Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

The development of a Java based GIS viewing tool

A thesis presented in partial fulfilment of the requirements for the degree of

Masters of Technology

in Information Engineering at

Massey University

James Maguire

1999

Abstract

Geographic Information Systems (GIS) industry sources quote the ratio of power users to casual users at 1000:1, within New Zealand this figure has been found to be 30:1. The casual user is often under-supported, with slow and cumbersome viewing tools. This project implements a full data download system in Java for use with Genasys (New Zealand) GIS software.

Three components were developed; a vector data handler, an image download system, and a database client. These components were integrated to form a powerful client that offered a significant performance increase over the "server based" client. The image download system outperformed the "server based" client by over 400%. The vector data handler outperformed the "server based" client by over 50%, while the database client was over 250% quicker.

GIS users rated all components to be of significant benefit, offering improved performance over their current GIS viewing tools. The work completed in this thesis provides Genasys (New Zealand) a useful tool to enable powerful, fast and stable Java based GIS viewing clients.

Keywords: ***GIS, Java, computer graphics, image pyramid.***

Acknowledgements

During the course of this project, I have been helped by numerous people. I would like to take this opportunity to thank all those people. I have listed a few below, without them this project would never have occurred.

Special thanks to Genasys New Zealand - Philip, Susan, Richard and Deborah - for providing the tools and technology for this project, and for fixing my problems.

To Genasys Australia - Bruce Lundy and co. Thanks for developing and providing the Spatial Nucleus toolkit.

To Hauraki District Council - Rex Leonhart, Matthew Radford for helping out when I needed it most.

Thanks to my family - Especially to Kate for doing lots and lots of proofing.

Special thanks to Wyatt Page and his purple cast, for all the great ideas, support and proofing.

Finally, I would like to thank Emma for putting up with me and then looking after me, oh and for the proofing too.

Contents

List Of Figures

List Of Tables

List of Code Fragments

Glossary

Cadastral	The legal definition of a land parcel as opposed to the physical boundary, which could be, made up of natural features like fences and hedges etc.
Class	A Java class file is the platform independent compiled source code.
Daemon	A daemon is a program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive. - Source: http://www.whatis.com
GIS	A Geographic Information System (GIS) is composed of groups of functions that manipulate spatial and attribute data. These functions embody the concepts of cartography, geodesy, mathematics and computer science. The purpose of the GIS is to provide the combined power of these disciplines for problem solving at a variety of levels of user expertise
GPS	Geographic Positioning System is a device that obtains its geographic position from a group of satellites orbiting the earth.
Raster	A raster image is a digital representation of an image. It consists of a value for each dot of the image, each dot is known as a pixel or picture element.
RDMBS	An RDBMS (relational database management system) is a program that lets you create, update, and administer a relational database
Socket	A socket is a method for communication between a client program and a server program in a network. A socket is defined as "the endpoint in a connection." - Source: http://www.whatis.com
TCP-IP	TCP/IP (Transmission Control Protocol/Internet Protocol) is the basic communication language or protocol of the Internet. - Source: http://www.whatis.com
TIFF	TIFF(Tag Image File Format) is a common format for exchanging raster (bitmapped) images between application programs
Vector	a sequence of commands or mathematical statements that place lines and shapes in a given two-dimensional or three-dimensional space

Chapter I. Introduction

Joe Tramper starts to get a little distressed, he feels like he is lost. Never mind, he pulls out his personal pocket Geographic Information System (GIS) & Geographic Positioning System (GPS). He pulls up the topographic map for the area, and asks the system to plot his location. While he is there he dials up the weather service provider, and displays the current weather map. It looks like it could get a little wet and windy on his current path, and with the hut still four hours away, he decides to cancel his hut booking and look for a camp site nearby. He displays a digital satellite image and scans the area looking for a good site. After locating a drinking water source, he changes his trip plan on the GIS & GPS system. After an hour he reaches his chosen campsite, and pitches his tent. He lights a fire to keep warm for the night.

All of a sudden, the wind picks up and throws a spark from the fire into the native forest. Should have checked the wind direction forecast! Never mind, he calls in the emergency services and gives them his exact location. He plots a safe & quick route away from the now ferocious fire.

The emergency services dispatch resources to fight the fire from air and land. Their GIS system provides them with the location of their resources' at all times. Constant weather updates and information about the surrounding vegetation allow them to model the expected progress of the fire. This model is uploaded to each resources personal GIS system, letting them know if they are in danger.

After fighting the fire all night, the wind changes direction. The GIS predicts that the fire will move towards a nearby township, civil defense is called in to be on standby.

The area contains some endangered native animal species. Worried about the consequences of the fire, the emergency services contact the conservation agency. The conservation agency downloads the fire information from the emergency services. They overlay the current location of the entire native species in the area overtop the fire information. A decision is made that they need to be rescued. With the aid of the GIS topographic information and track maps, they ask the GIS to calculate the quickest route in and out of the area. They can be rescued, but only just. The conservation rescue team is dispatched. With their personal GIS notebooks they locate the animals and remove them from the path of the fire.

While this example is in the extreme, the reality of GIS is that organisations will become increasingly dependent on the information they are able to provide. There is no limit to the possible uses for a GIS. Any information which can be spatially referenced may provide some increased knowledge and aid the decision making process. Key to the wide use of a GIS is the availability of a fast and reliable client viewing tool.

This project is an investigation into advances that can be made with Java GIS viewing tools. It has been undertaken in conjunction with Genasys (New Zealand), a developer of GIS solutions (Section 2.02.)

Section 1.01 Thesis Overview

In the majority of organisations with a GIS, the users can be classified into 3 types: casual users; operational users; power users. Genasys literature quotes the ratio between these users to be 1000:20:1[1]. Within New Zealand in an analysis of two medium - small local government Genasys sites, the ratio was 30:3:1[2].

The casual users view data, make enquiries on the data, and print the results. An operational user may run specific applications. These applications are typically for data modification or addition. A power user includes the system administrator, and is responsible for application development and some of the more powerful GIS functions.

Industry analysis indicates that display of spatial data and spatial analysis constitutes nearly 50% of the GIS operations undertaken. With viewing tools that are limited in their speed and ability, it is essential that this 50% market be catered for with easy to use and efficient tools. It is evident that no acceptable solution exists for casual users.

A GIS viewing tool can operate using one of three methods; a server based system, where the server draws data and the client retrieves a GIF image from the server display; a full-download system, where the client downloads binary data from the server and draws it on the client display, and a hybrid system, where the client obtains a GIF image or downloads data for drawing on the client depending on the scale of the client display.

This thesis is an investigation into the full-download system. The full-download system is based on the premise that client systems are now full powered machines, two local government sites investigated above have high specification machines[3]. These machines are more than adequate to handle a significant level of data processing. The system is designed to remove as much processing from the server as possible. In the server based system, the server processes maps, searches text strings and added numeric data for the client. For this investigation, all non-GIS specific processing is moved to the client. In addition, some simple GIS processing is handled on the client - rendering data, zooming, panning, and selecting data.

The system revolves around three components, a vector data handler to draw vector objects to the client screen, a database client to retrieve data from database servers for use with the

[1] Beyond Desktop Mapping : The Internet and the future of GIS : Tony Hart, Director Asia, Genasys II Pty Ltd
[2] Hauraki District Council / Buller District Council
[3] Intel Pentium II 300's, 64-128 MB RAM, 4 GB Hard Drives.

vector objects and for display to the client. The final component is the large image handler, which draws images to the client screen. The goals of this investigation were to:

- Move non-GIS processing from the server to the client
- Develop a large image handler service for the client
- Develop a vector data rendering service for the client
- Develop a database retrieval client
- Investigate new interface features

Section 1.02 Contents By Chapter

This thesis has been separated into chapters for each component of the system. In addition an introduction to GIS and Java is included.

Chapter II introduces GIS and the Genasys organisation. It contains a summary of the GIS products that Genasys develops. This chapter includes a history and introduction to the Java programming language.

Chapter III details the design and implementation of a large image handler. The large image handler is responsible for the rendering of images directly from the server. These images vary in size and can be greater than eighty megabytes.

Chapter IV describes the database client. The database client is responsible for retrieving data from a database server for display to the client, or to perform conditional rendering[4] of vector data.

Chapter V details the vector data handler. The vector data handler is responsible for the rendering of vector data. The vector data is downloaded from a server and drawn to screen. It may be drawn using symbols based on data that is obtained from the database client (Chapter IV)

Chapter VI deals with the integration of the three components described above. In addition, it describes some interface modifications that were trialed.

Chapter VII contains the results of a user evaluation that was performed at Hauraki District Council. It contains the results from three groups of users: casual users, operational users and power users.

Chapter VIII describes some of the difficulties that have been encountered with this project.

[4] Conditional rendering is the drawing of an object based in a style that is based on an external variable.

Chapter IX includes a discussion of the project, and future directions for this client. This section also contains a conclusion for the project.

Chapter II. GIS, Genasys and Java

Section 2.01 GIS

There are several formal definitions for GIS available, perhaps one of the better sources to find definitions of GIS is the Usenet news group comp.infosystems.gis. The following responses[5] were found after a recent discussion of the "true" definition for a GIS:

- "A Geographic Information System (GIS) is any system designed for the capturing, storing, checking, integrating, analyzing and displaying of spatially referenced data about the earth."

- "A GIS is a system that allows one to combine geographic data sets, or layers, and create new, unique geospatial data to which one may apply standard spatial analysis tools."

- "An organized collection of computer hardware, software and procedures designed to support the capture, editing, management, manipulation, analysis, modelling, and display of spatially referenced data for solving complex planning and management problems. A GIS is characterized by its ability to perform topological structuring of data."

- "A GIS is a set of principals, methods and tools used to capture, store, transform, analyze, model, simulate and display spatial and nonspatial data. The information resulting from the whole process aims at both understanding and explaining the spatial distribution of phenomena and events, and at supporting planning decision making process"

- "GIS relates a location to an asset or an event. The system may be manual or computerized."

- "A GIS (Geographical Information System) is any system that makes a link between statistical data and the geographic location involved. Therefore, to be fitted into a GIS, the data must include a geographical element. Either a set of coordinates (e.g. spot height, soil sample) or a link to an object that is defined in geographical coordinates (e.g. street, parcel)."

- "A relational database with a sense of space."

[5] comp.infosystems.gis discussion located at: http://www.census.gov/ftp/pub/geo/www/cols/jan98.html (January 1999)

From recent literature more succinct definitions of GIS are:

- "An information system that is designed to work with data referenced by spatial or geographic coordinates. In other words, a GIS is both a database system with specific capabilities for spatially-referenced data, as well as a set of operations for working with the data." [6]

- "A system of hardware, software, data, people, organizations and institutional arrangements for collecting, storing, analyzing, and disseminating information about areas of the earth."[7]

It is evident from the variety of definitions available that there will never be a single accepted definition for GIS. The Genasys[8] definition of a GIS states simply:

- "A Geographic Information System (GIS) is composed of groups of functions that manipulate spatial and attribute data. These functions embody the concepts of cartography, geodesy, mathematics and computer science. The purpose of the GIS is to provide the **combined power** of these disciplines for **problem solving** at a variety of levels of **user expertise**."

This definition leads to the three important facets of a GIS:

- A GIS is about the integration in a spatial context, of tools from a variety of disciplines.

- These tools are combined for the purpose of problem solving.

- Accessibility to the problem solving capabilities of a GIS is essential.

GIS are often confused with other forms of information technology these include:

- Computer Assisted Drawing (CAD) - used for construction drawings, finer scale, typically not maps, no 'useful' database link [9]

- Computer Mapping - visual generation tool, no analytical capabilities

- Database System - information without a spatial capability

From the three information technologies listed above, the requirements for a GIS include:

---

[6] Star, Estes, GEOGRAPHIC INFORMATION SYSTEMS: Introduction. Prentice Hall, 1990

[7] Dueker, Kjerne. Multipurpose cadastre : terms and definitions. American Society for Photogrammetry and Remote Sensing : American Congress on Surveying and Mapping, 1989.

[8] Genasys New Zealand are a GIS software developer, this project has been undertaken with their support. For more information on Genasys see Section 2.02

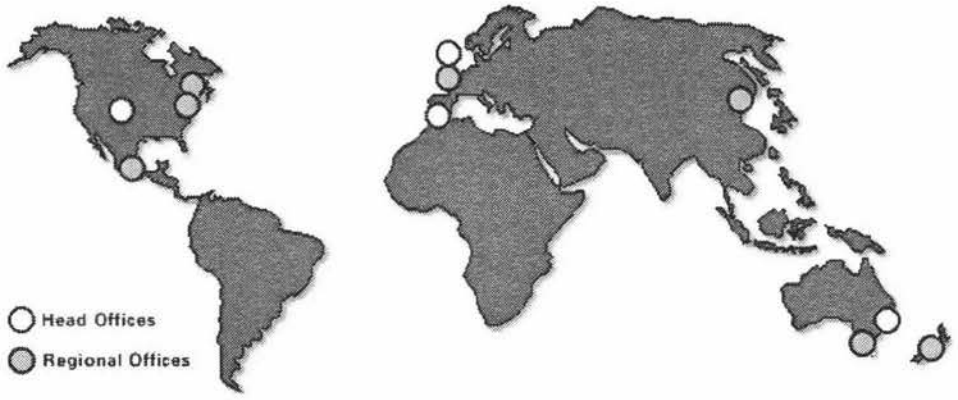
[9] The idea of a CAD system not being used for maps is being eroded, many consultants now use CAD as a cheap digitising option to a full GIS system. However the 'useful' link to a RDBMS and the ability to use the information from the database is still a defining factor.

- ❖ Ability to display information spatially
- ❖ Ability to interact usefully with database systems
- ❖ Analytical capability

Section 2.02 Genasys

The Genasys organisation started in 1976 when Burwood Council in North Sydney, Australia, installed the Genasys suite of local government software. Their first entry into the GIS market was in 1987, with software that was originally developed in North America.

Genasys is the 5th largest Australian owned software company, and the 7th largest exporter of Australian software. In Australasia, Genasys employs approximately 120 staff. Genasys is a worldwide organisation with offices in all regions of the world.



• Figure 1 Genasys Office Locations

Genasys have customers from four main industry sectors; oil and gas, transportation and communication, military, and local government.

They have lead the GIS world with a number of innovations, including the first client-server GIS with full X-windows graphics, the first GIS Advanced Programming Interface (API), and the first UNIX designed and implemented GIS.

Section 2.03 Genasys Software

(a) GenaMap

The Genasys Geographic Information Systems (GIS) product line revolves around their mapping product - GenaMap. The product is based on a client-server architecture, integrating vector information (Legal boundaries, pipes, cables, etc.) and raster information (satellite and aerial sources, terrain models, etc.) for display and analysis in a single environment.

GenaMap provides the following functions:

- Data Entry
- Data Selection
- Data Management
- Data Analysis
- Data Output

(b) Request Broker API

One of the advanced features of the Genasys software is the unique *Request Broker* architecture. The *Request Broker* is based on CORBA [10] principles and is a tool that allows Genasys applications to communicate between each other, but also allows third-party developers to enhance the existing Genasys application suite.

The Genasys *Request Broker* API (Appendix IV) is a command dispatcher product. This API enables the integration of Genasys products and allows third party developers to integrate with these products.

The *Request Broker* allows the addition of new functionality to the Genasys product line as well as allowing access other products that implement the *Request Broker* architecture. All the Genasys products are designed using the *Request Broker* architecture.

(c) Web Broker

Genasys have developed a method of delivering data to clients via Internet protocols. By utilising the *Request Broker* architecture, all of the functions of the Genasys product line are available to the *Web Broker*. This includes the complex spatial analysis tools that the GenaMap software provides. The system is an application that accesses the Genasys product line by executing GenaMap scripts and returning results to a client. These results can be textual data or GIF images. The *Web Broker* allows spatial data to be incorporated into World Wide Web pages. By using programming languages designed for use over the Internet such as Java, developers can create GIS viewing tools like the *Military GIS viewer (MGIS)* (Section 2.03(e)).

[10] CORBA (Common Object Request Broker Architecture) is an architecture and specification for creating, distributing, and managing distributed program objects in a network. It allows programs at different locations and developed by different vendors to communicate in a network through an "interface broker." (source: http://www.whatis.com/corba.htm)

One of the disadvantages of the *Web Broker* product is that it is unavailable for the Microsoft Windows NT platform. With Windows NT becoming a popular choice for a significant number of corporate users, Genasys currently have no solution for their Windows NT customers.

(d) Nova Vista / Spatial Nucleus

The *Nova Vista* technology is a spatial middleware[11] designed to allow application developers to spatially enable their software. The primary objective of this technology is to minimize the amount of knowledge about spatial systems that developers require. The developers' application is not required to deal with spatial data, relate it to non-spatial data or to read different data formats. The *Nova Vista* technology provides this functionality.

The system is comprised of two components; The *Client Development Kit (CDK)* is the beginning of a spatially enabled Java toolkit while *Spatial Nucleus* is a system that integrates spatial data into client applications.

The *Spatial Nucleus* toolkit incorporates an *Accessor* API. The accessor API defines how the *Nova Vista* technology can retrieve data from a datasource. In addition, the API allows the development of new accessors, allowing the system to retrieve data from a previously unsupported data format. The *Accessor* API can be used by third party applications to access data sources.

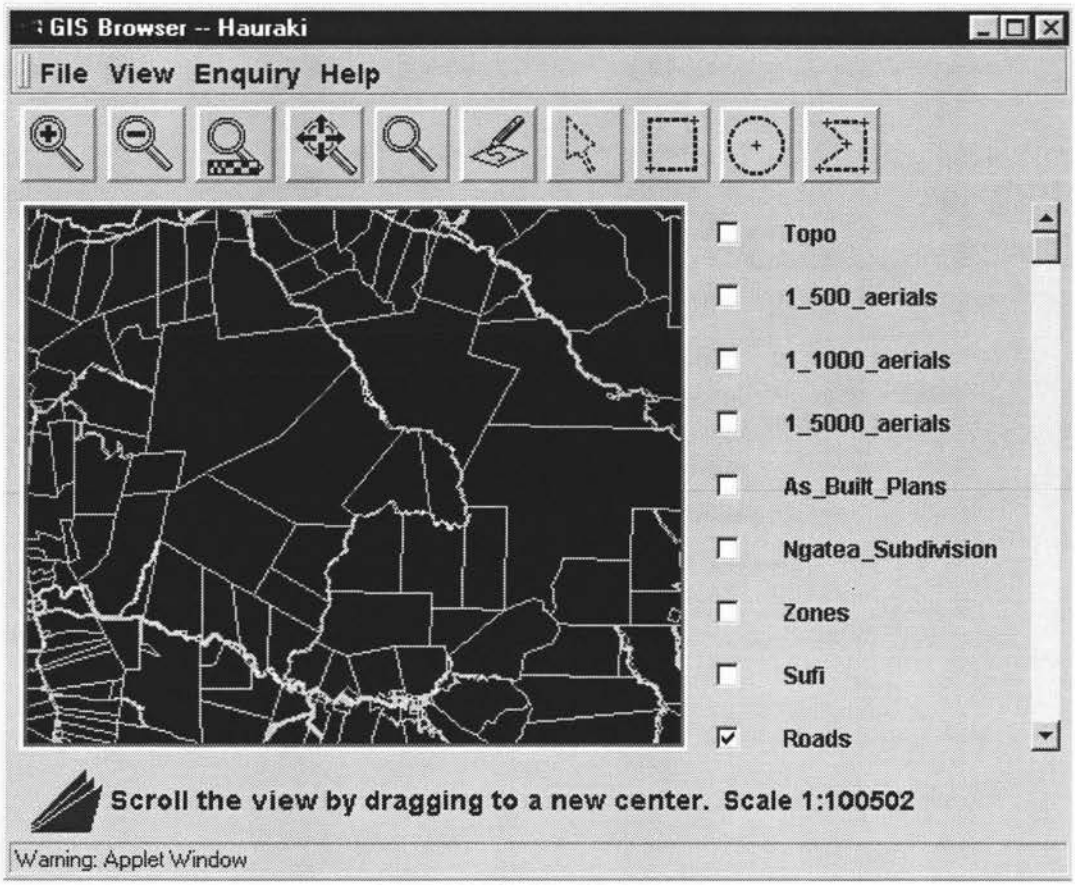
(e) MGIS Client

The MGIS client (Figure 2) was originally developed by Genasys for the Australian military. The MGIS client sends requests to the *Web Broker* and retrieves either a GIF image or textual data as required.

The MGIS client was originally a bare-bones client, which could handle enquiries, zooming and panning. The client was enhanced by the Hauraki District Council (Paeroa, New Zealand), to incorporate navigation, printing and some further display and measuring functions.

