           param)
{
      this.queries[this.queries.length] = new query(query_id,
                                             function_name, param);
};
```

```
// Remove an entry
query_stack.prototype.remove_entry = function(query_id)
{
	for (var i = 0; i<this.queries.length; i++)
	{
		if (this.queries[i].query_id == query_id)
		{
			delete this.queries[i];
			this.queries.splice(i, 1);
			break;
		}
	}
};

// Check whether the query stack is empty
query_stack.prototype.is_empty = function()
{
	if (this.queries.length == 0)
	{
		return true;
	}
	else
	{
		return false;
	}
};

// Find a query in the stack
query_stack.prototype.find_entry = function(query_id)
{
	for (var i = 0; i<this.queries.length; i++)
	{
		if (this.queries[i].query_id == query_id)
		{
			return this.queries[i];
			break;
```

```
            }
      }
      return null;
};

////////////////////////////////////////////////
var xsock;

// This script connects to the mantra server
function server_connect()
{
      // Depending on whether connecting from campus, or via a HTTP tunnel
      //var ip = "128.243.20.74";
      //var port = "8109";
      var ip = "127.0.0.1";
      var port = "4444";

      xsock = new XMLSocket();
      trace("---> connecting to mantra server at ["+ip+"] on port
                                                      ["+port+"]...");
      xsock.onData = process_xml_message;
      xsock.connect(ip, port);

      query_store = new query_stack();
      connected = true;
      is_connected = true;
      connect_msg = "Disconnect";
}

// Disconnects from server
function server_disconnect()
{
      trace("system: disconnecting from mantra server");
      xsock.close();

      connected = false;
```

```
	is_connected = false;
	connect_msg = "Connect";
}

// When data is received from the server, it must be processed
// The data received should be a <response>
function process_xml_message(msg)
{
	var xml_doc = new XML(msg);
	var node = xml_doc.firstChild.firstchild.nodeName;

	if (node == "response")
	{
		process_query_response(xml_doc);
	}
	else
	{
		process_unknown_response(xml_doc);
	}
}

// Processes message received from server, finds the function that needs
// the query, and executes it.
function process_query_response(xml_doc)
{
	var response_id = xml_doc.firstChild.firstChild.attributes.id;
	trace("message received from server: ["+xml_doc+"]...");
	var query_match = query_store.find_entry(response_id);

	if (query_match == null)
	{
		// response not in query stack - we have a problem!
		trace("error: matching query for response cannot be found");
	}
	else
	{
```

```
            query_store.remove_entry(response_id);
            // execute the command it is for
            query_match.function_name(xml_doc, query_match.param);
      }
}

function process_unknown_response(xml_doc)
{
      trace("error: received message is unrecognisable.");
      trace(xml_doc);
}

// Submits query and returns query id
function submit_query(query, function_name, param)
{
      var query_id = query_counter;
      query_store.add_entry(query_id, function_name, param);
      send_query(query, query_id);
      query_counter++;
}

// This function sends a query to the mantra server, and creates the XML
function send_query(query, query_id)
{
      var xml_doc = new XML();

      var message_tag = xml_doc.createElement("mantra");
      var query_tag = xml_doc.createElement("query");
      var content_tag = xml_doc.createTextNode(query);

      query_tag.attributes.id = query_id;
      query_tag.appendChild(content_tag);
      message_tag.appendChild(query_tag);
      xml_doc.appendChild(message_tag);

      trace("---> message sent ["+xml_doc+"]");
```

```
	xsock.send(xml_doc);
}
```

D.3. Actionscript from Actions Layer, Frame 1 (ZigZag View), Scene 2

```
var dim_up;
var dim_across;
var node_row;
var current_nodes;
var cursor;
var show_addresses;
var refresh_interval;
var current_cell;

// Grey out buttons
if (connected == true)
{
	enable_buttons();
}
else
{
	disable_buttons();
}

function enable_buttons()
{
	load_cells._alpha = 100;
	load_text.textColor = 0x000000;
	load_cells.enabled = true;
	add_image._alpha = 100;
	add_text.textColor = 0x000000;
	add_image.enabled = true;
}
```