The client has been developed in Java and is able to operate on a variety of platforms that handle Java version 1.02. The system has gained popularity amongst the New Zealand local government Genasys users, as the preferred means of delivering client access to the Genasys product suite.

[11] Middleware is a general term for any programming that serves to "glue together," mediate between, or enhance two separate and usually already existing programs - Source: http://www.whatis.com : January 1998



• Figure 2 MGIS Client Viewer

Despite its popularity, the MGIS client is noted to be slow under some circumstances, especially when working with images and processing information.

Section 2.04 Java

The MGIS system was developed using the Java Development Kit. Java offers a unique solution for software development, deployment and maintenance. No longer do software developers need to re-compile a piece of software for each possible platform. Java offers a solution that allows a client - with an unknown hardware platform - to connect to a web site and run a GIS application. As an update to the GIS application becomes available, the system administrator updates the central web site with the appropriate files. There is no need to visit and install software on the client machine, as the client web browser downloads the updated files from the web site automatically.

This section details the history of Java and introduces the language and its features.[12]

(a) History

Java was first released to the public in January 1995, but its history began five years earlier, in 1990. Java's concepts were developed in 1990, when a team from Sun Microsystems

[12] For further information regarding Java, visit http://java.sun.com : December 1998

developed some innovative concepts to be applied to consumer electronics. Consumer electronics have been driven by microprocessors for a long period of time, but each required its own interface, and its own remote control. The Sun researchers developed concepts to allow a single control for consumer appliances. The team of researchers formed a group code-named Green, whose task was to create a simple device that could control a wide variety of consumer electronics devices. These devices included appliances like the television, VCR, stereo systems, ovens, and security systems.

Team Green consisted of just two programmers and an engineer. Perhaps the most influential of the two programmers was James Gosling. James Gosling is now the Vice President, Sun Fellow, of Sun Microsystems, Inc. Gosling identified that traditional programming languages like C and C++ have an emphasis on speed. They often lacked the reliability that was required in home consumer devices. These devices have little need for speed, their procedures require just a few operations. The remainder of the team consisted of Patrick Naughton and engineer Mike Sheridan. This combination created a new language in August 1991, code-named Oak.

Oak was used to develop a VCR programmer. This device had no buttons, no keyboard, just a screen. The device was turned on by touching the screen, and controlled by moving your finger across the screen. The chips used in the device were too expensive, hence the technology was not embraced by the industry.

Nevertheless, the technology was not left sitting in the archives. In late 1994, early 1995, one of the co-founders of Sun, Bill Joy, saw the opportunity for robust, secure and architecture neutral language for use over the World Wide Web. Oak was renamed to Java so as to avoid a name clash with another product. The idea was to release Java for free over the Internet, attempting to make it the standard.

Java is now the defacto standard for programming languages used on the Internet. It was helped by Microsoft agreeing to use Java in the Internet Explorer browser. This provided Sun with the previously hard to attract Windows users. In addition, this was seen as an endorsement from the worlds' largest software manufacturer.

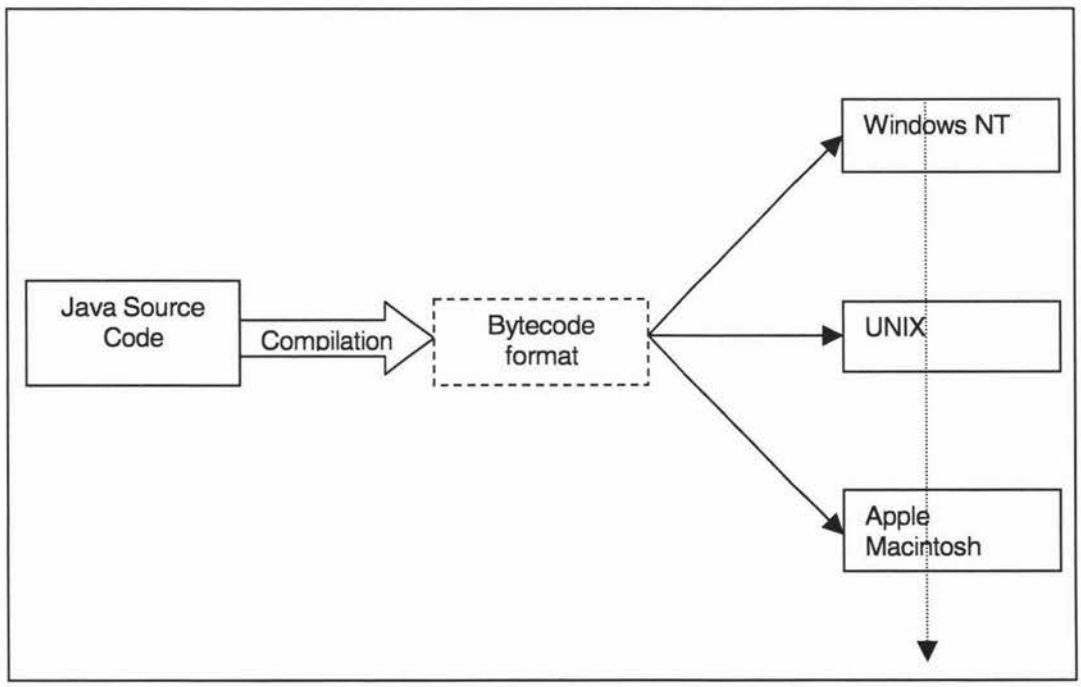
(b) Java Design

"A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language." This is the official description of Java, given by its makers, Sun Microsystems. Java is a continually evolving language, which has been designed to meet the official description. This section will describe some of the design goals of Java and how Sun Microsystems has managed to meet these goals.

In order for Java to become accepted and widely used, it was identified that Java would need to be a simple to learn language, for new programmers. But it would also need to be familiar to

those programmers who already use languages, like C++. As a result of these design goals, the fundamental concepts of Java can be understood by experienced programmers and new programmers alike. There is limited need for extensive programmer training. In addition, Java looks similar to C++. This allows C++ programmers to migrate quickly to the Java language. Table 1 illustrates some important similarities and differences between C++, Java and Visual Basic. A major design goal of the Java platform was for it to be reliable. With the original concept being for use with consumer electronics, an area that demands reliability, Java has several features to ensure reliability. These features include compile time checking and then extensive run-time checking. The memory management system limits the possibility for developed systems to include major problems. The majority of problems will be trapped during the development phase. The language directs programmers to employ reliable programming techniques.

Bill Joy proposed the use of Java over the Internet. One of the primary concerns became the security of the programming language. The system employs security features that are executed at run-time. This ensures that software cannot be invaded by unauthorised code, making it ideal for use in distributed environments like the World Wide Web. The system provides methods to limit the ability for Java to be used to create viruses.



• Figure 3 Process of compilation to platform independent bytecode format

Security checks are performed on the compiled Java source code (*bytecodes*), to ensure they haven't been modified between compilation and execution. A Java *bytecode* is a platform neutral format that is designed to allow the efficient transportation of the code between different platforms. Java source code is compiled into the *bytecode* format (Figure 3) This *bytecode*

format is interpreted on the target platform, it does not need to be recompiled for each platform, as the same *bytecode* can be used.

Java eliminates the possibility of incompatibilities between platforms by strictly specifying the basic language. This enables portability between platforms. The Java specification states the sizes of the data types and the behavior of the arithmetic operations. The *bytecodes* that are produced are interpreted on the desired platform.

The Java run-time system is dynamic in its linking stage. Classes are linked on demand, and can be linked from a variety of sources. For instance, a class that is continually being updated could be downloaded and linked on demand from the developers' web-site.

One of the early criticisms of Java was that it was too slow. However, with rapid advances in the area of Just In Time compilation, this criticism is being eroded. A Just In Time compiler compiles code just prior to being executed[13]. Just In Time compilers are providing performance as close to "C" and "C++" compiled code, and there is a belief that in the future they may surpass C and C++ performance (Appendix I).

• Table 1 Comparison between Java, C++ and Visual Basic modified from Byte, April 1996

FEATURE	JAVA	C++	VISUAL BASIC
Components and scripting	●	○	●
Extensible Source code	●	●	○
Portable Source code	●	●	○
Portable Object	●	○	○
Native-code compiler	●	●	○
Dynamic memory manager and garbage collector	●	○	●
Secure mode	●	○	○
Learning time	Medium	Long	Short
Mature tools and libraries	●	●	●

[13] Further information on Just In Time Compilers can be found at : http://java.sun.com/docs/jit_interface.html : January 1999

Chapter III. Large Image Handler

Section 3.01 Introduction

Documents in conventional media, such as books, newspapers and microfiche, can be converted into the digital form of raster images (Section 3.01(a)) by the use of scanners. Many data sets come in raster format, for example, aerial photographs, topographic maps, and historical maps. Advances in computer technology during the past decade have resulted in electronic means for generating, storing, retrieving and utilising vector data.

Raster Data is a major component of a GIS. However, due to the large size of raster data, support in client systems is usually limited. It is essential that raster data is available to a client, as some datasets such as aerial photographs are only available in raster form.

The following requirements were identified for a client image viewing system:

- Minimum image download time independent of image size and current location in the image
- Update client display as images become available
- Handle large images while maintaining usability

(a) Raster Images

A raster image is a digital representation of an image. It consists of a value for each dot of the image, each dot is known as a pixel or picture element. Depending on the type of image the possible value each pixel may have varies. In a two-bit or binary file, each pixel can be either black or white. Whereas in an eight-bit color image, each pixel may have a different level of 256 colours. In comparison, for a twenty-four bit image, each pixel can have a value from 16,777,216 colours. The number of available colours is known as the colour depth of the image. Images with a greater colour depth require more disk space to store the greater number of possible colours.

(b) Typical GIS Raster Data set

In a recent survey of Genasys New Zealand users[14], it was found that all local authorities using the GenaMap product use some form of Terralink Topographic Maps. The topographic maps come in various scales, one of the most common scales is 1:50,000. An example organisation, Hauraki District Council - a small/medium size local authority - was chosen to illustrate the typical size of a raster data set. The Hauraki District Council has an area of 1144 sq. metres.

[14] 1998 Genasys New Zealand User Group Conference - Wellington

Their topographic images are stored in uncompressed TIFF format (Section 3.01(c)). The total dataset size is 104 megabytes, representing 934 kilobytes of data per square metre.

It is common for a service organisation like Hauraki District Council to have many raster datasets. On a daily basis, the Hauraki District Council uses topographic maps, aerial photographs, historical maps, hand-drawn maps, and photographs. It is evident that with this number of raster datasets, a raster data handler is an essential tool for the GIS client.

(c) TIFF Image Format

The TIFF standard was developed by Aldus Corporation in conjunction with various manufacturers and software developers. The first public release was made in 1986. Two revisions have been made since that date, one in 1987, and the second in 1998. TIFF images are widely supported in image readers and writers. The majority of image data for GenaMap is stored in the TIFF image file format.

The TIFF file format is designed to describe and store raster data. It provides a powerful and scaleable data format, which can handle the most complex data information. Yet it maintains the ability to be used for small images. This is due to the limited number of required information fields. The format has been designed to allow improvements, without affecting existing software readers. The format has a set of minimum requirements for readers, all TIFF readers should implement all of these features. An in depth description of the TIFF image format can be found in Appendix V.

The TIFF format includes the following features:

- TIFF can handle a wide variety of colormap systems. They include bi-level (black and white), greyscale, palette colour, and full colour.
- The system also handles a number of compression systems. These include CCITT 1D, Group 3 and 4 FAX, PackBits, LZW and JPEG compression.
- TIFF is a portable graphics format that is not specific to one operating system or processor. Files indicate which byte-order they have been written in with a flag at the beginning of the file.
- TIFF allows developers to extend the TIFF format, including the storage of private information.
- TIFF handles stripped images and tiled images.

One of the major features of TIFF images is its ability to handle tiled images. A standard TIFF images' data is organised in stripped format - horizontally wide and vertically narrow. Each strip of an image is known as a scan-line. The tiled image format organises data in square tiles. It is

used for some compression schemes and stores high-resolution images, providing an increase in performance for large images, like the topographic map series described in above in Section 3.01(b). The TIFF reader that is incorporated in the large image handler handles the base-line TIFF format and the tiled image extension (Appendix V).

(d) Techniques of Reducing Transfer Bottleneck

In order to utilise large raster data sets it is essential to minimise the time a user is waiting to view the data. There are three options to speed the transfer of images: the first is to increase the bandwidth of the transfer channel. The second option is to reduce the amount of data that must be transferred. The third option is to compress the data set. These techniques are often combined to provide additional performance over a single method.

(i) Bandwidth Increase

Increasing the bandwidth is a site-specific option, which some GIS sites have already undertaken. Most are using 10 Megabit per second network topologies. In an ideal 100 Megabit/second network, transferring the entire data set of Hauraki District Council would take 12.2 seconds. Whereas in a 10 Megabit/second network it would take 122 seconds. Over a 28.8 kilobit/second modem, it would take 8 1/2 hours. These figures provide data transfer speeds in an ideal situation, typical figures are at least 70% less than those provided, depending on the bandwidth available. As the load increases over a network the transfer speed of raster data slows significantly.

While increasing the bandwidth allows image data to be transferred quickly, it does not control how usable the data is. Few clients have the memory capacity to handle the entire 100 MB of image data. Those that are able to handle this volume of data, tend to become slow and cumbersome.

(ii) Data Compression Techniques

Various techniques are employed to compress raster data. For example, the TIFF standard (Appendix V) allows a number of compression techniques. Compressing the Hauraki District Council dataset using the LZW compression scheme decreases its size by 21%. However compressing the dataset does not increase the usability of the data. The processing power required to decompress an image while displaying it, often outweighs the benefit of compressing it. In addition, accessing a specific region of interest in a compressed image becomes more complex.

(iii) Data Transfer Techniques

There are various techniques available to reduce the demands on Input / Output, network and client resources. Multi-resolution representations of images are a typical method for the

storage organization of images. Image decomposition techniques such as wavelets[15,16] can be used to provide multiple resolution images and compress the data.

In recent years, the use of wavelets for image storage and transfer, and image processing has been investigated. Wavelets are a mathematical tool for hierarchically decomposing functions. They allow the decomposition of an image into its overall shape and the image details. Wavelets offer an excellent technique for representing the various levels of detail present in an image.

Wavelets were investigated for use with this application, and while they require the same space as the original image, it would effectively double the space requirements, as the images are still needed in their original format for use in GenaMap. The second problem identified with wavelets was that the computing power required to decompress the image while displaying it on the client was significant.

A simpler method is to create a multi-resolution image pyramid.[17] With a multiple resolution image the image is represented by several sets of coefficients, each set of coefficients has a visual similarity to the original image but at a lower resolution. If a dyadic or an octave halving method is used then the maximum space increase for this type of image is one third of the original image. Multi-resolution systems display an image from the multi-resolution pyramid that matches the current display resolution. As the user requires better resolution image, it is provided from the image pyramid.

A multi-resolution image pyramid system was chosen as an alternative to the wavelet based system. The multi-resolution image pyramid uses the current region of interest to download only the data that is currently on screen. If the client requires a different region of the image, it is downloaded on demand. Sections of an image that have already been downloaded are cached[18] on the client machine. If an image section that is being cached is not used for a specified length of time, then it is discarded from memory and downloaded again if required.

Section 3.02 Pyramid Structure

An image pyramid is a representation of an original image at various resolutions. Each level of the pyramid, requires less data storage, and is of lower resolution than the previous level. If every level of a dyadic image pyramid is used, the pyramid will increase the storage requirements of the original image by one third. However, it is not usual for a dyadic image

---

[15] Stollnitz, DeRose, Salesin, Wavelets for computer graphics: A primer Part 1. IEEE Computer Graphics and Applications, 15(3):76-84, May 1995

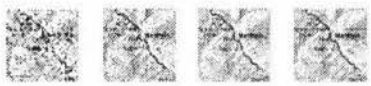
[16] Stollnitz, DeRose, Salesin, Wavelets for computer graphics: A primer Part 2. IEEE Computer Graphics and Applications, 15(4):75-85, July 1995

[17] Adelson et al, Pyramid methods in Image processing, RCA Engineer (29-6), Nov/Dec 1984

[18] Caching is when an object is stored somewhere more or less temporarily.

pyramid to be decomposed to each resolution level, so a typical increase in image size would be 30% of the original.

The resolution of the image being viewed should be matched to the current screen resolution. If an image is at a resolution significantly better than the resolution that can be displayed on client then it is inefficient to transfer the image as it must be down-sampled [19] on the client to be displayed. For this reason the pyramid structure stores images at varying resolutions, so the client can choose an image resolution that either matches the current screen resolution or is slightly better than the current resolution. This minimizes the data transfer requirements of the client. (Figure 4)



• Figure 4 Image Resolution Example: These images are of varying resolutions and there is little visual difference between the four images. Yet, there is a 2500% storage space difference between the far left and far right images.

Images are down-sampled from the best resolution in a dyadic fashion until a satisfactory initial image is obtained, as shown in Figure 5. The starting point is usually matched to the initial screen resolution of the client. The example image pyramid in Figure 5, uses four decomposition levels which represents a 16 times decrease in resolution from the original image. This provides a spread of images that covers the full resolution of the client system.

A user will typically zoom into a region of interest on an image. For instance, they may be interested in looking at a township. While zooming in on the township there is no need to download the remainder of the image that is currently off-screen. If at a later stage the user wishes to view the remainder of the image, it can be downloaded on demand.

In this system, images are stored in separate files. This eliminates the need for writing a custom TIFF writer to regenerate the images at the various resolutions of the pyramid. Users can generate each image by a GenaMap script or by an image-editing tool.

[19] Down-sampling is the reduction of resolution of an image

• Figure 5 A four level dyadic image pyramid.

The location and resolution of each image in the pyramid is stored in an ASCII text file that can be provided to the client. Each line of the text file represents a new pyramid level. Code Fragment 1 indicates the format of the file, the first portion being the name and location of the image pyramid file, and the second portion indicating the resolution of the images. The process for creating a new image pyramid is detailed in Appendix II.

```
\images\s12d.pyramid\5=125x94
\images\s12d.pyramid\4=250x188
\images\s12d.pyramid\3=500x375
\images\s12d.pyramid\2=999x750
\images\s12d.pyramid\1=1999x1500
\images\s12d=3998x3000
```

• Code Fragment 1 Image Pyramid Definition File

(a) Images within a spatial context.

To place images in a spatial context, an image needs to be transformed from image to mapping coordinates. GenaMap registers an image to mapping co-ordinates by the Genasys *Register* [20] program. This program transforms the co-ordinate system, generating a text file to indicate the transformations. The register program also performs any image warping that is

[20] The registration utility provides a general purpose mechanism for registering and reprojecting image data. The data can come from a variety of sources some of which include: orthophotos , satellite images, scanned works or engineering plans, scanned cadastral plans and topographic maps.: Genasys Image Registration Manual : 1998

required. The generated text file (Code Fragment 2) with an ".IF" extension identifies the co-ordinates of the image. The IF information is used to place the image in the correct spatial location.

```
extent
      minx = 2729998.642815
      maxx = 2750004.884483
      miny = 6429991.353873
      maxy = 6445002.291766
end
projection
      number = 32
      zone = 32767
      units = 2
      ellipsoid = 4
      parameter 0 = 6378388.0000000000
      parameter 1 = 0.0067226700
      parameter 2 = 0.0000000000
      parameter 3 = 0.0000000000
      parameter 4 = 173.0000000000
      parameter 5 = -41.0000000000
      parameter 6 = 2510000.0000000000
      parameter 7 = 6023150.0000000000
      parameter 8 = 0.0000000000
      parameter 9 = 0.0000000000
      parameter 10 = 135271374275838690916187840.0000000000
end
```

• Code Fragment 2 Sample Image Registration ".IF" File

Section 3.03 Image Server

The main task of the image server is to provide image data as fast as possible. However, it must also handle multiple clients efficiently. To provide data to multiple clients, the server is multi-threaded. One of the aims of this project has been to move as much processing as possible to the client. This means the servers' primary job is to open files, and deliver the requested sections of the file to the image client.

The server consists of a daemon[21] thread, which accepts connections from a client via a TCP-IP[22] socket[23]. As each client connects to the server, a new thread is created. This thread accepts commands from the client and acts on them. A client will request a new thread for each image, and several clients may access the same image simultaneously. The daemon thread continues to accept connections on the socket until the server is stopped.