```
// Enable the other buttons
function enable_2buttons()
{
      home._alpha = 100;
      home_text.textColor = 0x000000;
      home.enabled = true;
      show._alpha = 100;
      show_text.textColor = 0x000000;
      show.enabled = true;
      flip._alpha = 100;
      flip_text.textColor = 0x000000;
      flip.enabled = true;
}

function disable_buttons()
{
      load_cells._alpha = 40;
      load_text.textColor = 0x6C6C6C;
      load_cells.enabled = false;
      home._alpha = 40;
      home_text.textColor = 0x6C6C6C;
      home.enabled = false;
      show._alpha = 40;
      show_text.textColor = 0x6C6C6C;
      show.enabled = false;
      flip._alpha = 40;
      flip_text.textColor = 0x6C6C6C;
      flip.enabled = false;
      add_image._alpha = 40;
      add_text.textColor = 0x6C6C6C;
      add_image.enabled = false;
}

// This functionality has not been added yet, so these stay disabled.
edit_image._alpha = 40;
edit_text.textColor = 0x6C6C6C;
```

```
edit_image.enabled = false;

stop();

// Connect to server
// The main functions specified here are called by the various buttons
function connect_to_server()
{
      if (connected == false)
      {
            server_connect();
            enable_buttons();
      }
      else
      {
            server_disconnect();
            clear_layout();
            load_text.text = "Load Cells";
            disable_buttons();
      }
}


// The following code has been modified for use in Flash MX by Jo Carter
// From view_nodes.as, mantra_node.as, controller.as and model_nodes.as
// (Flash MX 2004)
// Original code copyright (c) 2005 James Goulding, Chris Coleman
// Licensed under the Apache License, version 2.0
//                            http://www.apache.org/license/LICENSE-2.0
// mantra_node.as

function mantra_node(_nri,
                              _content,
                              _target_xpos,
                              _target_ypos,
                              _target_height,
```

```
                                    _target_width)
{
      //trace("creating mantra node for nri " + _nri);
      this.nri = _nri;
      this.content = _content;
      this.target_xpos = _target_xpos;
      this.target_ypos = _target_ypos;
      this.target_height = _target_height;
      this.target_width = _target_width;
      this.mc = new MovieClip();
      this.current_xpos = 0;
      this.current_ypos = 0;
      this.current_height = 0;
      this.current_width = 0;
}

mantra_node.prototype.activate = function()
{
      mc.sensor.onPress = sensor_on_release;
      mc.sensor.onRollOver = sensor_on_rollover;
      mc.sensor.onRollOut = sensor_on_rollout;
      mc.sensor.enabled = true;
};


// mantra_nodes.as

function initialise()
{
      trace("initialise");
      dim_across = "d.contains";
      dim_up = "d.set";
      // address of "zzImage"
      cursor = "0.0.33";
}
```

```
function set_cursor(_cursor)
{
      trace("set cursor");
      cursor = _cursor.nri;
}

function construct_layout()
{
      var date = new Date();
      var milli = date.getMilliseconds();
      trace("constructing layout at " + milli);

      clear_nodes();
      node_row = new Array(7);
      current_nodes = new Array();

      var query = cursor+" | fetch this to +3 along \""+dim_across+"\" |
                                                                  resolve";
      submit_query(query, process_right);
      query = cursor+" | fetch -1 to -3 along \""+dim_across+"\" |
                                                                  resolve";
      submit_query(query, process_left);
}

function clear_layout()
{
      trace("clearing layout");

      clear_nodes();
      current_nodes = new Array();

      dim_up = "undefined";
      dim_across = "undefined";
}
```

```
function check_completion()
{
	if (query_store.is_empty() == true)
	{
		update(current_nodes);
	}
}

function process_right(xml_content)
{
	trace("process right");

	// Get array of returned nodes
	var nodes = xml_content.firstChild.firstChild.childNodes;