[21] A daemon is a program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive. - Source: http://www.whatis.com

[22] TCP/IP (Transmission Control Protocol/Internet Protocol) is the basic communication language or protocol of the Internet. - Source: http://www.whatis.com

[23] Sockets is a method for communication between a client program and a server program in a network. A socket is defined as "the endpoint in a connection." - Source: http://www.whatis.com

The server can perform the following tasks:

- Open & Deliver .IF and Pyramid definition files
- Deliver the colourmap of an image
- Deliver specified scan-lines(Section 3.01(c)) from an image

Section 3.04 Java Image Implementation

(a) Image Producer / Observer

There are two distinct sections of the Image class in Java. In order to minimise download time over the Internet, image downloading is handled asynchronously. Asynchronous loading implies that the image loading process occurs out of step with the rest of the application, in a separate thread. Java separates the loading of an image into two separate steps. The first is the definition of the image, and the second is the actual transfer of image pixel data. There are two Java interfaces that describe the processes; they are the Image Producer *(java.awt.image.ImageProducer)* and an Image Observer *(java.awt.image.ImageObserver)*.

The *Image Producer* is responsible for producing sections of a specific image. As the *Image Producer* produces sections of an image, it notifies an *Image Observer*. The *Image Observer* can then perform operations using the image.

Each AWT component implements an *Image Observer*. This allows each component of the AWT to be notified of the progress of an image producer. The notification is typically used to update the client display.

The asynchronous nature of image download is essential in the desired application, it allows the image handler to produce images without being concerned with the usage of that image.

(b) Image Producer

The image producer performs two tasks. The first task is the setting of image parameters. This provides each image consumer with new information about the image. This includes the height, width, and data about how the *ImageProducer* will deliver the pixel data.

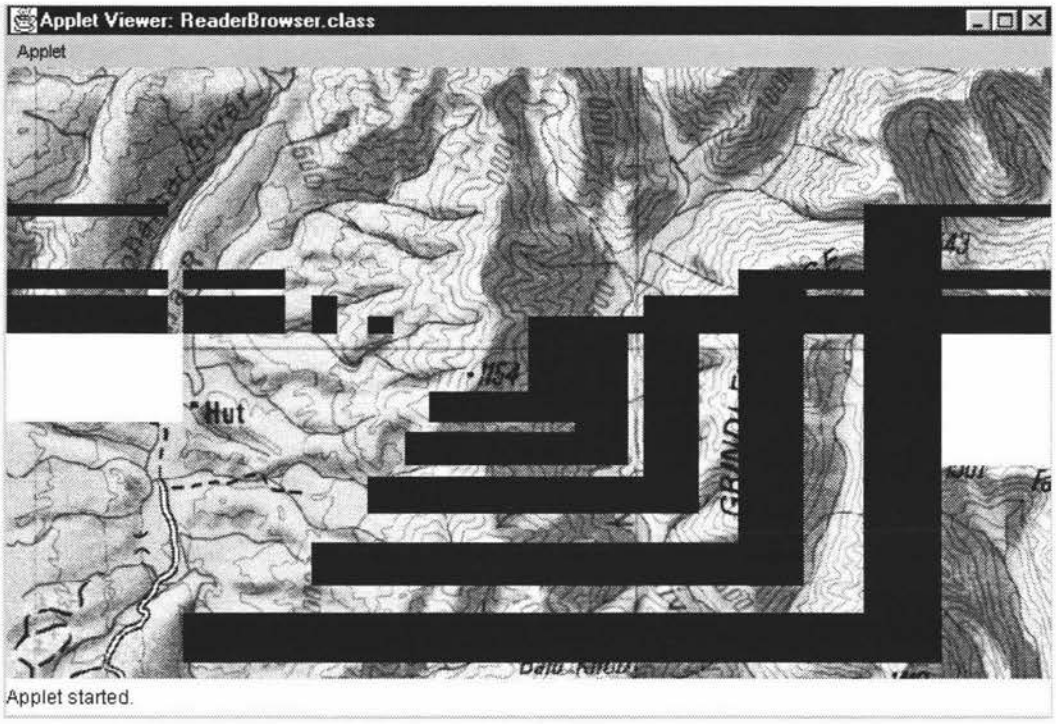
The second task of an image producer is the calling of *setPixels*. This is the method that delivers the actual image data or pixels to each image consumer. Before the image producer can call the *setPixels* method, it must download any image data that is required.

(c) Drawing An Image

The *java.awt.Image* class is a reference to the image data that is served by the image producer. Typically, the image data does not reside on the client computer, and must be

downloaded by the *ImageProducer* class. As data is downloaded by the *ImageProducer* it is able to be rendered to the *Image Object.*

While an image is being rendered, it is possible to draw the rendered sections before the image is complete, this is known as incremental drawing. Figure 6 shows how the incremental drawing feature is used to display loaded sections of an image while the remainder of the image is loading.



• Figure 6 Incremental Drawing Of Image

Section 3.05 Image Client

The image client is divided into three distinct sections, controlled by the *ImageManager* class. The first section is the image definition file readers. This section contains the *.IF* file reader and the image pyramid handler which decides on the appropriate image for the current screen resolution. The second section is the image memory handler. This allows the client to reference the entire image, but only have memory allocated for those sections which currently hold data. The final section is the image producer. This handles the download of data that hasn't already been downloaded from the server, and places it into the image memory. The image producer then serves data for the rendering of the image.

(a) Pyramid Manager

The pyramid manager reads in the *.IF* file, which allows it to reference the image in the New Zealand Map Grid co-ordinate system. The pyramid manager is called by the image manager

to provide the appropriate image producer from the image pyramid. There is an image producer for each level of the image pyramid.

The image pyramid manager determines which image producer to provide by comparing the region of interest in the mapping co-ordinate system and screen co-ordinates. The pyramid manager then traverses the image pyramid, calculating the region of interest width for each image. The pyramid level with the width most closely matches the screen co-ordinates width is chosen, and passed back to the image manager.

(b) Image Memory Handler

Datasets typically have a total size of over 100 megabytes and there may be more than a single dataset activated simultaneously. As such, it is essential that only the particular sections of any image that are being viewed are using memory. While an image may be stored on disk in complete form, there was a need to derive an in-memory version of the image that would allow the referencing of the entire image while only allocating memory on demand. The following section describes the memory model that was adopted for use in this application.

The essential requirement of the memory handler was for it to reference images in their complete form without allocating memory for the sections of the image which were not being viewed. Therefore, a structure was needed that could represent both allocated memory and unallocated memory.

The image manager calls the image producer to start the production of pixels for the current region of interest. The image producer creates an image memory handler to co-ordinate the allocation and de-referencing [24] of image memory. When the image memory handler is instantiated, it creates an image memory object. Image memory objects represent a region of an image. They either hold a reference to actual memory, or represent unallocated regions of an image. The initial image memory object that is created, represents an unallocated region covering the entire image. (Figure 7a)

If the image memory handler is asked to provide a region of an image, it first calls for all the image memory objects that are allocated within or covering the region of interest. This data is then passed back to the image producer for rendering.

The image memory handler then finds all the unallocated image memory objects that are within or covering the region of interest. These image memory objects are modified to reflect only the section that is within the current region of interest. (Figure 7b) Memory is allocated for this image memory object and new image memory objects are created for the remainder of the previous image memory object. (Figure 7b)

[24] It is the responsibility of the Java garbage collector to de-allocate un-referenced memory.

(c) Migration to Java2

This system was originally developed under Java version 1.02 and 1.1.4. These versions of Java did not handle image transparency for any images except GIF images. This meant that the vector layer (see Section 5.02) could not be placed over top of the image layer, or have multiple vector layers. The graphics engine was overhauled with *Java2* and a new API created. With the *Java2D* API, there is increased support for transparency in images. (see Section 5.04 page 72 for further details regarding the increased functionality with the Java2D API).

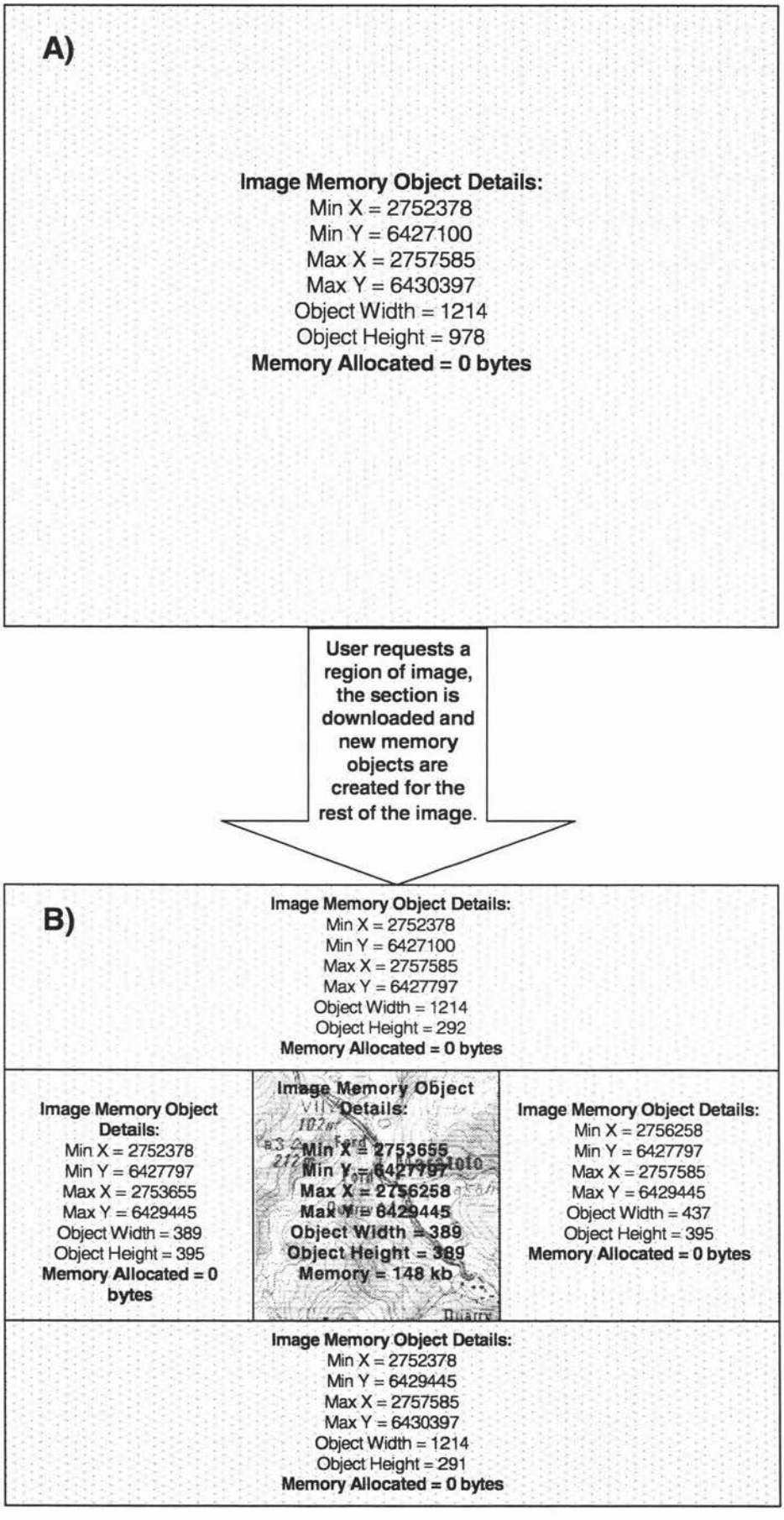
However, during the upgrade to *Java2*, an implementation bug[25] was discovered. The *Java2* Specification states that multiple calls to *drawImage* should render the currently available portion of an image. However this is not occurring, and the image is not being rendered until the *ImageProducer* calls *imageComplete*[26] with *STATICIMAGEDONE*.This bug caused the failure of the incremental drawing feature of the large image handler. The bug has been logged with Sun, the developers of Java, and it is hoped that the bug will be fixed in the next release of Java.

To work-around this bug a feature of the *ImageProducer/ImageConsumer* system has been used. The *ImageProducer/ImageConsumer* system allows multi-frame images. By treating the image as a multi-frame image we are able to incrementally draw the image. By calling *imageComplete* with a value of SINGLEFRAMEDONE, the image consumers believe that one frame of a multiframe image has been completed. In response, the *drawImage* method successfully renders the frame, or in our case each section of the complete image. This process is continued until the image is complete. This work-around slows the operation of our large image handler, because of the extra method being called, but allows us to incrementally draw the image.

---

[25] In computer technology, a bug is a coding error in a computer program - Source: http://www.whatis.com : January 1999

[26] The imageComplete method is called when the ImageProducer is finished delivering all of the pixels that the source image contains, or when a single frame of a multi-frame animation has been completed, or when an error in loading or producing the image has occured. : Java2 API Specification - http://java.sun.com/products/jdk/1.2/docs/api/java/awt/image/ImageConsumer.html : January 1999



• Figure 7 Image Memory Object Example Operation

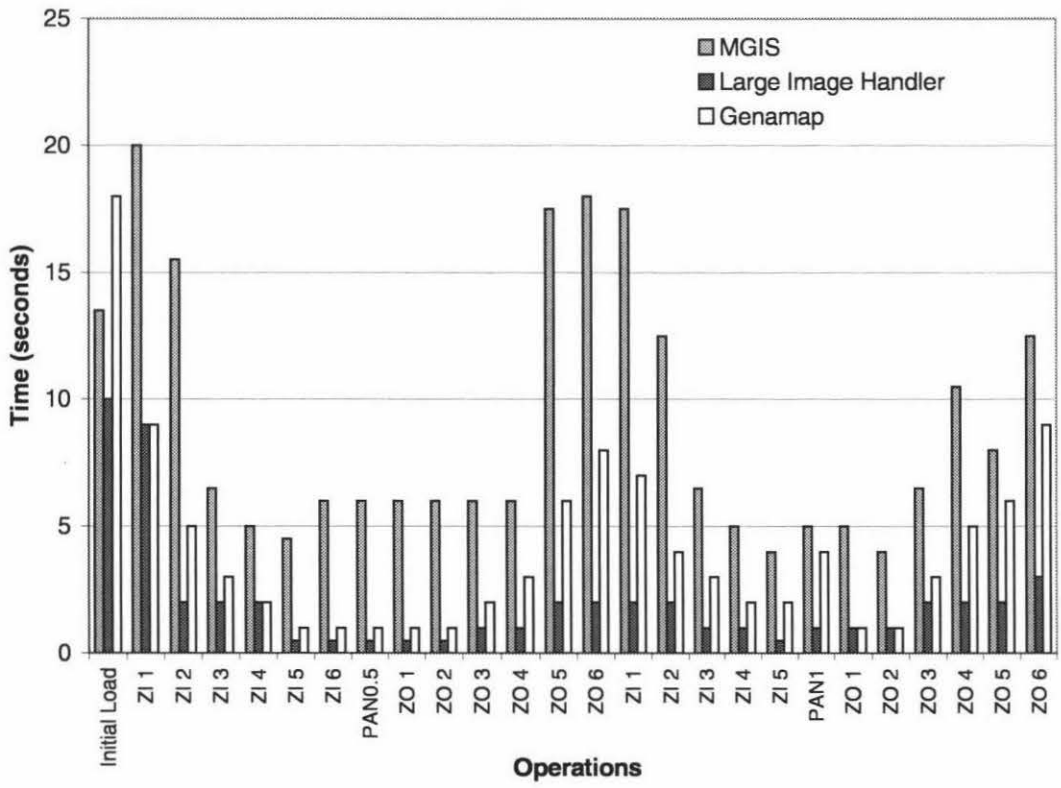
Section 3.06 Performance

The performance of the large image handler was quantified by comparing against two different systems. The first was a comparison with the performance of GenaMap for identical tasks. A second comparison was made to the MGIS Web Broker.

The comparisons involve measuring the time taken for operations to zoom in to a specific section of a large image, the time was then measured to pan left 1/2 a screen, followed by a pan left a whole screen width. Then finally a to zoom back to the original location. Both comparisons were made from the user interface

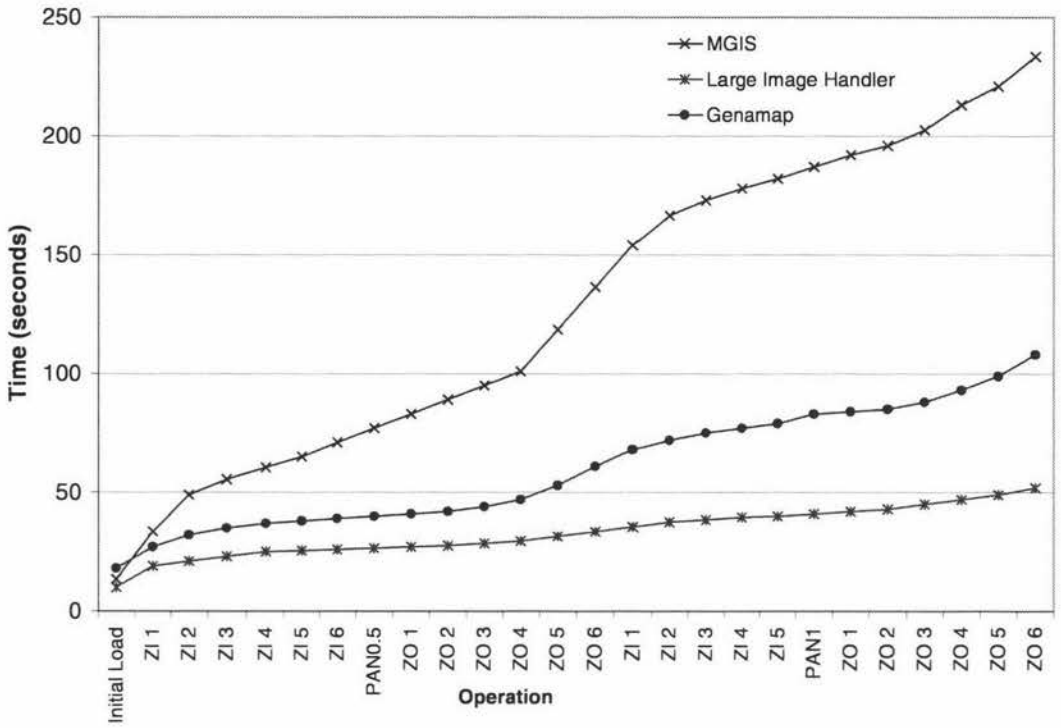
In all cases the large image handler outperformed both the MGIS and GenaMap systems. The MGIS system showed significant variability in its' performance, for certain operations it performed well in the first instance, but repeating the operation often took longer. (Figure 8)

The large image handler performance was exceptional. Once image data was loaded, the system redrew the screen within 2-3 seconds. (Figure 8) In comparison, GenaMap took a significant amount of time even when it had previously loaded the image data.



• Figure 8 Image Zooming and Panning Performance (ZI=Zoom In;ZO=Zoom Out;PA=Pan)

The large image handler was over fifty seconds quicker than the GenaMap product (Figure 9). This represents a substantial performance increase.



• Figure 9 Cumulative Image Zooming and Panning Performance (ZI=Zoom In;ZO=Zoom Out;PA=Pan)

Section 3.07 Summary

Raster images have limited support in client browsing systems. This is due to the large image sizes of raster images. A typical dataset can contain over 100 MB of image data. It is essential that a client provides a minimum image download time independent of image size.

Multi-resolution pyramid representations of images are a typical method for the storage organization of images. An image pyramid is a representation of an original image at various resolutions. If a dyadic decomposition scheme is used then the space requirements for an image will typically increase by 33% of the original image size. As the user requires better resolution image, it is provided from the image pyramid.

This image pyramid system provides a substantial improvement compared to existing products. The image pyramid system was over 50 seconds faster than GenaMap and over 150 seconds faster than the MGIS system when over 26 image zoom and pan operations were completed. This represents a speed increase of 220% and 450% respectively.

The second component of the GIS client is a system to provide vector data. Typically, this has been achieved by sending a request to a GIS server and retrieving a GIF image. It is proposed to download vector data to the client and render it on the client, in a similar fashion to the raster data. The rendering of vector data relies on links to external databases. The next section details the development of a database client to provide that link.

Chapter IV. Database Client For Java

Section 4.01 Introduction

A major component of GIS is the ability for a system to relate data from a variety of sources, be it vector data, raster data or textual information. Data can either be held internally as attributes to vector data, or can be held externally in a database. The majority of data will be held externally to the GIS, allowing multiple systems to update the data, e.g. a rating information system at a council will be updated by the rates system and accessed by the GIS. Thus, the ability for a system to access data from multiple real-time databases is essential to the success of a GIS. Database attributes connect to spatial features with a "key"[27] field. They can be used in query, analysis and presentation of data.

GenaMap supports an internal attribute database system. In addition, to support the variety of external data systems they have developed a Client-Server Database Server. The database servers provide a flexible and efficient access mechanism to enable GenaMap to communicate with multiple databases from anywhere on the network. Servers are currently available for Informix, Oracle, Microsoft SQL Server, Ingres, and Sybase.