	// If the specified range does not exist then it cannot be processed
	if (nodes[0].nodeName == "error" || nodes[0].nodeName == "warning")
	{
		return;
	}

	for (var i=0; i<nodes.length; i++)
	{
		var nri = nodes[i].attributes.nri;
		var content = nodes[i].firstChild;

		if (!content)
		{
			content = "";
		}

		current_nodes.push(new mantra_node(nri, content, (i*103), 0,
                                                                25, 102));
```

```
            // Query for down rank
            var query = nri+" | fetch next to +6 along \""+dim_up+"\" |
                                                                  resolve";
            submit_query(query, process_down, i);

            // Query for up rank
            query = nri+" | fetch prev to -6 along \""+dim_up+"\" |
                                                                  resolve";
            submit_query(query, process_up, i);
      }
}

function process_left(xml_content)
{
      trace("process left");

      var nodes = xml_content.firstChild.firstChild.childNodes;

      if (nodes[0].nodeName == "error" || nodes[0].nodeName == "warning")
      {
            return;
      }

      for (var i=0; i<nodes.length; i++)
      {
            var nri = nodes[i].attributes.nri;
            var content = nodes[i].firstChild;

            if (!content)
            {
                  content = "";
            }

            current_nodes.push(new mantra_node(nri, content, -((i+1)*103),
                                                      0, 25, 102));
```

```
            // Query for down rank
            var query = nri+" | fetch next to +6 along \""+dim_up+"\" |
                                                            resolve";
            submit_query(query, process_down, -(i+1));

            // Query for up rank
            query = nri+" | fetch prev to -6 along \""+dim_up+"\" |
                                                            resolve";
            submit_query(query, process_up, -(i+1));
      }
}

function process_up(xml_content, index)
{
      trace("process up, index " + index);

      var nodes = xml_content.firstChild.firstChild.childNodes;

      if (nodes[0].nodeName == "error" || nodes[0].nodeName == "warning")
      {
            check_completion()
            return;
      }

      for (var i=0; i<nodes.length; i++)
      {
            var nri = nodes[i].attributes.nri;
            var content = nodes[i].firstChild;

            if (!content)
            {
                  content = "";
            }

            current_nodes.push(new mantra_node(nri, content, (index*103),
                                                -((i+1)*27), 25, 96));
```

```
	}

	check_completion()
}

function process_down(xml_content, index)
{
	trace("process down, index " + index);

	var nodes = xml_content.firstChild.firstChild.childNodes;

	if (nodes[0].nodeName == "error" || nodes[0].nodeName == "warning")
	{
		check_completion()
		return;
	}

	for (var i=0; i<nodes.length; i++)
	{
		var nri = nodes[i].attributes.nri;
		var content = nodes[i].firstChild;

		if (!content)
		{
			content = "";
		}

		current_nodes.push(new mantra_node(nri, content, (index*103),
							((i+1)*27), 25, 96));
	}

	check_completion()
}
```

```
// controller.as

function initialise_display()
{
	trace("initialise display");

	show_addresses = false;
	show_text.text = "SHOW";
	load_text.text = "Reload Cells";
	enable_2buttons();

	initialise();
	construct_layout();
}

function node_on_press(new_cursor)
{
	trace("node on press");

	set_cursor(new_cursor);
	construct_layout();
}

function home_on_press()
{
	trace("home on press");
	// zzImage address
	cursor = "0.0.33";

	construct_layout();
}

function flip_dims()
{
	trace("flipping dims");
```

```
	var temp = dim_up;
	dim_up = dim_across;
	dim_across = temp;

	construct_layout();
}


// View_nodes.as

// Find the node in the array
function match_node(nri)
{
	for (var i=0; i<current_nodes.length; i++)
	{
		if (current_nodes[i].nri == nri)
		{
			return i;
		}
	}
	return -1;
}
function update_focus_node(focus_node)
{
	current_cell = focus_node.nri;
	var query = current_cell + " | fetch all along \"d.type\" |
                                                                resolve";
	_root.submit_query(query, checkType);
}

function checkType(xml_doc)
{
	xmlNode = xml_doc.firstChild.firstChild.childNodes;

	if (xmlNode[1].firstChild == "t.image")
	{
```

```
            thumbnail2.loadMovie(xmlNode[0].firstChild);
            thumbnail2._xscale = 25;
            thumbnail2._yscale = 25;