This section details the design and development of a Database Client for the Genasys database server. Previously to obtain a list of data from a server, the client would call the Web Broker, and then call GenaMap to provide the list of data. GenaMap would then act as a client to a database server, requesting the data and then passing it back to the Web Broker. This system is extremely inefficient, passing the data through two intermediate systems before reaching the client. In most cases, it is not essential for GenaMap to handle this operation. For example, when a client requested data for textual display only, and not for use in a GenaMap display. This created a bottleneck as the database retrieval can take a significant time. In order to eliminate the bottleneck it was decided to develop a Java client which could access the database server to provide the necessary data, thereby eliminating the use of GenaMap for non-GIS processing and hence eliminating the bottleneck. It is envisaged that the database client can provide any non GIS-specific data that is required, increasing the overall efficiency of the system.

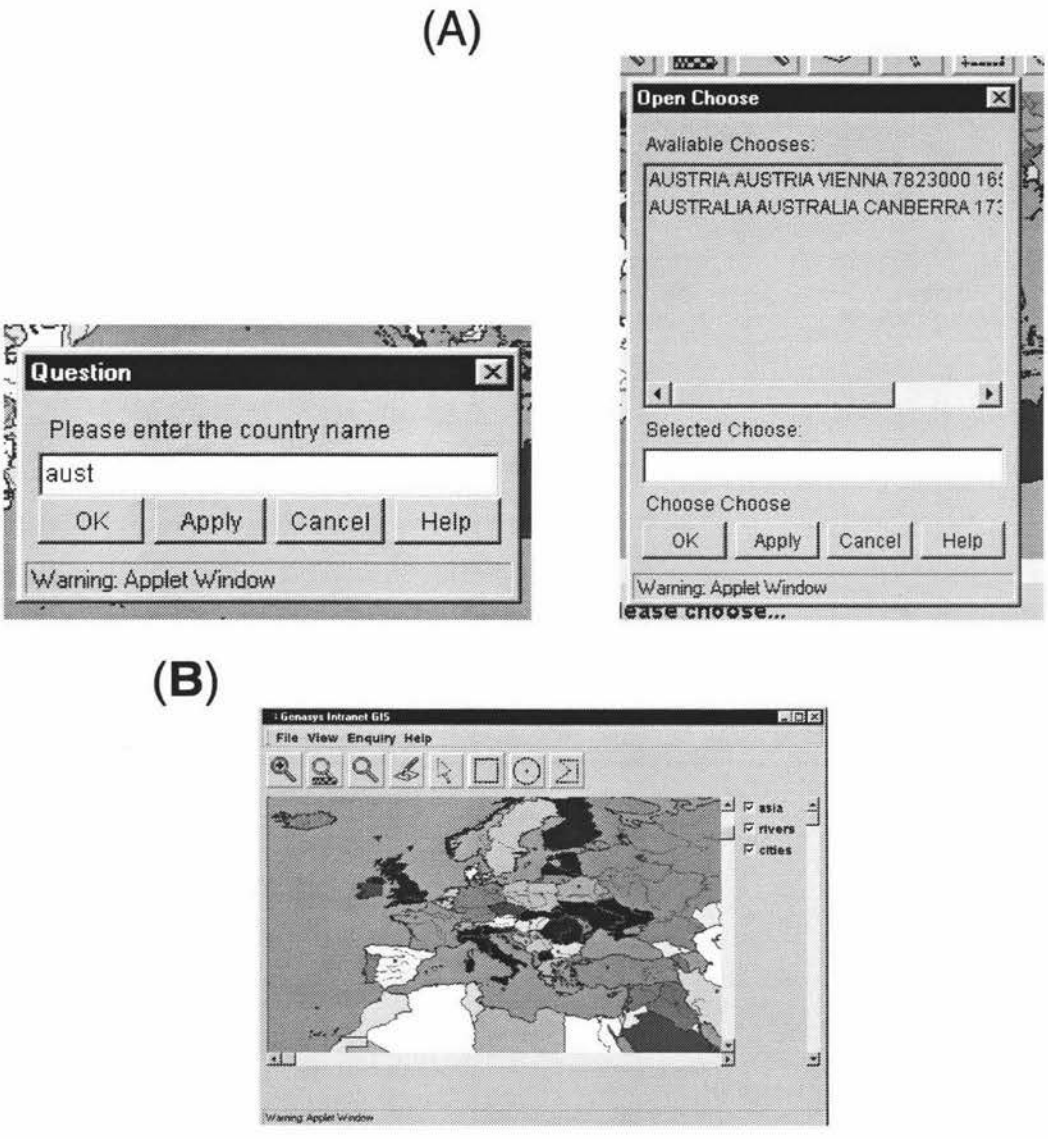
Section 4.02 Typical Scenario for Database Use

A typical use for the Java database client would be to execute a "Navigate" command.(Figure 10). A "navigate" command was previously controlled by a two-step process, i) obtain a list of objects to "navigate" to (Figure 10A) ; ii) "navigate" to the object as chosen by the user (Figure 10B). This two step process incorporates two separate sub-systems. The first is essentially a

[27] A key field is the linkage between the GIS and the RDBMS, it is often a unique identifier shared between the two systems

database query, handled by a *WebBroker* script. This step could be replaced by the database client. The second step is a spatial GIS function, handled directly by GenaMap.

While the "navigate" data is being retrieved from the database (step i), no other users are able to use the *WebBroker* system. Once the data is obtained from the database server, the GenaMap script then processes it, and sends the data to the client. On receipt of the data, the client re-processes it before displaying the data to the user.



• Figure 10 Example Navigation performed in the intranet client to country "Austria"

In contrast, the Java Database Client was designed to communicate directly with multiple database servers throughout the network. This would allow the database client to request non-GIS specific data directly and process it eliminating the re-processing steps, and allowing the *WebBroker* to remain free for use. This will provide a significant performance increase.

Section 4.03 Structure & Commands of GenaMap Database Servers

Genamap database servers provide access to third party databases. They provide a simple method of retrieving and updating data, which is typically achieved through the GenaMap product. The following table details the commands that are available to a client of the GenaMap database servers. The Java database client implements the following commands:

• Table 2 GenaMap Database Server command descriptions

create *creates a work table in the database. This table will be named tablename, and will have fields of the names and types specified in the field list. Valid field types are:*
insert *inserts a row of data in the current create table. There must be one word (as understood by the shell) per field in this table. Words of incorrect types will be silently converted to their 'nearest' equivalents.*
endcreate *terminates creation of a work table, flushing all rows to the database. insert may not be called after this statement.*
drop *deletes the named table from the database. This table must have been created in the same session using create, and endcreate must have already been called.*
select *initiates a query on the database. If the first fieldlist is *, all fields from the first table are retrieved, otherwise the specific list of fields will be retrieved. If the 'with' keyword is present, a join will be performed between the first and second tables. The first join field must correspond to a field from the first table, and the second join field to one from the second table.* *If no fields are specified for retrieval from the second table, none will be retrieved. Otherwise the named fields will be retrieved, and put at the beginning of the result.* *select will not work if a select is already current, or a table is being created.*
fetch *retrieves a row of data from the current select. Fields are separated by newlines.*
endselect *terminates the current select. Fetch may not be called after endselect was executed.*

Section 4.04 Description of Standards used in Database Server

The GenaMap database servers were designed to use the Remote Procedure Call (RPC)[28] and External Data Representation (XDR)[29] standards. These standards provide an industry-

[28] RFC1831: RPC: Remote Procedure Call Protocol Specification Version 2 : August 1995 : Sun Micrososystems

accepted method of communicating over a TCP-IP network. While they provide an easy method of developing network communication systems, they are not particularly efficient. A typical problem evident in the RPC and XDR standards is the padding of integer and Boolean values. A Boolean value requires one bit of information, however the XDR standard requires that Boolean values be stored as Integers. Integers are represented by 4 bytes of data, which is equivalent to 32 bits. This means that the 1 bit Boolean value must be padded with 31 bits of data.

(a) XDR

XDR is a standard for the description and encoding of data. This system is used when RPC is employed. It assumes that by using 4 byte data blocks as opposed to the traditional 8 byte it becomes useful for transferring data between different computer architectures. This assumption in theory, makes XDR ideal for the GenaMap database servers.

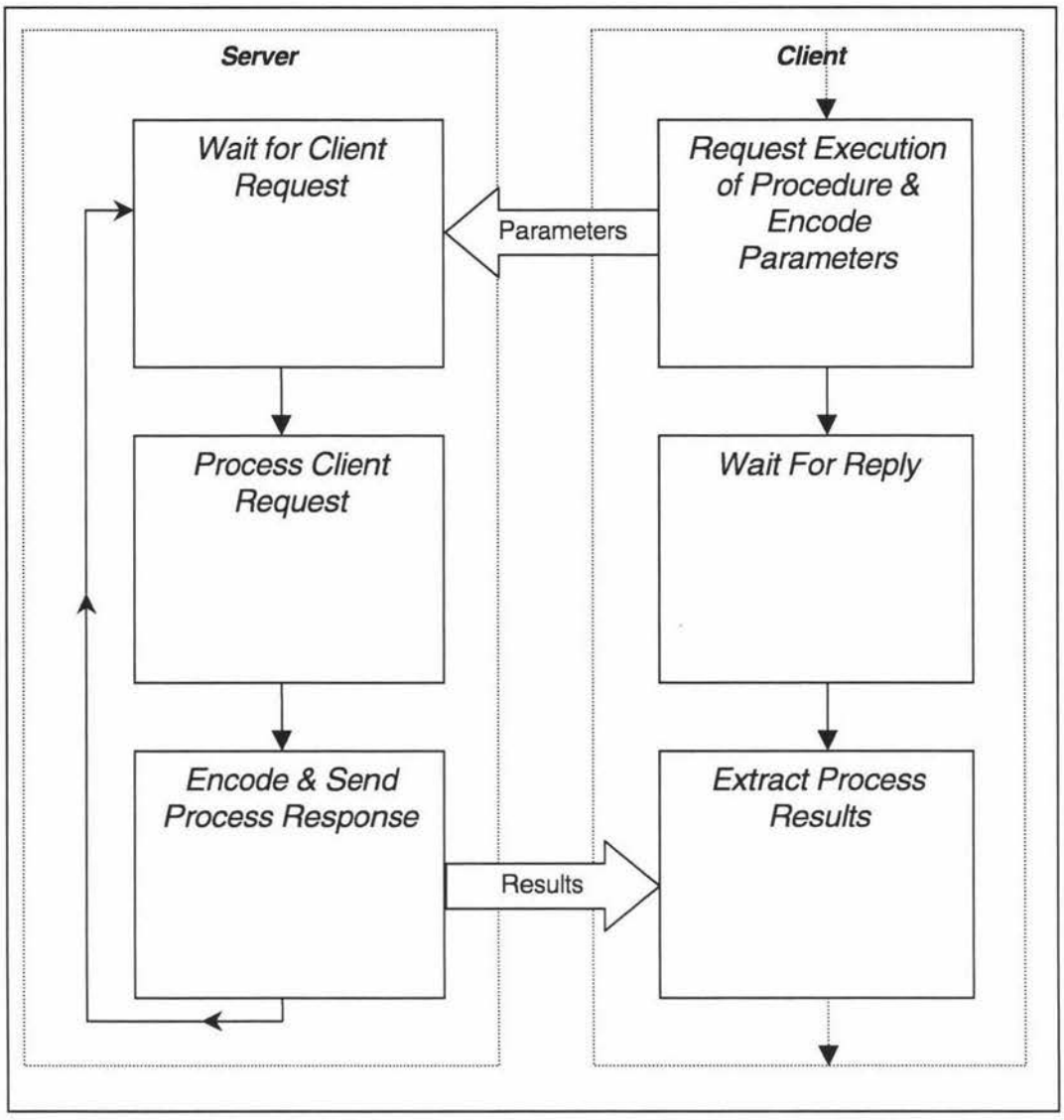
XDR uses a descriptive language to allow designers to describe complex data formats in a simple and an easily understood manner. The RPC command, *rpcgen* has been developed to allow this descriptive language to be converted to C source code. This code can then be incorporated into the RPC application.

(b) RPC

As the database Server and GenaMap are often on separate servers, it is essential that there is a mechanism to allow different systems to communicate and coordinate their activities. One common technique is RPC, which is used in the GenaMap database servers & GenaMap client. RPC is a developed method with several specifications and implementations.[30] The different implementations include: Open Network Computing (ONC) RPC, Distributed Computing Environment (DCE) RPC, and the RPC specification from the International Organization for Standardization (ISO).

This section will describe the concepts behind RPC, and how it was implemented in the GenaMap database server / client software.

[29] RFC1832: XDR: External Data Representation Standard : August 1995 : Sun Microsystems
[30] Comparing Remote Procedure Calls : Sun Microsystems : John Barkley October 1993



• Figure 11 RPC Client & Server Processes

In the RPC model one thread of control logically winds through two processes: the caller's process, and a server's process (Figure 11). The caller process first sends a call message to the server process and waits (blocks) for a reply message. The call message includes the procedure's parameters, and the reply message includes the procedure's results. Once the reply message is received, the results of the procedure are extracted, and caller's execution is resumed.[28]

The server on which the process is to be executed will have a dormant process awaiting a call message from a client. When a call message arrives, this process will extract the parameters, execute the procedure and return the results via a reply message (Figure 11). The process will then return to a dormant state, awaiting the next call message.

A RPC message can be of two types, "Call" or "Reply". Both follow a similar format. Code Fragment 3 is an annotated TCP-IP trace of a "Call" Message, obtained using Microsoft SMS Network Monitoring tool.

```
0030                    80 00 00 48 34 47 74 01 00 00      ...H4Gt...
                       [          XID        ] [Call
0040 00 00 00 00 00 02 00 00 00 00 00 00 00 00 00 00 ..............
           ] [ rpcvers ] [ prognum ] [ versnum ] [
0050 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..............
       proc] [          cred          ] [       verf
0060 00 00 00 00 00 01 00 00 00 18 70 72 6F 70 65 72 .........proper
           ] [      procedure specific parameters
0070 74 79 20 2F 64 61 74 61 31 2F 70 72 6F 70 65 72 y./data1/proper
                    procedure specific parameters
0080 74 79                                          ty............
        ]
```

• Code Fragment 3 Annotated RPC procedure TCP-IP trace

The Code Fragment 4 below details the structure of a RPC.

```
struct rpc_msg {
        unsigned int xid;
        union switch (msg_type mtype) {
        case CALL:
           call_body cbody;
        case REPLY:
           reply_body rbody;
        } body;
    };
```

• Code Fragment 4 RPC Message Structure

The structure of a call message is shown in the Code Fragment 5 below. It contains several flags of which only a few are of use in this application. As the destination server is not using a RPC *port mapper*[31] there is no need to concern the implementation with RPC version numbers, programme numbers or versions - *rpcvers, prog, vers* (Code Fragment 5). The first specific flag for this application to set is the procedure number - *proc* (Code Fragment 5). The GenaMap database server implementation ignores authorisation and verification structures - *cred, verf* (Code Fragment 5), therefore these have not been implemented.

[31] The port mapper program maps Remote Procedure Call (RPC) program and version numbers to transport-specific port numbers. The port mapper program makes dynamic binding of remote programs possible Source: AIX Version 4.3 Communications Programming Concepts

```
struct call_body {
        unsigned int rpcvers; /* must be equal to two (2) */
        unsigned int prog;
        unsigned int vers;
        unsigned int proc;
        opaque_auth  cred;
        opaque_auth  verf;
       /* procedure specific parameters start here */
      };
```

• Code Fragment 5 Structure of RPC Call Message

Code Fragment 6 below details the structure of a reply message. The reply message indicates if the message was accepted or denied. There may be an error even if the message is accepted. If this is the case, the error message will be indicated within the body of the accepted message.

```
union reply_body switch (reply_stat stat) {
      case MSG_ACCEPTED:
         accepted_reply areply;
      case MSG_DENIED:
         rejected_reply rreply;
      } reply;
```

• Code Fragment 6 RPC Reply Message Structure

Section 4.05 Suitability of Java for Implementation of RPC

(a) Java Database Connectivity (JDBC) API

JDBC was developed to allow application developers to write code that is independent of the particular database connectivity mechanism being used. The JDBC API is a standard that provides SQL compliant database systems with a simple and efficient mechanism to integrate their RDBMS with Java.

Unfortunately, the Genasys Database Servers do not provide a full set of SQL functions. For example, any SQL statement can be executed, however the system is unable to obtain the reply (DBEXEC). This limits the usefulness of the JDBC standard. As the mapping between the Genasys Database Servers and the JDBC API was inefficient, it was decided that JDBC was inappropriate for use with this application. By developing only those functions that a typical Java client would require, performance could be increased.

(b) Java Security System

The Java security system is designed to prevent unwanted attacks from malicious applets and applications. Un-trusted applets are typically able to communicate only with the host from which they were downloaded. This creates a problem for database servers that are not located on the GenaMap server. Java v1.2 will help eliminate this problem by introducing standard tools to allow the digital signing of an applet. When an applet is signed it becomes known as a trusted applet, thereby reducing the security "sandbox" (see Section 4.05(c) below), allowing the communication with other hosts.

(c) Java Security Sandbox

Java has developed into an extremely powerful language. During the early stages of development, Java was based around the idea of running software over the Internet.(Section 2.04) Java developers came up with the idea of a "sandbox". The majority of Java applets are contained in a "sandbox". These applets are unable to threaten the security of the machine they are running on. Such operations that are deemed threatening include reading files, writing files, and communicating with other machines. Any operation, which could inflict damage or unnecessary harm to a machine, is forbidden.

During the early stages of development, most systems that ran Java programmes had no mechanisms for "trusting" Java applets. A "trusted" applet is allowed to break some or all of the conditions described above, to work outside the "sandbox". Applets are identified as "trusted" by digital ID's. JDK 1.1 provides the basic technology for loading and authenticating signed classes. This enables a browser to run trusted applets in their environment at the discretion of the user.

The database client API will utilise the code-signing technology. Without the code-signing technology, the client will be unable to access hosts from which the applet did not originate. This means the client can not communicate with a database from a separate database server.

Section 4.06 RPC & XDR Implementation

(a) XDR Implementation

The implementation of XDR for the application was relatively straightforward. The XDR encoding and decoding scheme consists of two classes (Code Fragment 7). Only those sections of the XDR standard that were relevant to this application have been implemented.

The functions that have been implemented cover all of the encoded XDR data that will be encountered. The *XDRLong* function decodes all numeric integer values regardless of the XDR encoding scheme used. (Code Fragment 7)

The *XDRDecoders* class (Code Fragment 7) and *XDREncoders* class (Code Fragment 8) methods have been declared public static, allowing a client to call the methods without instantiating the respective class. This increases the speed of the decoding scheme.

```
public class XDRDecoders
   public static boolean DBStat(DataInputStream is)
   Analyses the status flags on the incoming data stream
   public static String StringPtr(DataInputStream is)
   Decodes the encoded string pointer, returning a string
   public static String XDRString (DataInputStream is)
   Decides the encoded string, returning a string
   public static int XDRLong(DataInputStream is)
   Returns a encoded long number, as an integer
```

• Code Fragment 7 XDRDecoders class Functions

The *XDREncoders* class (Code Fragment 8) encodes data into XDR format. The *XDRLong* method allows the encoding of integers and long variables, by way of type casting during the method call.

```
public class XDREncoders
   public static byte[] XDRString(String encode)
   Encodes a string variable, returning a byte array containing the XDR string
   public static byte[] XDRStringPtr(String encode)
   Encodes a string variable, returning a byte array containing the XDR string
   public static byte[] XDRStringArray
                                   (Vector stringArray)
   Encodes a Vector of strings, returning a byte array containing the XDR string
   public static byte[] XDRLong(long us)
   Encodes a integer value, returning a byte array containing the XDR long value
```

• Code Fragment 8 XDREncoders class functions

(b) RPC Implementation

RPC was implemented in three sections. The first is a RPC Procedure interface class. All RPC procedures are implemented using this interface. The second is a "Call Message", which generates the RPC wrapper[32] for a procedure call. The third is a "Reply Message", which decodes the RPC wrapper.

[32] A wrapper is data that is put in front of or around a transmission that provides information about it and may also encapsulate it from view to anyone other than the intended recipient. A wrapper often consists of a header that precedes the encapsulated data and the trailer that follows it. - Source: http://www.whatis.com

The *RPCProcedure* interface requires 6 methods to be implemented. They are described Code Fragment 9.

```
interface RpcProcedure
{
   public String description();
   Decodes the encoded string, returning a string
   public int procedureNumber();
   Decodes the encoded string, returning a string
   public byte[] details();
   Decodes the encoded string, returning a string
   public int detailsLength();
   Decodes the encoded string, returning a string
   public void processReply(DataInputStream is);
   Decodes the encoded string, returning a string
   public boolean success();
   Decodes the encoded string, returning a string
}
```

• Code Fragment 9 Java RPC Interface Class

This method of developing procedures allows the implementation of further RPC procedures with limited development time. The DBExec function is shown in Code Fragment 10. It executes a SQL command on the server. The GenaMap database server does not enable this command to retrieve results from an SQL statement. The command is commonly used to execute SQL statements like "drop table xxxx", which will remove the table xxxx from the current database.

```
public class DbExec implements RpcProcedure
{
    byte[] detail;
    public DbExec(String exec)
    {
        detail = XDREncoders.XDRStringPtr(exec);
    }

    public int procedureNumber()
    {
        return 0x09;
    }
    public String description()
    {
        return "DBExec";
    }

    public byte[] details()
    {
        return detail;
    }

    public int detailsLength()
    {
        return detail.length;
    }

    public void processReply(java.io.DataInputStream is)
    {
        res = XDRDecoders.DBStat(is);

    }

    boolean res;
    public boolean success()
    {
        return res;
    }

}
```

• Code Fragment 10 DBExec Implementation

Section 4.07 Database Server Security

One noticeable failure in the GenaMap Database Server system is the lack of a true security mechanism. There are no user authentication processes. The closest security system in place is executing a UNIX RLOGIN command to start the database server. However, this process only occurs on a Unix Database Server. If a database server is operating on a Windows NT system there is no RLOGIN requirements and hence, no user control.

Ideally, user access restrictions would be placed on the database server, with some authentication system, as RPC allows for. However, as the RPC server has already been developed, some other form of user authentication system would need to be provided through

the Client API, hence, the requirement for a user database. While this has not been implemented at this stage, the design of the Client API has allowed for this. The underlying classes have been made as restrictive as possible, hence a developer is unable to call the processes, without instantiating the Client API classes, where user authentication process could be placed.

If the database client were to allow the updating of data, the system would need some user authentication process. This is a strong possibility, due to the likely future requirement for a system to be able to upload data from the field.

Section 4.08 Performance

In order to compare the performance between querying a database using GenaMap and using the Java client two methods were used. The first method of comparison was to retrieve data from the database server without processing. The second method was to place the new Java client into the GenaMap Intranet browser. The second method provides a more realistic indication of performance as it is similar to the end-user application.

(a) Retrieving data without processing

This performance test indicates the raw retrieval speed from the GenaMap database server. It is worth noting some of the differences between the way GenaMap handles a *dbfetch* command and the equivalent command from the Java client. The GenaMap *dbfetch* command actually calls the *dbfetchlots* command to provide a buffering effect. However, the only way a GenaMap script can then access this buffered data is by individual calls to *dbfetch*. The Java client places the data directly into a *Vector*, and allows the direct access of this structure, which is a significantly faster method. Code Fragment 11 shows the GenaMap script for the retrieval without processing. Timing for this method was achieved with calls to the system command date command. It is important to note that the system date command is accurate to one second only. Code Fragment 12 details the Java database client testing class.

```
# Start timing
print "$(sys date)"

# Create select statement
tabledesc=$(db select country from countryatt)

# Get the first record
selected="$(db fetch)"

# Continue fetching until complete
while [ "$selected" ]
do
        let count=$count+1
        selected="$(db fetch)"
done

# End the select
db endselect

# print the total retrieved
print "Count: $count"


# End timing
print "$(sys date)"
```

• Code Fragment 11 Genamap Script To Retrieve Data from External Database

There proved to be a significant difference between performance of these two methods (Figure 12). On average the GenaMap script took 12 seconds to complete the retrieval of 2958 rows, compared with the 4.2 seconds the Java client took. This difference decreased as the number of rows retrieved from the database decreased. When only 51 rows were retrieved, there was negligible difference in the time taken. Section I.III contains the results of the database performance tests.

```
// Start Timing
long start = System.currentTimeMillis();

// Create View
DbExec dbe = new DbExec
      ("create view cs as select *
            from countryatt where country like
                                    '%"+args[0]+"%'");
cm.changeProcedure(dbe);
cm.sendMessage();

// Start select
DbSelect dbs = new DbSelect("cs","country");
cm.changeProcedure(dbs);
cm.sendMessage();

// Fetch Data
DbFetchLots dbf = new DbFetchLots(dbs.table);
cm.changeProcedure(dbf);
do {
  cm.sendMessage();
} while (dbf.doMore());

// End Timing
System.out.println("Takes: " +
                (System.currentTimeMillis()-start)*1000);
```

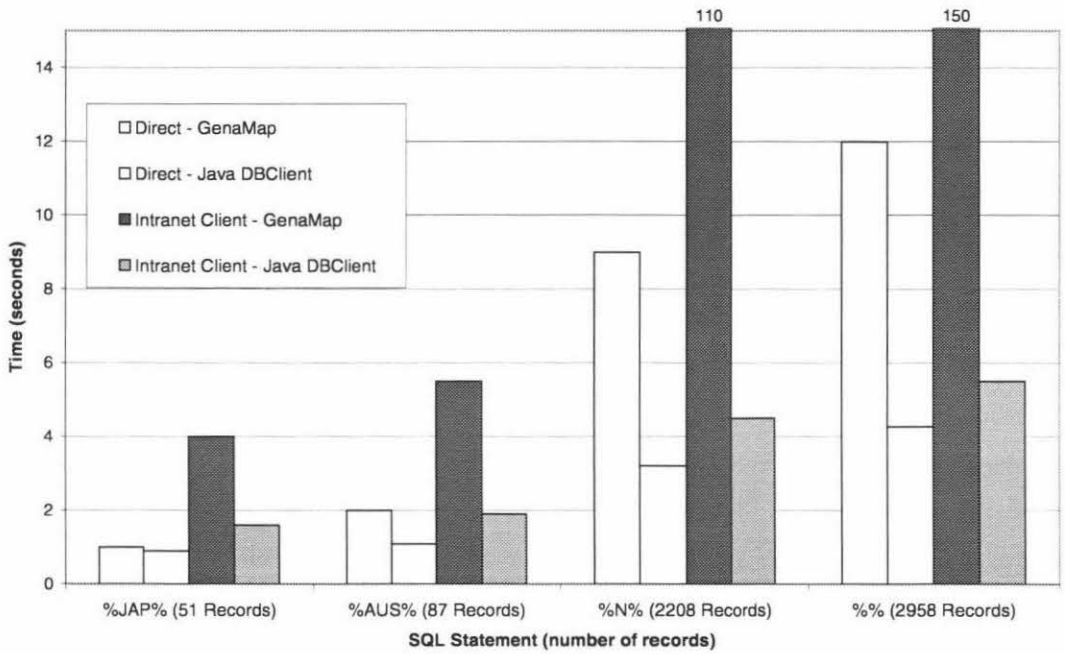
• Code Fragment 12 Java Database Retrieval Test

(b) Retrieving data through Intranet client

The second performance test undertaken was designed to provide a more realistic measure of speed for our target application. As described in Section 4.02, accessing the database server through GenaMap for use in the Intranet client requires several steps. By accessing the database server directly, extra processing of the data is minimized. This test involved measuring the time taken to retrieve data from the database, and display it in a dialog box. This

measurement indicates the total time a user would have to wait, from the time they enter their request until the time they receive visual feedback.

Section I.III details the results of this performance test. From the results it appears that by accessing the GenaMap database server user-waiting time is reduced significantly (Figure 12). For instance if a user was to perform a query that returned 2958 rows, using GenaMap for handling the request would take 150 seconds. In comparison, if the Java Database Client was used to handle the request then the user need only wait 5.5 seconds. The performance for retrieving data through the Java Database client was consistently over 50% faster than retrieving data using GenaMap. In some situations, it was 96% faster.



• Figure 12 Comparison of Database Methods ('%' is a SQL wildcard character)

Section 4.09 Conclusion

The development of a Java client for the GenaMap database servers has proven worthwhile. There have been two main advantages when used with the GenaMap Intranet client; the first being that GenaMap is free from non-GIS processing tasks, for example retrieving a list of data from the server. The second main advantage is a significant increase in speed. The Java client is approximately 2.5 times quicker overall than GenaMap when retrieving data without processing it. The client is 20 times quicker when used with the Intranet client processing the retrieved data.

During the development of the Java client, a RPC server for Java has been developed. This system provides the ability to implement further RPC commands with minimal development time.

It was identified that the security system of the GenaMap database servers was not ideal. It is suggested that the database client include some additional security checking. It will be necessary for the database client to utilise code-signing technology for it to contact database servers that are not resident from the machine the client was downloaded. Previous code signing tools were not cross-platform or cross-browser compatible. *Java2* has implemented a standardised code-signing technology that could be used in future releases.

The database client has been developed to provide data for both textual information and for use with a vector rendering system. When incorporated with a vector rendering system, vector data can be drawn based on data from an external database. For instance, a map of properties could be shaded based on which zoning area they reside in. With the use of the database client, the zoning information could be retrieved from a zoning database not present on the GIS server.

Chapter V. Vector Data Handler

Geographic Information Systems have the ability to place non-spatial data in a spatial context. The database client allows the retrieval of non-spatial data from external databases. However, this data does not contain any spatial information. Vector data from GenaMap provides the spatial context.

Vector data contains mathematical information to describe the image. Typically, this information describes graphic elements like lines and arcs. The more advanced vector formats can handle bezier[33] and b-spline[34] curves.

Vector data in a spatial context is referred to as geographical data. Geographical data revolves around three basic concepts:

* The point feature
* The line feature
* The area feature

Each feature is tagged with a label to identify it in a database, or to indicate its type. This tag could be a unique number, like a SUFI[35] number that is used to identify cadastral[36] features, or a non-unique descriptive label, like "railroad".

A map is defined as a collection of point features, line features and area features. These features are located in a coordinate system and linked to non-spatial data. This ability to tag data and to retrieve information from non-spatial databases, is what separates a CAD system from a GIS system. This non-spatial information can be used to render vector data to indicate the object type it represents. For example, all power-lines could be drawn in a dashed line style as opposed to road centre lines, which could be drawn in a solid line style.

---

[33] A Bezier curve in its most common form is a simple cubic equation that can be used in any number of useful ways. Originally developed by Pierre Bézier in the 1970's for CAD/CAM operations

[34] This is class of piecewise polynomials that is used for curve and surface definition. The B stands for basis. B-splines are mathematically defined using vertex control points, which do not lie on the curve or surface.

[35] SUFI : Static Unique Feature Identifier

[36] The legal definition of a land parcel as opposed to the physical boundary, which could be, made up of natural features like fences and hedges etc.

There are several advantages of Vector Data:

* Accurate graphics (positional) and accurate area calculations
* Pleasing visuals because of retention of detail
* Compact data structure
* Small storage requirements

Section 5.01 Vector Server

The vector server has been implemented following the same design as the Image server, allowing the two servers to be merged into a single server. The vector server is responsible for the following operations:

* Opening and locating Master Project Areas
* Delivering data from GenaMap maps

Three methods of accessing vector data from GenaMap were explored. The first method was downloading data through a GenaMap scripting process. This would provide the easiest method of accessing the vector data. The speed penalty for this ease of access is extremely large. The second method was to download data from the GenaMap files directly. This would require knowledge of the GenaMap file format. In addition, our server would require some spatial knowledge. The third method that was investigated was the use of a GenaMap API.

Two API's were investigated, the first was the GenaMap Spatial API. This API is the lowest level API available to access GenaMap data and provides access to GenaMap functions. This was an ideal solution to our requirements, however the Spatial API utilises the Nutcracker development suite. Nutcracker allows developers of UNIX applications to port their software to Microsoft Windows NT, but suffers performance problems and significant financial outlay.

During the development of this system, Genasys began the development of the Nova Vista software suite. Nova Vista includes a new API, the Accessor Shared Library API. This API allows the retrieval of data from GenaMap.

(a) Vector Data Accessors

"Spatial Nucleus" contains an Accessor Shared Library Interface. The original purpose of this interface specification was to allow Genasys products to access new data sources without the need to recompile the entire Genasys product. The libraries are dynamically linked when they are required. An accessor provides the following functions:

- Connection and disconnection from the data source
- Describe the contents of the datasource
- Query the datasource
- Retrieve the results of the query

There are currently five Genasys developed accessors. They provide access to the following datasources:

- *ESRI* Shape Format
- *Genasys* GenaMap Format
- *Intergraph* DGN Format
- *MapInfo* Interchange Format (MIF)
- *Oracle* Spatial Data Option (SDO)

The primary aim of this system was to access GenaMap map files and retrieve the vector data from these files. With the Genasys shared library specification, this system can access any of the datasources that have a Genasys shared library accessor. This means with limited modification, the system could in theory handle *ERSI* Shape files, *Intergraph* DGN Files, *MapInfo* Interchange Format files etc.

```
typedef struct
    {
        int         version;

        char        *format;

        AC_DB       *(*openDb)(int argc, char const * const argv[]);

        void        (*closeDb)(AC_DB *db);

        AC_DESCRIPTOR
                    *(*describe)(AC_DB *db, const char *dataset);

        void        (*freeDescriptor)(AC_DESCRIPTOR *descriptor);

        AC_QUERY *(*queryRect)(const AC_DESCRIPTOR *descriptor,
                      const struct GeRect3d *rectangle);

        AC_FETCH_STAT
                    (*fetch)(AC_QUERY *query);

        void        (*closeQuery)(AC_QUERY *query);
    }
    AC_FUNCTIONS;
```

• Code Fragment 13 Genasys Shared Library Interface Functions

In order to access the Genasys shared libraries from Java, non-native code must be used. The shared libraries are provided with 'C' source code headers.(Code Fragment 13) These headers are used to access the required functions from the shared libraries. There are several methods of accessing native code from Java. These include platform specific methods from Microsoft. It was decided that the Java Native Interface (JNI) method of accessing native code from Java was the best option for this situation, as the JNI method allows portability of source code. Some of the other methods for accessing native code require rewriting of source code for each vendors' Java Virtual Machine.

(b) Java Native Interface

The Java Native Interface (JNI) allows the programmer to write native code accessible from Java. Amongst other things, the JNI allows:

- Creation of Java objects
- Call Java methods
- Throw Java exceptions
- Load classes
- Perform runtime checking

To access native code from Java, methods are declared with the native modifier.(Code Fragment 14) In addition, they do not contain any Java code. (Code Fragment 14) Code is

provided for the native method from a shared library. When the class is used, the shared library is loaded and the methods are free to be called from Java code.

```
class RetrieveData {

 static {
      System.loadLibrary("rdtj");
 }

public native void setupRetrieve(String mapName);

public native void getData(java.io.DataOutputStream os,
    double minX, double minY, double maxX,  double maxY);
                  }
```

• Code Fragment 14 Example of accessing native code from Java

The construction of the native method library was undertaken in *Microsoft Visual C++*. The Java Development Kit contains the *javah* command to produce header files with the required method identifiers. Dynamic linkers resolve entries based on these identifiers. (Code Fragment 15) The method identifier is generated from the following components:

- the prefix `Java_`
- a fully-qualified class name
- an "_"
- method name

```
JNIEXPORT void JNICALL Java_RetrieveData_setupRetrieve
                        (JNIEnv *, jobject , jstring);

JNIEXPORT void JNICALL Java_RetrieveData_getData
     (JNIEnv *, jobject, jobject, jdouble, jdouble,
                                    jdouble, jdouble);
```

• Code Fragment 15 'C' Native Code Headers

There are various methods available to return the data provided by the accessor to the Java client. One option involves returning individual values back from the native method to Java code, and then sending the data from the Java server to the client. However, the JNI system provides an option of transferring data directly from the JNI native code to the client. A Java method can be called from the native code, writing data directly to the client via a Java data output stream. (Figure 13)

Three data types are returned from the native code; doubles, integers and strings. By passing a reference to the *DataOutputStream* the methods *writeDouble*, *writeInt* and *writeString* can be called to transfer data from the native code directly onto the Java stream to the client.

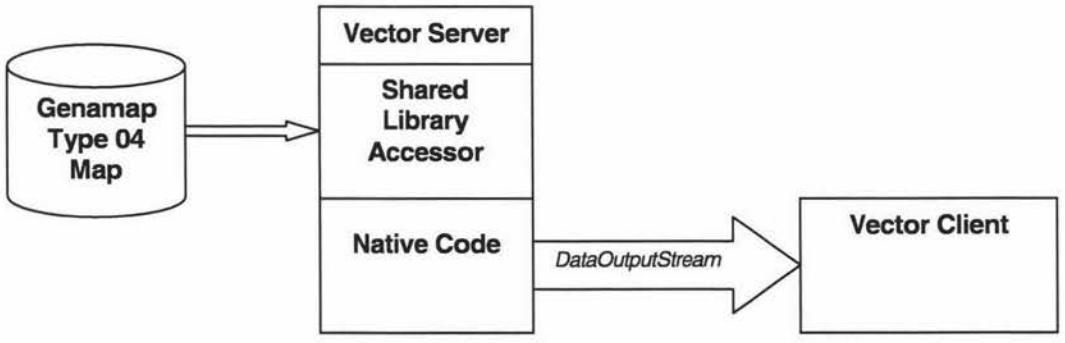
(c) Vector Server

The Java Native Interface provides a method of retrieving vector data from the Genamap data format. The remainder of the server is implemented in Java. The server is multi-threaded, in theory enabling maps to be opened simultaneously for transfer to a client.

One of the deficiencies in the Genasys shared library accessors is their inability to handle multiple connections to the same map, they are not multi-threaded safe. It will be necessary for these accessors to become multi-threaded safe for this system to work efficiently. Currently if one person was to open a large map, which took 3 minutes to open, no other client can access that map until the original client has completed. In addition, if an accessor is simultaneously accessed then the system crashes.

In order to prevent the system from crashing it must block multiple uses of the same accessor. The server achieves this by maintaining a *hashtable* of accessors and the maps they represent. When a client needs to access a map, it calls the appropriate accessor for that map. If the accessor does not exist then the system creates a new accessor and places it in the *hashtable*. If the accessor does exist, the system retrieves it. It then checks if the accessor is in use, blocking the client from using the accessor if it is currently in use. When it becomes available, the block is removed.

To retrieve data, the system calls the accessor to provide a region of a map to the client. The accessor then searches the map file and returns all objects from within the region of interest. The accessor is able to provide a range of information from the maps. The server requests only information required by the client, speeding the retrieval of data. (Figure 13)



• Figure 13 Vector Server and Client Communication

Section 5.02 Vector Client

The following requirements were identified during the design of the Vector Client:

- Speed of download
- Incremental drawing
- Usability while downloading

Initial investigations into the download speed showed that the *java.io.** package[37] of classes suffer significant performance penalties. This was mainly because of the use of synchronized methods. (See Appendix I) It was evident that some custom form of buffered input stream class would be required.

In order to provide incremental drawing of the vector data, the system is multi-threaded. The first thread controls the fetching and downloading of the data, while the second thread controls the rendering of the data to screen. By having two separate threads, the downloading and rendering can happen asynchronously. This also allows the user to remain in control of the system while the data is being downloaded and rendered. For instance, if the user decided that the current region of interest was too large, they could zoom in, and the appropriate data would be rendered to the new region of interest. Similarly, if the user decided that the current region of interest was in the wrong area they could zoom out or pan to a new region of interest. The system will continue to download the original data and provide the new region of interest.

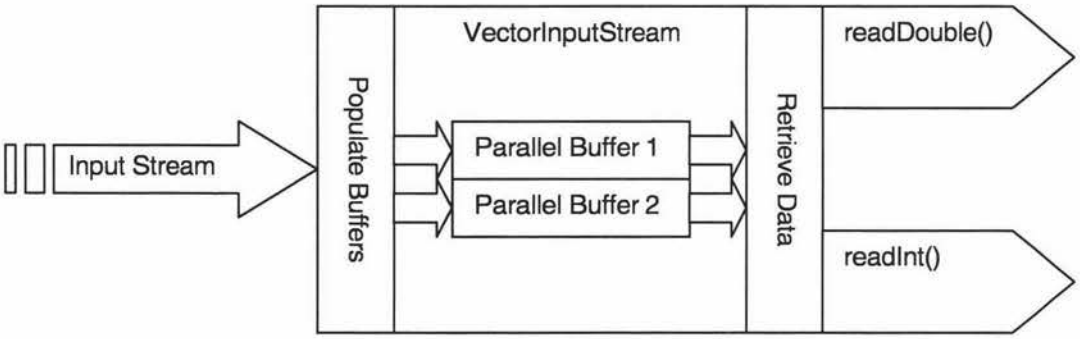
(a) Vector Input Stream

The Java buffered stream reader *(java.io.BufferedInputStream)* suffers significant performance problems. The read method is marked as synchronized, to allow for multi-threading. A synchronized method call in Java is shown to be 10 to 100 times slower than an unsynchronized method call (Appendix I). A second problem is the number of array copies taking place. These deficiencies mean that the throughput over a 10Mbit/second network connection is between 90 to 120 Kilobytes per second, in comparison to the typical transfer rates of 800 to 900 Kilobytes per second. It is evident that the *BufferedInputStream* class is unusable for copying large amounts of data across the network.