            var query = current_cell+' | fetch +1 along \"d.name\" |
                                              resolve';
            _root.submit_query(query, showName);
            query = current_cell+' | fetch +1 along \"d.date\" | resolve';
            _root.submit_query(query, showDate);
            query = current_cell+' | fetch +1 along \"d.tags\" | fetch all
                                              along \"d.set\" | resolve';
            _root.submit_query(query, showTags);
      }
}

function showName(xml_doc)
{
      xmlNode2 = xml_doc.firstChild.firstChild.childNodes;

      if (xmlNode2[0].attributes.nri == current_cell)
      {
            image_name.text = "[no name]";
      }
      else
      {
            image_name.text = xmlNode2[0].firstChild;
      }
}

function showDate(xml_doc)
{
      xmlNode3 = xml_doc.firstChild.firstChild.childNodes;

      if (xmlNode3[0].attributes.nri == current_cell)
      {
            image_date.text = "[not defined]";
```

```
	}
	else
	{
		image_date.text = xmlNode3[0].firstChild;
	}
}

function showTags(xml_doc)
{
	xmlNode4 = xml_doc.firstChild.firstChild.childNodes;
	var no_tags = false;
	var _total = xmlNode4.length;

	for (var i=0; i<_total; i++)
	{
		if (xmlNode4[i].attributes.nri == current_cell)
		{
			no_tags = true;
		}
	}

	if (no_tags == true)
	{
		image_tags.text = "[no tags]";
	}
	else
	{
		var tags;
		tags = xmlNode4[0].firstChild;

		for (var i = 1; i<_total; i++)
		{
			tags = tags+", "+xmlNode4[i].firstChild;
		}

		image_tags.text = tags;
```

```
	}
}

function clear_nodes()
{
	trace("clearing nodes");
	for (var i=0; i<current_nodes.length; i++)
	{
		// Need to set alpha to 0, because otherwise nodes will not
		disappear // off the screen

		current_nodes[i].mc._alpha = 0;
		nodes_mc.removeMovieClip(current_nodes[i].mc);
	}
}

function refresh_nodes()
{
	var finished = 1;

	for (var i=0; i<current_nodes.length; i++)
	{
		var dif_xpos = current_nodes[i].target_xpos -
                                                current_nodes[i].mc._x;
		var dif_ypos = current_nodes[i].target_ypos -
                                                current_nodes[i].mc._y;
		var dif_height = current_nodes[i].target_height
                                        - current_nodes[i].mc._height;
		var dif_width = current_nodes[i].target_width
                                        - current_nodes[i].mc._width;

		// Update the node's x position on screen
		if (Math.abs(dif_xpos) > 2)
		{
			finished = 0;
			current_nodes[i].mc._x += (dif_xpos/4);
```

```
}
else
{
      current_nodes[i].mc._x = current_nodes[i].target_xpos;
}

// Update the node's y position on screen
if (Math.abs(dif_ypos) > 2)
{
      finished = 0;
      current_nodes[i].mc._y += (dif_ypos/4);
}
else
{
      current_nodes[i].mc._y = current_nodes[i].target_ypos;
}

// Update the node's width on screen
if (Math.abs(dif_width) > 2)
{
      finished = 0;
      current_nodes[i].mc._width += (dif_width/4);
}
else
{
      current_nodes[i].mc._width =
                              current_nodes[i].target_width;
}

// Update the node's height on screen
if (Math.abs(dif_height) > 2)
{
      finished = 0;
      current_nodes[i].mc._height += (dif_height/4);
}
else
```

```
            {
                  current_nodes[i].mc._height =
                                          current_nodes[i].target_height;
            }
      }

      if (finished)
      {
            clearInterval(refresh_interval);
      }
}

function update(update_nodes)
{
      trace("update nodes");
      clear_nodes();

      for (var i=0; i<update_nodes.length; i++)
      {
            // Find the nri in the current nodes
            var match = match_node(update_nodes[i].nri);