There were two design objectives for the custom *VectorInputStream* class. The first was that it would implement only those functions required for the current use, thereby limiting the size of the class. The second design objective was to make the class fast, by eliminating the synchronized methods, and by eliminating the array copy calls.

[37] A Java Package is a collection of Java class files, the base Java classes are arranged into packages, like *java.io.**

The *VectorInputStream* design revolves around two parallel arrays.(Figure 14) These arrays are switched between and populated in turn. The *VectorInputStream* has been optimised to handle Java double and integer values.



• Figure 14 VectorInputStream structure

The reader uses two threads; the first thread fills the parallel buffers as data arrives and as space is available in the buffers. The second thread decodes the data from the buffers. This results in exceptional performance. In a comparison between the *java.io.BufferedInputStream* class from *Java2*, and the *VectorInputStream*, the later was shown to be 58% faster when operating on a single machine, with the *BufferedInputStream* transferring at 552 kilobytes per second and the *VectorInputStream* transferring at 874 kilobytes per second. When operating over a 10 MegaBit/second network the *VectorInputStream* was shown to be 139% faster. The *BufferedInputStream* transferring at 157 kilobytes per second, peaking at 190 kilobytes per second. The *VectorInputStream* transferred at 374 kilobytes per second, peaking at 583 kilobytes per second.

The *VectorInputStream* provides a significant improvement over the standard Java buffering classes. By eliminating the copying of arrays and the synchronized methods, the class operates at an improved speed.

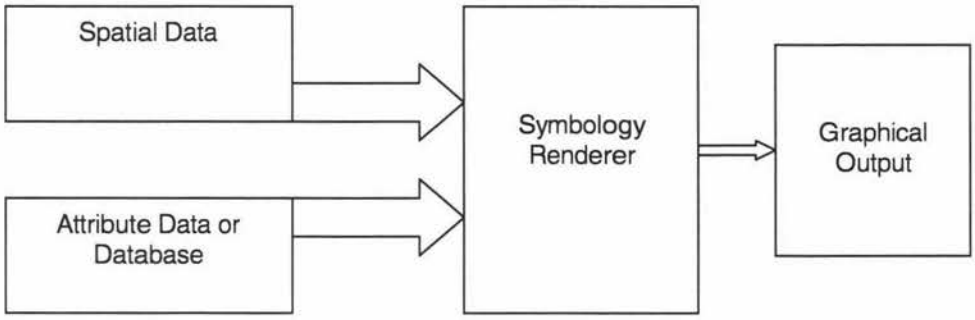
(b) Incremental Drawing

Incremental drawing is handled by a separate thread. The *VectorDrawer* class has a reference to the VectorObjects data collection that contains the vector data. It sleeps until there are new VectorObjects in the data collection. When new objects are placed in the collection, the thread resumes execution and calls the VectorObjectHandler draw method. This draws each object at the required location.

The system uses a double buffering technique. The draw method is called with a Graphics context for an off screen image. When an update occurs on the offscreen image, it is drawn to the output screen.

Section 5.03 Symbology

Symbology is the rendering of vector data based on a variable. It is the process that combines attribute and spatial data. (Figure 15) A symbol can be rendered based on an internal attribute or a value from an external database. Genasys have recently developed a new symbology system. The symbology is assigned to map features based on named procedures stored in an ASCII text file. The symbology procedures support drawing attributes including color, line styles, fill styles, and line thickness. In addition, the symbology system supports arcs, filled features, and bitmap data. Symbols can contain mathematical expressions and can be passed parameters to be used in the symbol rendering process.



• Figure 15 Symbology rendering based on attribute and spatial data

(a) Symbology Procedures

Symbology procedures control how vector data is rendered. This involves the modification of the current graphics context. For instance, the graphics context can be modified by changing the size of rendered lines, the style of the lines or some other feature. Once the graphics context is modified to the desired settings, the path of the symbol is rendered. There are several graphics primitives available for use in symbol procedures, these are listed in Appendix IV.

A symbol may contain a large number of primitive graphics operations, in addition to these standard operations, each symbol may call another named symbol. This allows the reuse of symbols, making the system extremely powerful.

There are three types of symbols. The first type is a marker symbol, drawn around a single co-ordinate. Marker symbols are typically used to represent objects, like manhole covers and the like. The second type of symbol is line symbology. This type of symbol is used to represent a series of line and arc segments. Line symbology is often used to represents objects like power lines, gas lines and roads. The final type of symbology is fill symbology, which can be used to fill a closed polygon. This can be used to shade areas, like zoning information of a land area.

(b) Implementing Symbology in Java

The *SymbolManager* class finds each procedure from the symbol source file. It calls the Symbol class to create each symbol, storing the result in a *hashtable*. The *SymbolManager* is also called to draw a symbol given the symbol name. It searches the *hashtable*, and then calls the symbol to draw itself at the appropriate location.

The Symbol class is called by the *SymbolManager* with the symbols' source. The Symbol class contains a vector of symbol procedures. These *SymbolProcedures* represent graphics primitives. If there is no *SymbolProcedure* for a procedure that is called in the symbol source, then it is assumed to be another named symbol being called. The availability of this symbol to be called is checked at runtime. The *SymbolProcedure* interface is implemented with graphics primitives to set the graphics context to the appropriate state, or to render sections of the symbol. These *SymbolProcedures*' map to the graphics primitives listed in Appendix IV.

The system described above maps extremely well to the *Java2D* API, and the symbology code could be modified to work with *Java2D*. However, until the symbology system is modified to work with the *Java2D* API, and until Genasys specify how users are better able to utilise the symbology system, it will not be included in the *VectorHandler*.

Section 5.04 Java 2D API

The *Java2* release saw the major redesign of the Java graphics system. Previously the graphics system allowed a user to draw graphics primitives. However, there was no control over line width or line styles. *Java2D* has been designed to increase the usability of the Java graphics system. The main advances with the *Java2D* API are:

- Colors and patterns: gradient fills, fill patterns from tiled images, transparency
- Local fonts
- Pen thicknesses, dashing patterns, and segment connection styles
- Coordinate transformations

With previous versions of the Java *AWT*, a shape was drawn by calling the *drawXxx* methods, for example *drawLine*, *drawArc* etc. To fill a shape, the *fillXxx* methods were used. With *Java2D*, it is still possible to call *drawXxx* and *fillXxx* methods, but a more powerful method for drawing has been created. *Java2D* requires the creation of a *Shape* object, and then a call to either draw or fill that object. (Code Fragment 16)

```
// Assume x, y, and diameter are instance variables
  Ellipse2D.Double circle =
    new Ellipse2D.double(x, y, diameter, diameter);
  g2d.fill(circle);
  ...
```

• Code Fragment 16 Elipse2D shape object used as a filled shape

Various shapes are provided with *Java2D*, and it also allows the developer to create their own. The shapes that are included are:

- Arc2D
- Area (a shape built by adding/subtracting other shapes)
- CubicCurve2D
- Ellipse2D
- GeneralPath (a series of connected shapes)
- Line2D
- Polygon
- QuadCurve2D
- Rectangle2D
- RoundRectangle2D

With previous versions of Java, the *drawXxx* methods rendered solid, 1-pixel wide lines, and it did not have the ability to change the line style or line size. However, *Java2D* allows the setting of pen thickness, dashing patterns, and the pen colour or pattern. In addition, it allows the specification of how line segments are drawn, the shape at the end of the line, and how lines are joined together. The *BasicStroke* object is passed to the *setStroke* method to control these features. To fill a shape, the user now has the option of solid colours, gradient fills, textured fills, or a user defined paint.

A major improvement in the *Java2D* API was the addition of transparency to the graphics context. Previously, transparency was only available with GIF and JPEG images. *Java2D* now permits transparency values to be assigned to drawing operations. The inclusion of transparency allows the layering of images and vector data.

Java2D also includes the ability to rotate and scale the coordinate system and hence the symbol. This eases the task of rotating object like arcs etc. The initial system for rotating

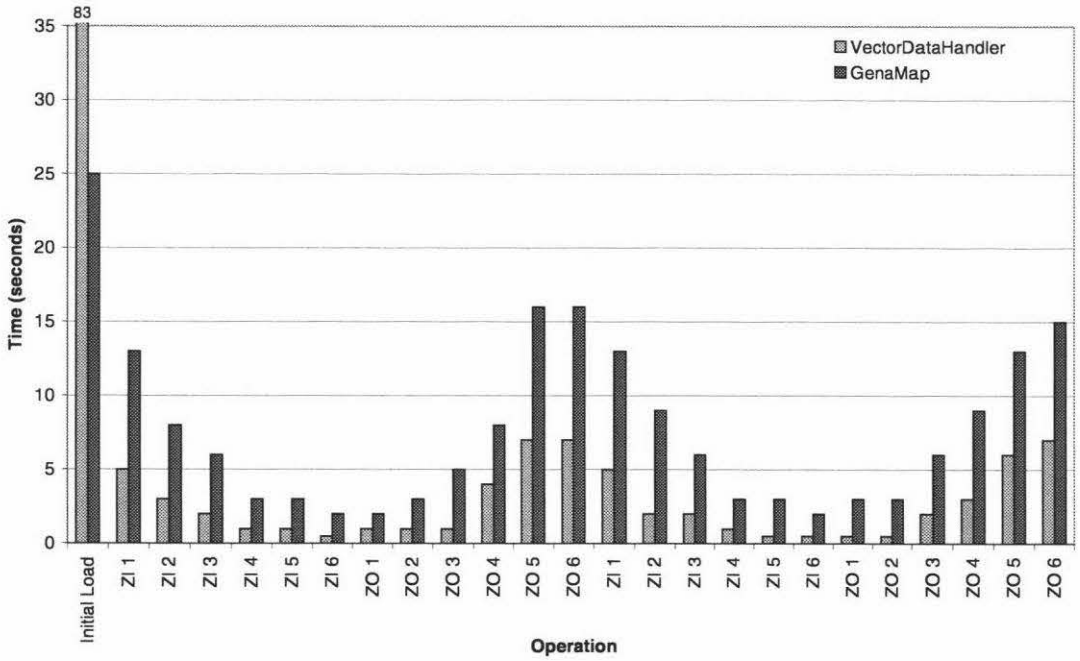
symbols was developed in *Java 1.1.4*, and while functional, they suffer performance problems. These symbol classes will be modified to operate under the more efficient *Java2D* system.

Section 5.05 Performance

A comparison was made to GenaMap to measure the performance of the *VectorDataHandler*. As with the image handler, measurements were taken of the time taken to zoom in on a specific point, and then the time taken to zoom out. In addition, the time taken to pan half a screen width and the time taken to pan a whole screen width were measured. These measurements were made on a 25 MB cadastral map.

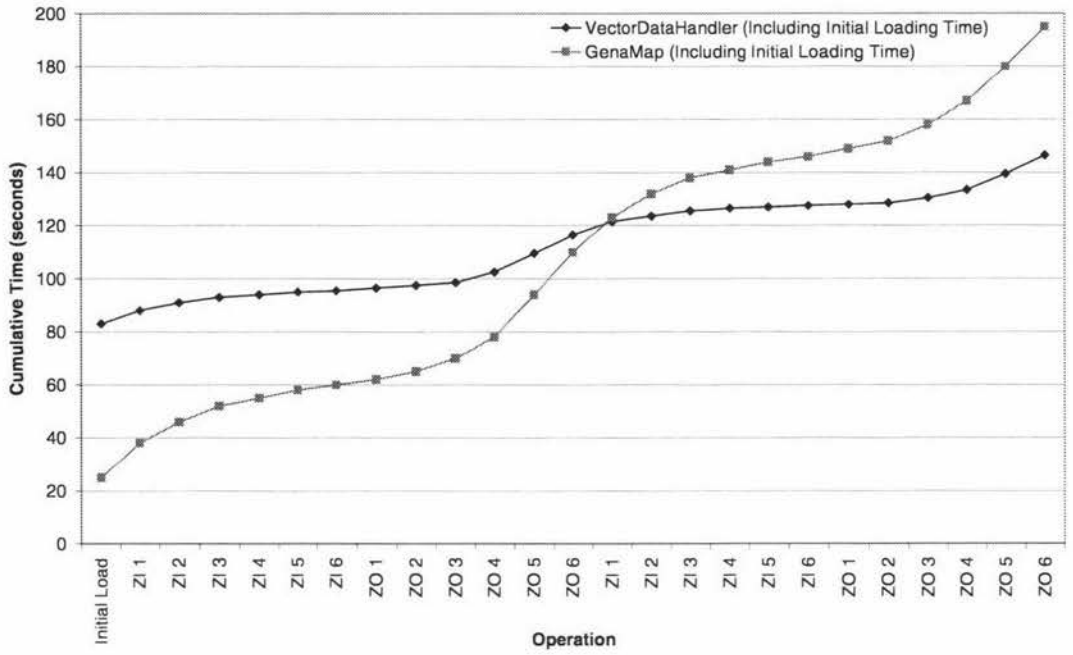
It is evident from Figure 16 that the majority of the vector data handlers' time is spent on the initial loading. The initial loading time is a consequence of three factors. The first is the speed of the TCP-IP socket connection. To transfer 25 MB of data over the connection takes approximately 28 seconds, if transferring at 874 kilobytes/second. (Section 5.02(a)) However, data is not provided at this speed from the Genasys shared library accessors. To retrieve the dataset using the shared library accessors takes, 35 seconds. The third factor is the calling of Java methods from *"C"* native code.

Beyond this initial loading time, the vector data handler is significantly faster than GenaMap. (Figure 18) On average, the vector system is 55% faster for each individual operation than GenaMap over 25 operations (see data in Section I.V).

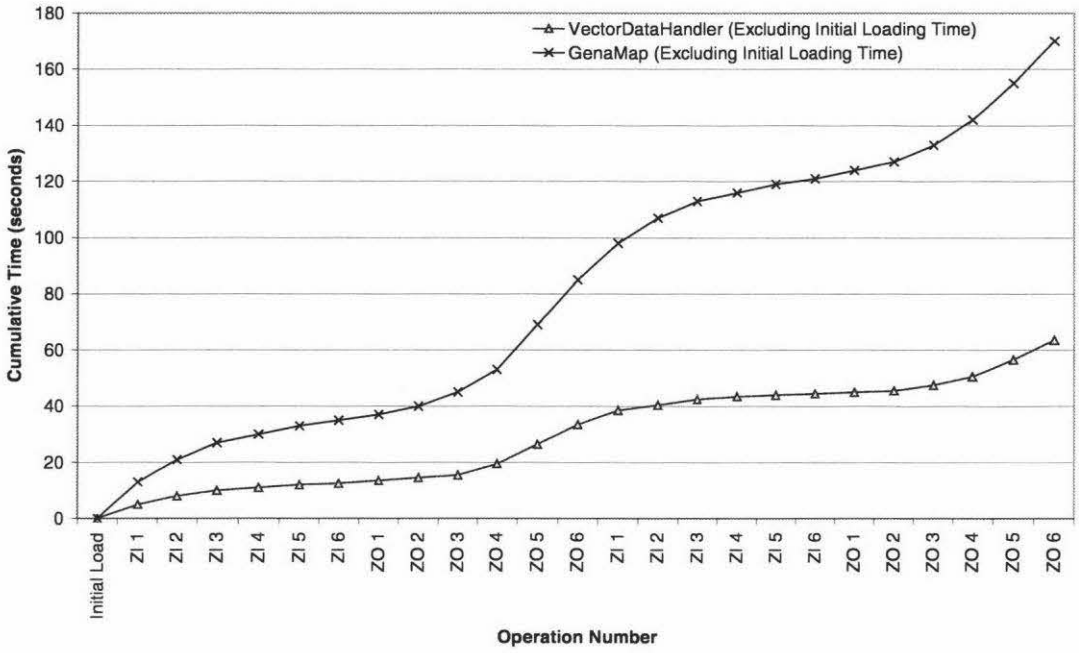


• Figure 16 Vector Data Zooming and Panning Operation Times(ZI=Zoom In;ZO=Zoom Out;PA=Pan)

The system is 25% faster overall (for 25 operations), representing over 50 seconds. (Figure 17) Beyond the 14th operation the system had taken less time overall than GenaMap. (Figure 17) This indicates that for users who perform a few operations only, the GenaMap or MGIS system would be quicker. However, for users who perform more than 14 operations the vector client is more efficient. (Figure 18)



• Figure 17 Vector data cumulative zooming and panning times, including initial loading time(ZI=Zoom In;ZO=Zoom Out;PA=Pan)



• Figure 18 Vector data cumulative zooming and panning times, excluding initial loading time (ZI=Zoom In;ZO=Zoom Out;PA=Pan)

Section 5.06 Conclusion

The vector data handler was designed to minimise time waiting for vector data. The system utilises the shared library interface specification developed by Genasys to access the map data. The shared library system does not allow simultaneous access of the same map, limiting the multi-user performance of the vector server. It is expected that this limitation will be removed in the next release of the vector data accessors. The system out-performed GenaMap at all stages beyond the initial drawing stage, being over 25% quicker in a standard set of 25 zooming and panning operations.

The vector handler system has been designed to link to the database client. This will allow the system to utilise the symbology system. When used in conjunction with the large image handler, the system can provide excellent performance. The next section details the integration of the vector, image and database clients.

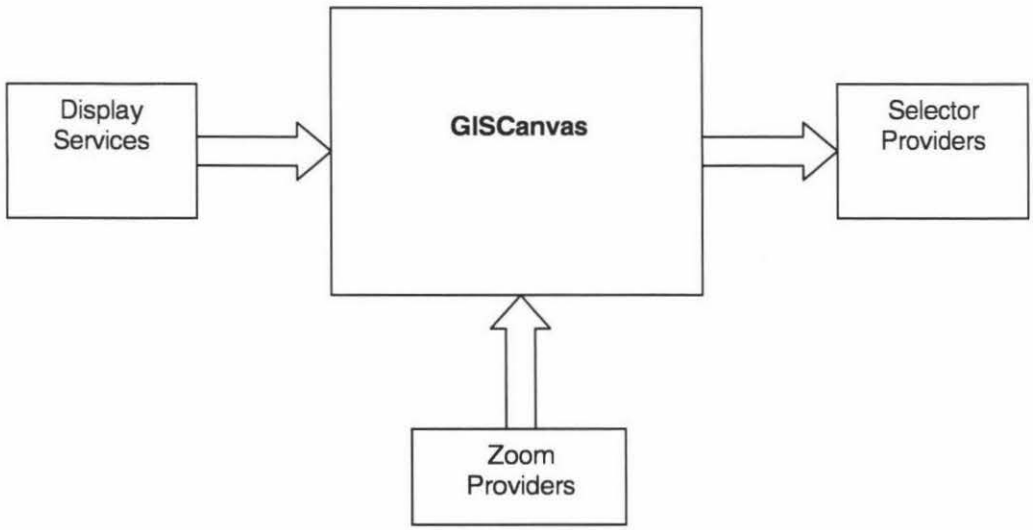
Chapter VI. System Integration

The system has been developed in three distinct stages, the large image handler, the database client, and the vector data handler. This section will detail the integration of those three components for the Java client. In addition to these components, a component that allows the execution of GenaMap scripts was also developed. It executes scripts on the server by using the Genasys *Request Broker* interface.

Section 6.01 Client integration

(a) Display Services

To integrate the Vector provider and the Large Image provider an interface was created. This interface was designed to allow a central class, the *GISCanvas,* to call various display services for images. At this stage, there are two display services, the vector data handler and the large image handler. (Figure 20) The system has been designed to allow the easy addition of future display services. Such display services could include a class to display textual information on the map which is provided by the database client.



• Figure 19 Interacting Components with GISCanvas

Classes that implement the *DisplayService* interface must include the following methods:

- ***ProvideImages***

This method calls the Display service to retrieve data required for rendering and to initiate the rendering process. *ProvideImages* is called by the *DisplayServicesThread*. This thread allows the multiple display services to retrieve data simultaneously.

- ***DrawImages***

This method calls the display service to render any data currently available to the graphics context. This method is called multiple times, updating the display as new data is downloaded.

- ***ProvideElements***

This method provides a Vector of objects within a specified region of interest. This method is used by an object selection class, like the selector provider.

- ***SetEnabled***

This method allows the display service to be disabled or enabled. This is called by the layer manager to handle the toggling of the display of each layer.

(b) Selector Provider

The *GISCanvas* class controls the various display services. It maintains the link between any zoom providers, which may be a navigation tool or a standard zoom manager, and the display services. The *GISCanvas* also provides a link between the *SelectorProviders* and the display services. These selector providers can be either an enquiry tool or a selection from a navigation provider. They call the *provideElements* method of each display service, retrieving a vector of elements that they can utilise.