            // If exists then:
            if (match >= 0)
            {
                  // Copy ALL current info across apart from the NRI which
                  // is obviously the same and the content which may have
                  // changed
                  update_nodes[i].current_xpos =
                                    current_nodes[match].target_xpos;
                  update_nodes[i].current_ypos =
                                    current_nodes[match].target_ypos;
                  update_nodes[i].current_height =
                                    current_nodes[match].target_height;
                  update_nodes[i].current_width =
                                    current_nodes[match].target_width;
```

```
}
// If doesn't exist
else
{
      // Set node's current info to target
      update_nodes[i].current_xpos =
                              update_nodes[i].target_xpos;
      update_nodes[i].current_ypos =
                              update_nodes[i].target_ypos;
      update_nodes[i].current_height =
                              update_nodes[i].target_height;
      update_nodes[i].current_width =
                              update_nodes[i].target_width;
}

// Create MovieClip, and position appropriately according to
// information that just set in previous statements
update_nodes[i].mc = nodes_mc.attachMovie("node_container",
                                          "node_container_"+i,
                                          (30+i));

update_nodes[i].mc._x = update_nodes[i].current_xpos;
update_nodes[i].mc._y = update_nodes[i].current_ypos;
update_nodes[i].mc._height = update_nodes[i].current_height;
update_nodes[i].mc._width = update_nodes[i].current_width;
update_nodes[i].mc.node_content = update_nodes[i].content;

if (show_addresses)
{
      update_nodes[i].mc.node_nri = update_nodes[i].nri;
}
else
{
      update_nodes[i].mc.node_nri = "";
}
```

```
            update_nodes[i].activate();
      }

      current_nodes = new Array;
      current_nodes = update_nodes;

      var date = new Date();
      var milli = date.getMilliseconds();
      trace("layout constructed at " + milli);

      refresh_interval = setInterval(refresh_nodes, 40);
}

function switch_addresses_show()
{
      if (show_addresses)
      {
            show_addresses = false;
            show_text.text = "SHOW";
            addresses_hide();
      }
      else
      {
            show_addresses = true;
            addresses_show();
            show_text.text = "HIDE";
      }
}

function addresses_show()
{
      trace("show addresses");

      for (var i=0; i<current_nodes.length; i++)
      {
            current_nodes[i].mc.node_nri = current_nodes[i].nri;
```

```
	}
}

function addresses_hide()
{
	trace("hide addresses");

	for (var i=0; i<current_nodes.length; i++)
	{
		current_nodes[i].mc.node_nri = "";
	}
}

function sensor_on_press(nri)
{
	trace("sensor on press");
	node_on_press(nri);
}

function sensor_on_rollover(nri)
{
	trace("sensor on rollover");
	update_focus_node(nri);
}
```

D.4. Actionscript from Dim Pop-ups, Frame 1, Scene 2

D.4.1. Methods Layer

```
function populateDims(xml_doc)
{
	var xmlNode = xml_doc.firstChild.firstChild.childNodes;
	var total = xmlNode.length;
```

```
      for (var i = 0; i<total; i++)
      {
            dimScroll.addItem(xmlNode[i].firstChild);
      }
}
```

D.4.2. Actions Layer

```
popupTitle.text = "DIMENSIONS (1)";
var dim_query = "gather dims | resolve";
_root.submit_query(dim_query, populateDims);

stop();
```

D.4.3. Close Button

```
on (release, keyPress "1")
{
      _root.dim_up = dimScroll.getSelectedItem().label;
      _root.construct_layout();
      gotoAndStop(1);
}
```

D.5. Actionscript from Action Layer, Frame 2 (Normal View), Scene 2

```
var current_cell;
var p;
var total;
var image;
var description;
```

```
// Grey out unused buttons
if (connected == true)
{
      enable_buttons();
}
else
{
      disable_buttons();
}

function enable_buttons()
{
      load_button._alpha = 100;
      load_images.textColor = 0x000000;
      load_button.enabled = true;
      add_image._alpha = 100;
      add_text.textColor = 0x000000;
      add_image.enabled = true;
}

function disable_buttons()
{
      load_button._alpha = 40;
      load_images.textColor = 0x6C6C6C;
      load_button.enabled = false;
      home._alpha = 40;
      home_text.textColor = 0x6C6C6C;
      home.enabled = false;
      add_image._alpha = 40;
      add_text.textColor = 0x6C6C6C;
      add_image.enabled = false;
}