(c) Zoom Provider

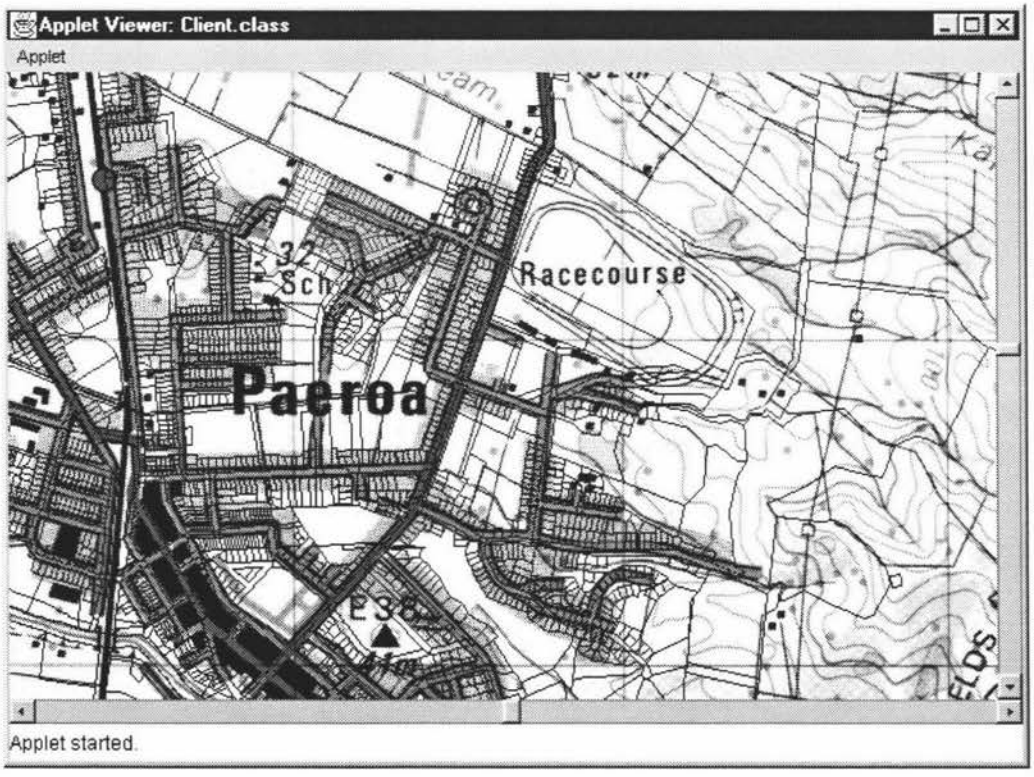
To modify the current region of interest, a user typically draws a rectangle on the canvas. This indicates the new region of interest. The canvas bounds are then modified to reflect this area and each display service is called to regenerate their display, downloading any new data required. However, this is not the only method for modifying the current region of interest. Performing a "navigation", as described in Section 4.02 also requires such a modification. In order to handle the variety of methods that can modify the region of interest, an interface was created named *ZoomProvider*.

Section 6.02 Client User Interface

Two interface modifications were trialed with this project. Their origins were from popular image viewing tools. Previous GIS clients supported zooming with the client drawing a box around a region of interest. The client then zoomed to this area and regenerated the display. Panning is handled by dragging the current image to a new location, the image is then redrawn. The following interface modifications were trialed:

- Zooming with single clicks; a user can click the left mouse button on the current image, the client then zooms in 50% on the location of the mouse click; the right mouse button operates in a similar fashion, but zooming out by 50%. These zooming ratios, match the dyadic nature of the image pyramid system (Section 3.02).

- Panning with scroll bars; a user can pan in all four directions using scroll bars. Dragging the button scrolls the screen to the absolute location, clicking the arrow pans half the current screen width in the desired direction; clicking the scrollbar between the arrow and the bubble pans a whole screen width in the appropriate direction. (Figure 20)



• Figure 20 Integrated vector data handler and large image handler with scroll bar interface.

Section 6.03 GenaMap scripting access

The implementation of an entire GIS system is a major task. It was identified that the client system required the basic zooming and panning functions, some selection capabilities, and several navigation tools. There is little need to provide many of the more advanced GIS topology analysis tools. However, it is desirable to have the ability to provide this functionality.

The simplest method for obtaining this functionality was through a link to the GenaMap scripting environment. The Request Broker architecture (Section 2.03(b)) allows the execution of scripts via the GenaMap scripting environment. The system is then able to return textual results, or create an X-window, draw to the window and capture the result as a GIF file. This is a similar system to the GenaMap *Web Broker* environment (Section 2.03(c)). It was decided to incorporate the scripting functionality using the *Request Broker* interface rather than using the *Web Broker* scripting capability, as the *Web Broker* system is unavailable on Windows NT.

The *Request Broker* architecture is described in detail in Section 2.03(b). The system accesses the 'C' functions by the Java Native Interface, described in Section 5.01(b). Four functions are available to the client:

- Create X-window pixmap
- Destroy X-window pixmap
- Capture X-window pixmap to GIF
- Execute GenaMap Script

These functions allow the full utilisation of the GenaMap environment. For instance, the system is able to execute advanced rendering functions, capture the resulting display via an X-window and render the image amongst the Java vector data and image data.

During the early stages of this project, the scripting functionality was incorporated into the large image handler server to provide a fully functional Windows NT server for GenaMap.

The scripting component has been combined with the large image handler and installed at the Buller District Council. The Buller District Council are using this system successfully for a number of users.

Section 6.04 Conclusion

The vector data handler and the large image handler have been integrated using a generic software interface that allows easy integration of future components. These future components could include services that provide textual information, or display data obtained from

GenaMap. In addition, two user interface modifications were trialed for this project, single point zooming and panning using scroll bars.

A link to allow the execution of GenaMap scripts has also been included. This permits the system to perform advanced GIS functions. Using the display service interface, a GIF image could be returned and integrated with the vector and image handler data. This system is in active use at the Buller District Council.

Chapter VII. User Evaluation

Each section of the client system has undergone extensive performance testing. However, certain sections of this system have included functions that keep a user informed while a process is being undertaken, like incremental drawing. This is designed to increase the perceived speed and user appreciation of the product. These two factors cannot be measured by raw performance figures, and require user evaluation to determine the subjective performance of the product. This section details the user evaluation of the three systems that have been designed, the large image handler, the vector data handler and the database client.

Section 7.01 Evaluation Description

The evaluation involved asking the test subjects to perform a variety of simple tasks, utilising the navigation features of the client. Each system was tested separately. When they had performed the set tasks, they were then permitted to use the system for a further ten minutes to gain an understanding of "use" and "feel".

For the image and vector components the users were asked to:

- Zoom into a specific area of interest, using the incremental zoom option
- Zoom out from the specific area
- Follow the coast line using the pan function
- Zoom directly into a specific area
- Pan in all directions

For the database components, users were asked to rate the performance of the Java Database Client compared to the GenaMap system in the Genasys Intranet Client.

Three groups of users (two users in each group) were presented with the test and interviewed. They represented power-users, operational users and casual users (as described in Chapter I). All users were from the Hauraki District Council, a user of the MGIS Intranet browser.

Section 7.02 Evaluation Results

The users were asked to rate questions on a scale from 1 to 10. Ten represents the most desirable solution, with one indicating the least desirable. For the comparisons, 0-4 represents that the previous solution was better, whereas 6-10 represents the new solution with five indicating no difference. The following questions were used for the evaluation:

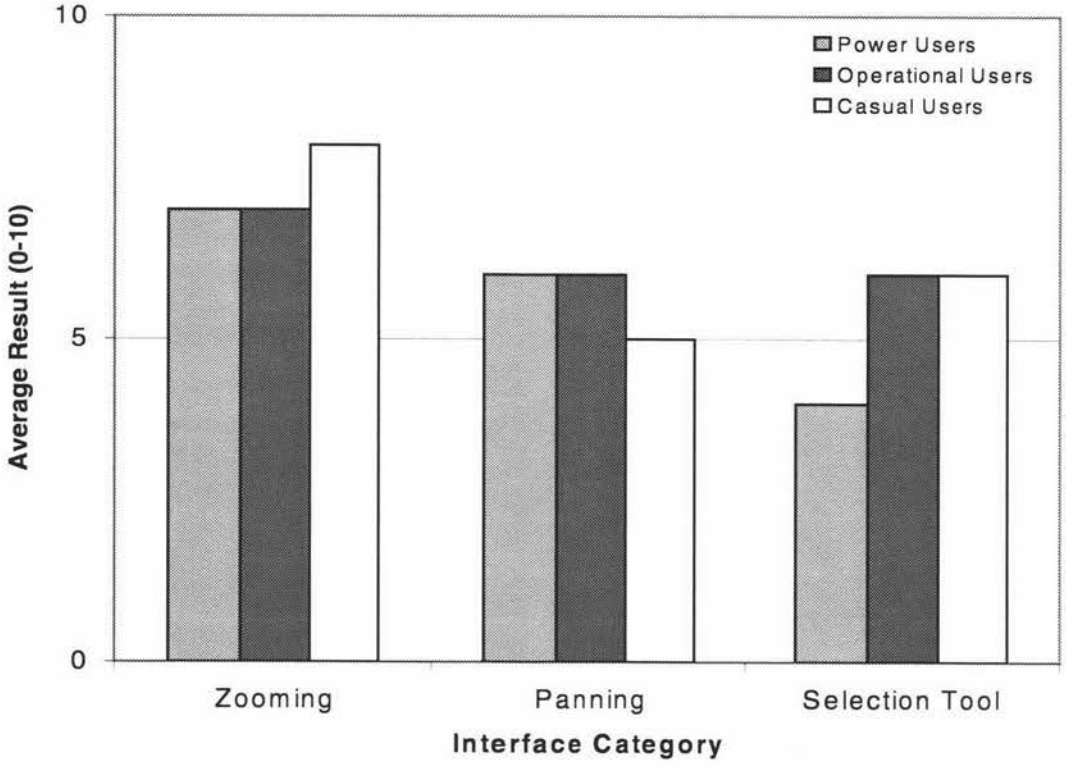
1. Rate the functionality provided for zooming
2. Rate the functionality provided for panning
3. Rate the functionality provided by the selection tool
4. Rate the selection tool speed

The following questions were asked for both the image handler and the vector data handler:

1. Rate the responsiveness to zooming and panning
2. Rate the startup and loading time
3. Rate the visual quality of display
4. Overall impression of the component
5. Comparison of the component to the MGIS system

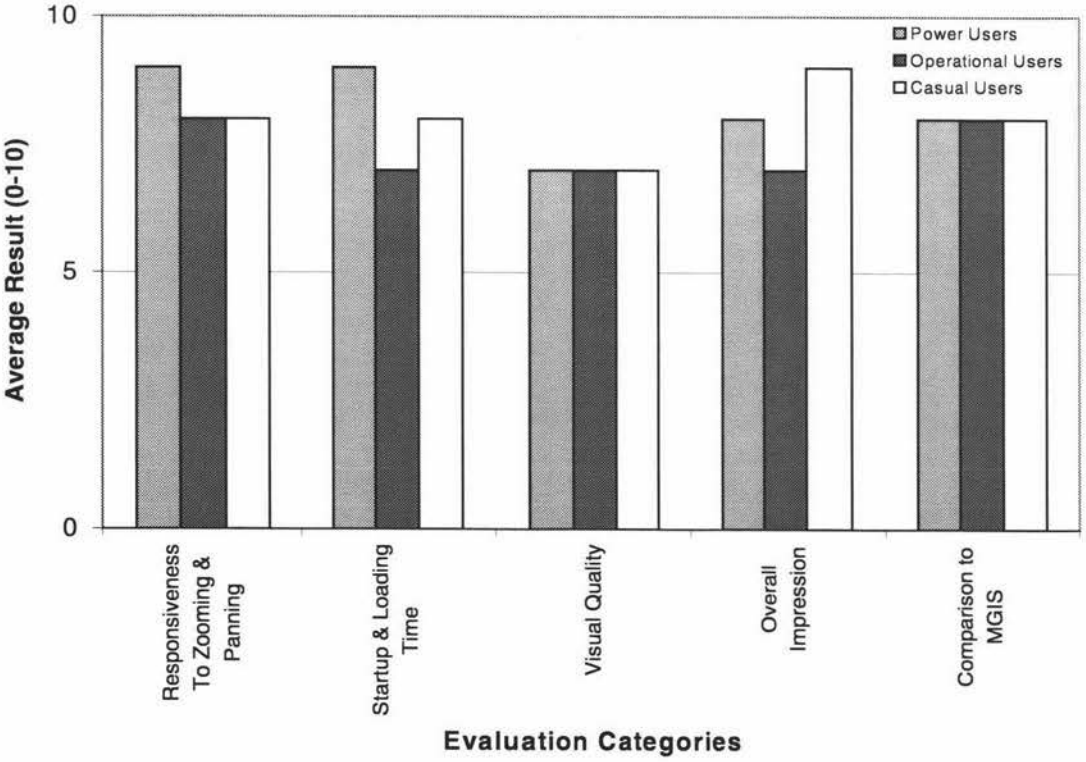
(a) Interface Evaluation Results

Overall, the users were happy with the interface tools for zooming, panning and selection that were provided. (Figure 21) They commented that the zooming tool provided worthwhile additional functionality. However, they did not view the method of panning to be of greater use than the previous method. They felt that the panning tool limited their ability to follow features in a diagonal direction, where the previous panning method allowed this. The selection tool was viewed to be a worthwhile addition by most. (Figure 21)



• Figure 21 User Interface Functionality Results

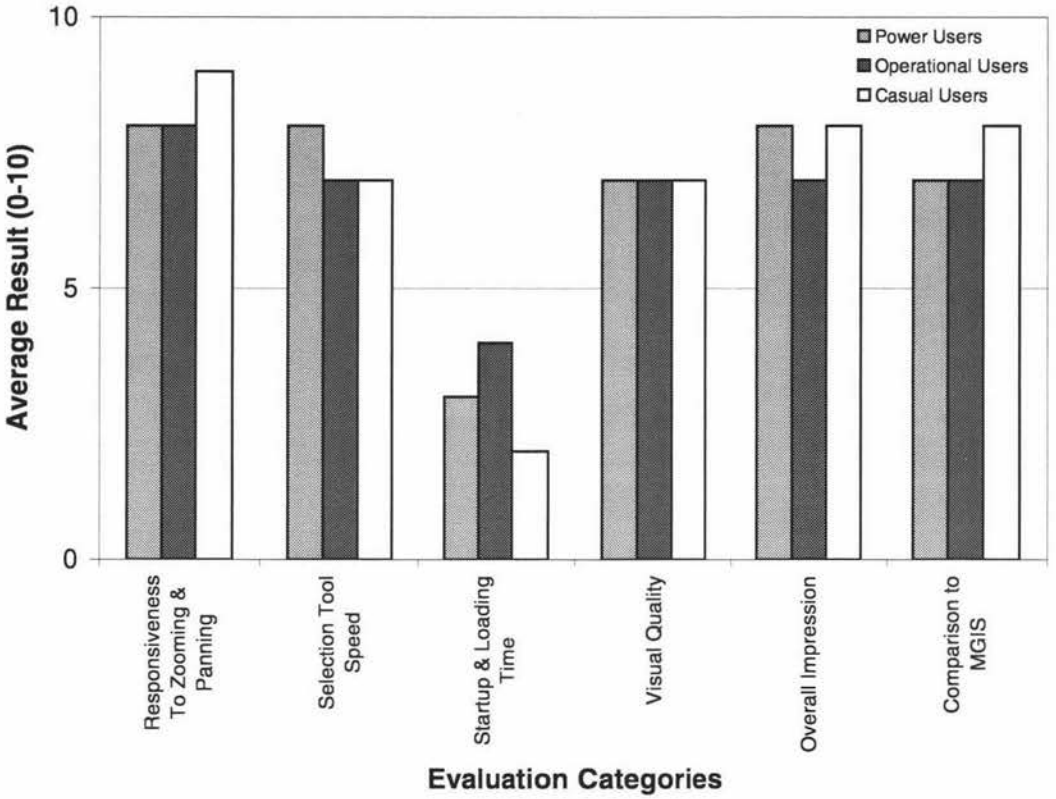
(b) Image Handler Evaluation Results



• Figure 22 Image Handler Evaluation Results

The image handler scored well in the evaluation results (Figure 22). Users commented on its responsiveness and startup/loading time. However, when the garbage collector operated users noticed the decrease in performance.

(c) Vector Handler Evaluation Results



• Figure 23 Vector Handler Evaluation Results

The vector handler scored well on all questions, except the initial startup and loading time. (Figure 23) However, it was viewed to be extremely responsive system beyond the initial loading time. Overall when compared to the MGIS system the average score was over 7/10, indicating a high subjective performance rating. Similar scores were obtained for the selection tool, scoring over 7/10 or better. (Figure 23)

Section 7.03 User Comments

The users were interviewed about each component. The following comments were made by the users during this informal discussion.

(a) Image Handler

"... the images are provided a lot faster than is available in the current MGIS system, if we had that functionality we would probably navigate using the images and then turn the cadastral on."

"....good performance when moving around whole image, got a little slow at one point, but then after a while it got up to speed again."

(b) Vector Data Handler

"Vectors took a long time to fully load, but once they were loaded the performance was a lot better than what the current system is."

"We wouldn't want to load the whole cadastral if it takes that long to load, would load some road lines, or images and zoom to the appropriate area, before turning them on."

(c) Database Client

"Wow"

Section 7.04 Conclusion

The user evaluation illustrated that users viewed the system to have increased performance in comparison to the MGIS system. In all cases, except the startup and loading time of the vector data, they indicated that the system was more responsive and faster.

Of the changes to the user interface, the zooming was rated as an improvement, while the panning using scroll bars and the selection tool were of limited use.

Overall, the test evaluation subjects indicated that they would like to see the technology they evaluated included into future GIS Intranet browsing tools.

Chapter VIII. Project Difficulties

Section 8.01 Java

During the course of this project, Java has undergone significant changes. Three major releases of Java, and several minor releases have occurred during this project. As Java is a developing language, it can be expected that some API changes will be experienced. However, during the three major releases of 1.0, 1.1 and finally Java2 the following systems have undergone significant overhauls:

- Abstract Windowing Toolkit (AWT) Event Model
- Graphics toolkit, to create Java2D
- Java Native Interface
- Java Advanced Imaging Toolkit
- Security Subsystem

The result of these API changes, was the need to rewrite significant portions of code.

While it is hoped that each release would bring improvements and bug fixes, this has not always been the case. For instance, the *setPixels* bug described in Section 5.04 was present in Java 1.0x. It was fixed for the Java 1.1x release and has reappeared in the Java 2.x release

There have been several bugs involving image producers and consumers. These bugs have revolved around the failure to observe *offset* values. Work-arounds' to this type of bug typically involve considerable computational overhead. The current code includes a workaround to observe such an offset deficiency. To combat the bug, the system must create a new array, copy the existing data into the array, then set the appropriate pixels in the image and then free the array. The array creation and copying procedures create a significant speed penalty.

As a language develops it is expected that the performance the language will improve, either due to compiler developments, or in the case of Java, interpreter developments. This has not always occurred. The release of *Java2* has created significant performance problems with the production and rendering of images. This is due to the inclusion of an alpha channel to handle transparency. It is expected that the performance of the *Java2* graphics will improve as further development of the *Java2* platform occurs.

Section 8.02 Interaction With Genasys

This project was undertaken with the assistance of Genasys New Zealand. The project has required access to industry sensitive code. Because of the need there have been several delays, while agreements were reached regarding the disclosure of information.

This project would have benefited from being able to access vector data directly from the map files. However, due to the industry sensitive nature of the vector data file format, the author was unable to obtain the file format information. Consequently, it has been necessary to use the *Nova Vista* technology. While this technology has satisfactory performance, the requirement to utilise native code through JNI limits the overall performance of the system.

Further complications arose when the Genasys organisation was placed in receivership during 1998. This has delayed access to some required sections of code towards the end of this project.

Chapter IX. Discussion and Conclusions

Display of spatial data constitutes nearly 50% of all GIS operations undertaken (Section 1.01). Yet, viewing clients for GIS are typically cumbersome and lack adequate performance. Previous Genasys GIS viewing clients have operated using a server based method, where the server draws the requested information and then returns the image as a GIF file. A more efficient method is to download data to the client and render the data directly to its display. This thesis has investigated the merits of a full-download GIS client. The project had the following goals:

- Move non-GIS processing from the server to the client
- Develop a large image handler service for the client
- Develop a vector data rendering service for the client
- Develop a database retrieval client
- Investigate new interface features

This chapter is divided into two sections, the first section details future developments while the final section concludes this thesis.