// This functionality has not been added yet, so these stay disabled.
edit_image._alpha = 40;
edit_text.textColor = 0x6C6C6C;
```

```
edit_image.enabled = false;
goto._alpha = 40;
goto_text.textColor = 0x6C6C6C;
goto.enabled = false;
back._alpha = 40;
back_text.textColor = 0x6C6C6C;
back.enabled = false;

stop();

// Connect to server
// The buttons call the methods specified in this code
function connect_to_server()
{
      if (connected == false)
      {
            server_connect();
            enable_buttons();
      }
      else
      {
            server_disconnect();
            load_images.text = "Load Images";
            disable_buttons();
      }
}

// Load images
function load_images()
{
      var dim_query = '0.0.34 | fetch tail along \"d.contains\" | fetch
                                          all along \"d.set\" | resolve';
      _root.submit_query(dim_query, createSlideshow);
      p = 0;
      load_images.text = "Reload Images";
```

```
	// Enable other button
	home._alpha = 100;
	home_text.textColor = 0x000000;
	home.enabled = true;
}

// Create the slideshow
function createSlideshow(xml_doc)
{
	xmlNode = xml_doc.firstChild.firstChild.childNodes;
	image = [];
	description = [];
	total = xmlNode.length;

	for (var i = 0; i<total; i++)
	{
		image[i] = xmlNode[i].firstChild;
		description[i] = xmlNode[i].attributes.nri;
	}

	firstImage();
}

// Populate the sidebar
function showName(xml_doc)
{
	xmlNode2 = xml_doc.firstChild.firstChild.childNodes;

	if (xmlNode2[0].attributes.nri == current_cell)
	{
		image_name.text = "[no name]";
	}
	else
	{
		image_name.text = xmlNode2[0].firstChild;
	}
```

```
	var dim_query = current_cell+' | fetch +1 along \"d.date\" |
                                                                resolve';
	_root.submit_query(dim_query, showDate);
}

function showDate(xml_doc)
{
	xmlNode3 = xml_doc.firstChild.firstChild.childNodes;

	if (xmlNode3[0].attributes.nri == current_cell)
	{
		image_date.text = "[not defined]";
	}
	else
	{
		image_date.text = xmlNode3[0].firstChild;
	}

	var dim_query = current_cell+' | fetch +1 along \"d.caption\" |
                                                                resolve';
	_root.submit_query(dim_query, showCaption);
}

function showCaption(xml_doc)
{
	xmlNode4 = xml_doc.firstChild.firstChild.childNodes;

	if (xmlNode4[0].attributes.nri == current_cell)
	{
		desc_txt.text = "[no caption]";
	}
	else
	{
		desc_txt.text = xmlNode4[0].firstChild;
	}
```

```
	var dim_query = current_cell+' | fetch +1 along \"d.tags\" | fetch
                                        all along \"d.set\" | resolve';
	_root.submit_query(dim_query, showTags);
}

function showTags(xml_doc)
{
	xmlNode5 = xml_doc.firstChild.firstChild.childNodes;
	var no_tags = false;
	var _total = xmlNode5.length;

	for (var i = 0; i<_total; i++)
	{
		if (xmlNode5[i].attributes.nri == current_cell)
		{
			no_tags = true;
		}
	}

	if (no_tags == true)
	{
		image_tags.text = "[no tags]";
	}
	else
	{
		var tags = xmlNode5[0].firstChild;

		for (var i=1; i<_total; i++)
		{
			tags = tags+", "+xmlNode5[i].firstChild;
		}

		image_tags.text = tags;
	}
}
```

```
// Display images/ navigation etc
this.onEnterFrame = function()
{
	filesize = picture.getBytesTotal();
	loaded = picture.getBytesLoaded();
	preloader._visible = true;

	if (loaded != filesize)
	{
		setProperty("_root.preloader", _xscale, 100*loaded/filesize);
	}
	else
	{
		preloader._visible = false;

		if (picture._alpha<100)
		{
			picture._alpha += 10;
		}
	}
};

function nextImage()
{
	// Loop through array infinitely
	if (p<(total-1))
	{
		p++;
	}
	else
	{
		p = (p+1-total);
	}

	if (loaded == filesize)
	{
```

```
            on_focus(description[p]);
      }
}

function prevImage()
{
      if (p>0)
      {
            p--;
      }
      else
      {
            p = (p-1+total);
      }

      if (loaded == filesize)
      {
            on_focus(description[p]);
      }
}

function firstImage()
{
      if (loaded == filesize)
      {
            on_focus(description[0]);
      }
}

function picture_num()
{
      current_pos = p+1;
      pos_txt.text = current_pos+" / "+total;
}
```

```
function on_focus(cell_address)
{
      current_cell = cell_address;

      picture._alpha = 0;
      unloadMovie(picture);
      loadMovie(image[p], picture);
      picture_num();

      unloadMovie(thumbnail);
      loadMovie(image[p], thumbnail);
      thumbnail._xscale = 25;
      thumbnail._yscale = 25;

      var dim_query = current_cell+' | fetch +1 along \"d.name\" |
                                                                  resolve';
      _root.submit_query(dim_query, showName);
}
```

D.6. Actionscript from Actions Layer, Frame 3 (Add Image), Scene 2

```
var image_nri;

// Disable buttons not currently using
add_tags._alpha = 40;
tag_text.textColor = 0x6C6C6C;
add_tags.enabled = false;
add_folders._alpha = 40;
folder_text.textColor = 0x6C6C6C;
add_folders.enabled = false;

stop();
```

```
// Preview image
function preview()
{
      image_name.text = name.text;
      image_caption.text = caption.text;
      image_date.text = "[not defined]";

      _root.thumbnail3.loadMovie(image_url.text);
      _root.thumbnail3._xscale = 25;
      _root.thumbnail3._yscale = 25;
}

// Save Image
function saveimage()
{
      var image = image_url.text;

      var query = "0.0.34 | fetch +1 along \"d.contains\" | fetch tail
                                          along \"d.set\" | ";
      query = query + "create + along \"d.set\" | change content = \"" +
                                          image + "\"";
      query = query + " | save slice 0";
      submit_query(query, savetype);
}

function savetype(xml_content)
{
      var nodes = xml_content.firstChild.firstChild.childNodes;

      // If there is no error, then continue saving
      if (nodes[0].nodeName != "error" && nodes[0].nodeName != "warning")
      {
                  image_nri = nodes[0].attributes.nri;
                  var query = image_nri + " | create + along \"d.type\" |
                                    insert + 0.0.38 along \"d.clone\"";
                  query = query + " | save slice 0";
```

```
                submit_query(query, savename);
        }
}

function savename(xml_content)
{
        var iname = name.text;
        var nodes = xml_content.firstChild.firstChild.childNodes;

        if (nodes[0].nodeName != "error" && nodes[0].nodeName != "warning")
        {
                var query = image_nri + " | create + along \"d.name\" | change
                                        content = \"";
                query = query + iname + "\" | create + along \"d.type\" |
                                        insert + 0.0.57 along \"d.clone\"";
                query = query + " | save slice 0";
                submit_query(query, savecaption);
        }
}

function savecaption(xml_content)
{
        var icaption = caption.text;
        var nodes = xml_content.firstChild.firstChild.childNodes;

        if (nodes[0].nodeName != "error" && nodes[0].nodeName != "warning")
        {
                var query = image_nri + " | create + along \"d.caption\" |
                                        change content = \"";
                query = query + icaption + "\" | create + along \"d.type\" |
                                        insert + 0.0.57 along \"d.clone\"";
                query = query + " | save slice 0";
                submit_query(query, saved);
        }
}
```

```
function saved(xml_content)
{
	var nodes = xml_content.firstChild.firstChild.childNodes;

	if (nodes[0].nodeName != "error" && nodes[0].nodeName != "warning")
	{
		save_text.text = "Saved!";
		save._alpha = 40;
		save.enabled = false;
	}
}
```