Section 9.01 Client Developments

The clients' handling of vector data has been limited by the inability of the Genasys data accessors to handle a multi-threaded server. It is viewed that for the vector data client to become of significant use, the accessors would need to become multi-threaded safe. Currently if one user is to download the cadastral database, and a second user was to request the same information, the second user may need to wait before receiving the same data. This wait could be up to 90 seconds long.

The addition of a point-based zoom feature was viewed as a significant improvement. This zooming feature was matched to the dyadic nature of the image pyramid to increase the image performance. Scroll bars where included with the interface for panning support. Users believed these lacked the ability to pan in a diagonal direction, to follow a road for instance. It is thought that the inclusion of a diagonal panning tool would be of use, where a client can effectively drag a point on the current screen to a new location.

The imaging system could be modified to scale in-memory sections of an image, while waiting for the download of the new image. This would be of use over a slow connection. The scaling of in-memory sections was trailed, but a significant bug in the Java *imageProducer* system

ignored requests to render the scaled image. This bug has been fixed for the next release of Java, and could be incorporated into the image handler.

Section 9.02 Java Developments

Java is a rapidly advancing language. The release of *Java2* included some significant new features which could be integrated into the system. The following advancements would enhance the client.

(a) Just In Time Compilers

During the time this project has been under development, significant advances have been made in the performance of Java. The Just In Time compiler has been responsible for the majority of these performance gains. James Gosling, one of the founders of the Java language, believes that further gains will be made, and that the Java language has the potential to improve on the speed of the *C* and *C++* languages for certain functions.(see Appendix I)

(b) Java Plug-In

One of the consequences of the rapidly advancing Java technology is the lack of support on client systems. Previously it was the requirement of each Internet browser manufacturer to include some form of Java support. Because of this, browsers often contained bugs in their Java implementation, and incomplete support for new technology. The development of the Java Plug-In has rectified this problem.

The Java Plug-In requires the modification of the "HTML" page to start the client applet. The Internet browser then searches for the Sun Java Plug-In, if it is unable to find an installation, it fetches a copy from Suns' web site, or from an Intranet site. Once the software is installed, the user has a copy of Suns Java Virtual Machine, which runs the applet. Other applets are still run using the Internet browsers' virtual machine. This technology allows developers to utilise the most recent version of Java without worrying about client support.

(c) Reference Objects API

The recent release of *Java2* has incorporated some new technology for interaction with the garbage collector. Garbage collection has been identified as one of the limiting factors in Java software performance. The Reference Objects API allows the programmer to give the garbage collector some indication as to what should be collected. This then allows the garbage collector to reclaim unused objects, as it requires, typically when memory is running low. This limited

ability to interact with the garbage collector will allow significant gains in controlling how memory is used, improving system performance.[38]

Section 9.03 Conclusion

The main aim of this project was to move non-GIS specific functions from the server to the client. Three components were identified as keys to achieving this aim:

- Raster Data Handler
- Database Client
- Vector Data Handler

The raster data handler uses a dyadic image pyramid to store and transfer images. The client calculates which sections of an image it requires, downloads it from the server, and renders it to the screen. It stores downloaded data in memory to reduce the required number of downloads. This system has proved to perform exceptionally well, outperforming the GenaMap product by over 200%, and the MGIS product by over 400%. This component allows the use of images without incurring significant overheads.

The database client interfaces with the Genasys database server product. The server uses RPC and XDR standards to communicate to the client system. The developed client uses a 100% pure Java implementation to handle the retrieval of data. This system has shown to be 2.5 times quicker than GenaMap and over 25 times quicker than the MGIS product.

The vector data handler downloads information from the *Spatial Nucleus* shared library accessors. The vector data is then rendered on the client machine. Once the data is loaded into the client, the vector data is rendered significantly quicker than in GenaMap. The system has shown to be over 50% quicker than GenaMap.

The system has shown to provide excellent performance and stability. During the user evaluation, users commented that they thought the overall performance was a marked increase over the MGIS system they currently use.

Sections of the client system have been integrated with access to GenaMap scripting to form a replacement for the Genasys Web Broker product. The integrated product is in use at Buller District Council. This product has shown to be an excellent alternative to the Web Broker system, with excellent image performance.

---

[38] For more information on the Reference Objects API see: Reference Objects and Garbage Collection ; Monica Pawlan; 1988 : http://developer.java.sun.com/developer/technicalArticles/monicap/RefObj/refobj.html

The project had an aim of moving non-GIS specific tasks from the server to the client. This has been achieved with the implementation of a vector download system, an image download system and a database client. These three components all offer significant performance advantages over the equivalent systems in GenaMap and the MGIS / WebBroker products.

Introduction to Appendices

The following pages contain information additional to the main body of the thesis. Included in these appendices is a 3.5-inch floppy disk. This disk is formatted for use with IBM compatible computers, and contains various information including some demonstrations of the work completed in this thesis. For further details regarding this disk, please open "*index.html*" from the disk using your web browser.

Appendix I. Java Performance

Section I.I Java performance problems

The following section includes some benchmarking information taken from *http://www.cs.cmu.edu/~jch/java/benchmarks.html : January 1999.* This information measures the times taken for various operations.

Description

The table below shows the time in microseconds to execute various Java operations on a 486 PC (AMD DX4-120 with 24MB RAM and 256kB L2 cache, running Windows 95). The Java implementations tested are:

- Sun JDK 1.0.2.
- Symantec Café 1.51.
- Symantec Visual Café PR 2.
- Microsoft Internet Explorer 3.0 (Visual J++ uses the same VM).
- Netscape Navigator 3.0.
- Asymetrix SuperCede beta.

The Sun JDK uses an interpreter, while all the rest use just-in-time compilers. Times are only quoted to two significant figures because the variance between runs is typically on the order of 5-10%.

Description	JDK	Café	VCafé	IE	NN	SC
Loop overhead: `while (Go) n++`	1.1	0.052	0.050	0.066	0.067	0.065
Local variable assignment: `i = n`	0.48	0.037	0.027	0.009	0.006	0.006
Instance var. assign.: `this.i = n`	1.0	0.043	0.041	0.035	0.034	0.034
Array element assign.: `a[0] = n`	1.2	0.11	0.066	0.043	0.087	0.033
Byte increment: `b++`	1.3	0.068	0.055	0.048	0.053	0.007
Short increment: `s++`	1.3	0.067	0.054	0.048	0.053	0.014
Int increment: `i++`	0.31	0.030	0.022	0.006	0.011	0.006
Long increment: `l++`	1.2	0.071	0.044	0.049	0.038	0.007
Float increment: `f++`	1.3	0.25	0.18	0.17	0.18	0.18
Double increment: `d++`	1.2	0.32	0.20	0.23	0.19	0.18
Object creation: `new Object()`	13.0	9.5	8.2	13.0	26.0	5.9
Array creation: `new int[10]`	13.0	11.0	9.2	13.0	42.0	39.0

Method call: `null_func()`	2.2	0.22	0.12	0.13	0.16	0.13
Synchronous call: `sync_func()`	19.0	13.0	3.6	4.1	16.0	5.1
Math function: `abs()`	4.9	0.68	0.13	0.55	0.59	0.68
Inline code: `(x < 0) ? -x : x`	0.55	0.087	0.09	0.084	0.19	0.61

Analysis

Variable accesses:

The first benchmark times a null loop (this time has been subtracted from all the other results). The next three benchmarks time how long it takes to store an integer in a local variable, in an instance variable, and in an array. As might be expected, local variables are fastest. Instance variables are slower because there's an extra field operation involved (tip from KB Sriram), and array accesses are slower still, due to the bounds checking that Java performs. Note that most JIT compilers store local variables in registers, resulting in an additional speedup.

Data types:

The next six benchmarks time how long it takes to increment the `byte`, `short`, `int`, `long`, `float`, and `double` data types. `int`s are consistently fastest, especially when using JIT compilers. For floating-point code, `double` is typically only slightly slower than `float`.

Objects and methods

Finally we time creating objects and integer arrays, calling normal and synchronized methods, and calling a predefined math function versus inlining a simple version of the same function. Creating an object is slow, and current JIT compilers haven't improved things much. It's typically about the same cost as creating an integer array of length 10. Synchronized method calls are 10-100 times slower than the normal variety. Finally, replacing a system function such as `Math.abs` with simplified inlined code (`(x < 0.0) ? -x : x;`) can significantly improve performance for interpreters and older JIT compilers. However, Visual Café represents the newer wave of JIT compilers with specialized native code for some built-in functions -- you're unlikely to be able to do better than these!

Just-in-time *vs* interpreted:

Comparing the results for Sun's JDK interpreter with a JIT compiler (e.g. VJ++), we can see that just-in-time compilers improve the performance of most operations by 5-30 times. However, the time to create an object hasn't improved at all, and the time to call a synchronized method has only improved by a factor of four, so reducing the number of objects created (see the speed page) and minimizing the amount of synchronization in your code have both become even **more** important.

Section I.II Comments from James Gosling

Java performance is getting pretty interesting. The current round of JIT compilers is pretty good, but they're doing standard static compiler stuff. One of the places where that runs into trouble is with inlining--inlining optimizations are one of the biggest wins. But in OO programming, it's essentially impossible to inline virtual methods, and good OO style drives people to use lots of them. But the new generation of compilers come not from the static compiler school, but from the dynamic school. There's a whole pile of research that has been done over the years on how to make languages like Lisp and SmallTalk fast. If you take a couple of these techniques with strongly typed language semantics, you get something pretty cool.

We're building something called "HotSpot," which is a new generation VM that has a pile of interesting properties: a fancy generational/train garbage collector that's very fast and has short pause times (3-5millisec), new ways to implement synchronized methods that make lock acquisition much faster, and the big deal: a dynamic compiler. As programs run, it collects statistics on what is happening--what the common paths are, the hot spots--that the program is going through. It can then do all sorts of special-purpose optimizations. It can do things like inline virtual methods when it knows what the mix of classes running, and it can do special stuff in the code generator, since it knows precisely which chip the program is executing on. It's pretty far along, but we don't have final benchmarks yet. It's looking pretty cool. I'm really jazzed by it. I think we've got a great chance to actually beat C and C++ performance.

Source:

http://developer.java.sun.com/developer/toolbar/transcripts-9-97/JamesGosling.html

Appendix II. Image Pyramid Setup Instructions

There are three phases to the setup of images for use with the large image handler. The first required the use of a image editor like Paint Shop Pro. Images need to be stored in decreasing resolutions. The following instructions relate to the generation of multiple resolutions using Paint Shop Pro v4.12:

1) Create a directory named "<image name>.pyramid", located in the same directory as the image.

2) Open the image in Paint Shop Pro.

3) Choose Resize from the Image Menu

4) Half the dimensions of the image -- i.e. if the original image was 3998x3000 then resize to 1999x1500,. Use the "maintain aspect ration" option to keep the original aspect ratio.

5) Save the image with a descriptive name that represents the first level of the pyramid. It is recommended that you save as "1.tif" in the "<image name>.pyramid" directory.

6) Continue this process, saving each image as a seperate name, until around 5 levels have been reached.

7) Create a file called <image name>.pyr located in the same directory as the base image, which lists the location and name of each image with the dimensions of that image. i.e. <pyramid image name> = <x dimension>x<y dimension> As shown in the example below.

The following file is called s12d.pyr and is located in the directory \images\s12d.pyr

```
\images\s12d.pyramid\5=125x94
\images\s12d.pyramid\4=250x188
\images\s12d.pyramid\3=500x375
\images\s12d.pyramid\2=999x750
\images\s12d.pyramid\1=1999x1500
\images\s12d=3998x3000
```

Appendix III. Performance Results

The following sections contain the raw performance figures for the three components developed with this thesis. All times are recorded in seconds, some have been timed by computers, and others have been timed using a stopwatch.

Section I.III Database Client

	Direct - Java DBClient	Direct - GenaMap	Intranet Client - Java DBClient	Intranet Client - GenaMap
%JAP% (51 Records)	0.901	1	1.6	4
%AUS% (87 Records)	1.092	2	1.9	5.5
%N% (2208 Records)	3.205	9	4.5	110
%% (2958 Records)	4.266	12	5.5	150

Section I.IV Large Image Handler Client

	MGIS	Large Image Handler	Genamap
Initial Load	13.5	10	18
ZI 1	20	9	9
ZI 2	15.5	2	5
ZI 3	6.5	2	3
ZI 4	5	2	2
ZI 5	4.5	0.5	1
ZI 6	6	0.5	1
PAN0.5	6	0.5	1
ZO 1	6	0.5	1
ZO 2	6	0.5	1
ZO 3	6	1	2
ZO 4	6	1	3
ZO 5	17.5	2	6
ZO 6	18	2	8
ZI 1	17.5	2	7
ZI 2	12.5	2	4
ZI 3	6.5	1	3
ZI 4	5	1	2
ZI 5	4	0.5	2
PAN1	5	1	4
ZO 1	5	1	1
ZO 2	4	1	1
ZO 3	6.5	2	3
ZO 4	10.5	2	5
ZO 5	8	2	6
ZO 6	12.5	3	9

	VectorDataHandler	GenaMap
Initial Load	83	25
ZI 1	5	13
ZI 2	3	8
ZI 3	2	6
ZI 4	1	3
ZI 5	1	3
ZI 6	0.5	2
ZO 1	1	2
ZO 2	1	3
ZO 3	1	5
ZO 4	4	8
ZO 5	7	16
ZO 6	7	16
ZI 1	5	13
ZI 2	2	9
ZI 3	2	6
ZI 4	1	3
ZI 5	0.5	3
ZI 6	0.5	2
ZO 1	0.5	3
ZO 2	0.5	3
ZO 3	2	6
ZO 4	3	9
ZO 5	6	13
ZO 6	7	15

Appendix IV. Genasys API Documentation

Section I.VI Request Broker API

The following information is a detailed description of the Request Broker API. This information is taken from Genasys Request Broker manuals.

Chapter 1 - Introduction

The Request Broker architecture facilitates integration of separate software products into one application. It is incorporated in Genasys workstation products, and provides a simple mechanism for extending and integrating these products. Figure Figure 1.1 shows an example of an application developed using the Request Broker:

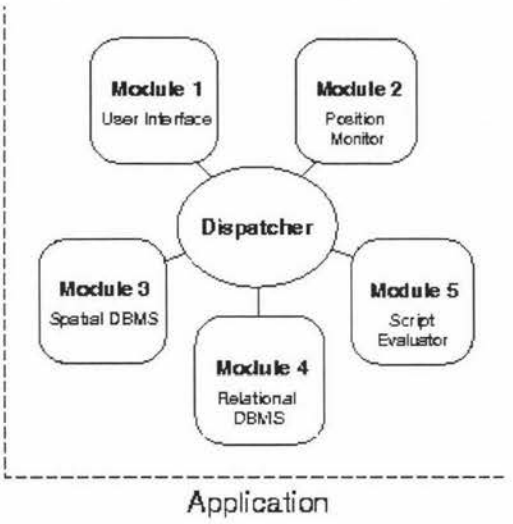


Figure 1.1: Example Application

The various modules may be Genasys products (e.g. Genius) or programs developed by others.

This guide describes how to integrate the Request Broker architecture into your own application.

Chapter 2 - Implementation

2.1 Architecture

The Request Broker architecture is based on a multi-process message passing protocol. Each process is started by the Request Broker. The Request Broker will continually monitor all processes for requests. A request is made up of a service identifier (which is simply an integer) and associated message. When the Request Broker receives a request, it looks up an internal directory of service providers to see which client, if any,

can attend to that request. The Request Broker then forwards the message to the appropriate provider. The Request Broker knows nothing of what the message contains, so it is up to requesters and providers to agree on the format of their messages.

A broad analogy can be drawn between the Request Broker and a telephone exchange. The translation from a telephone number (service request) to a particular line (process) is known by the exchange (Request Broker), which will make the connection: but it is up to the people at either end of the line to agree on their common language, be it English, French or Group 3 (Faxish).

The directory of services is constructed by the Request Broker each time it is run. Each client that provides a service must register that service identifier with the Request Broker. A Request Broker client need not provide any services whatsoever. In this case it is presumably a request initiator, perhaps a user interface. Other clients may only be service providers, and never make requests of their own. Most clients, however, will initiate some requests, and service others. Figure Figure 2.1 shows the sequence of events in such a client:

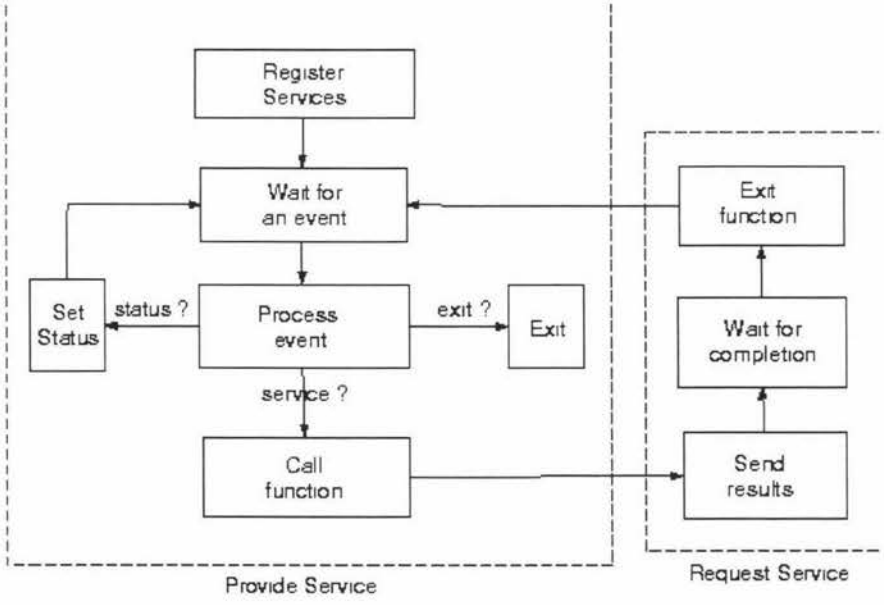


Figure 2.1: Event Sequence

The diagram shows that, in servicing a request (possibly from a user interface) a program may need to request the service of another module (e.g. a user interface, most likely the same user interface from which the original request came).

Section I.VII Symbology Primitives

The following graphics primitives are available for use in symbology procedures. This information is taken from the Genasys Graphics Symbology Manual.

colour (*model* (*component1 component2* ...))
sets the colour for all subsequent rendering. Currently, `RGBi` is the only colour model available. There are 3 colour components that specify the red, green and blue components of the colour. Each of these components are fractional numbers between 0 and 1. If the *model* is set to `clip`, then all subsequent filled areas will not be rendered but become regions that are protected and will not be obliterated by anything else. However, clipping is not useful for line symbolisation.

linewidth (*width*)
sets the width for all subsequent line drawing. The *width* is relative to the scaling that is in effect when the line is drawn.

linestyle (*endstyle joinstyle offset* (*dash pattern lengths* ...))
sets the style for all subsequent line drawing.

edge ( )
draws the outline of the current path. The colour, style and thickness of the line are whatever was previously defined by the colour, linestyle and linewidth primitives. The line is drawn destructively in all cases.

markerline (*procedure* ( ) *spacing offset*)
This primitive is used to place marker symbols at evenly spaced intervals along a line. Each symbol is rotated to follow the angle of the path.

offsetpath (*distance*)
This primitive can be used to create an offset line. It offsets the current path by *distance* units to its right.

fill ( )
flood fills the current path with the current colour. The path must be a closed polygon and defined in a clockwise direction.

hatch (*procedure* ( ) *spacing angle*)
This controls hatching using the line symbology *procedure* to draw parallel lines across the interior of the polygon.

path ((*x0 y0 x1 y1* ... *xn yn* ... [close]))
defines a path. Circular arcs and island polygons may be included in the path. An arc is defined as

path ((... *xstart ystart* arccw *xcentre ycentre xend yend* ...))
where `arccw` denotes a clockwise arc and `arcccw` denotes a counter-clockwise arc. If the start and end points are identical a full circle is defined. Polygons may be closed using a `close` keyword in the path. Polygons must be defined in a clockwise direction, and be closed. Any coordinates after the `close` are interpreted as an island polygon. Island polygons should be defined in a counter-clockwise direction, and be closed.

size (*xsize ysize specifier*)
scales the coordinate system in the x and y axes. If *specifier* is *relative*, then scaling is a multiple of the current scale. If *specifier* is *absolute* then the scale is forced to the sizes given in millimetres. All subsequent paths defined will be scaled by these factors. A negative scale will allow mirroring of symbols.

rotate (*angle specifier*)
rotates coordinate system by *angle* degrees counter-clockwise. If *specifier* is `relative` then the rotation is added to the current rotation. If *specifier* is `absolute`, then the rotation is forced to the angle given (this is useful for forcing marker symbols to remain at a particular rotation). All subsequent paths defined will be rotated by this angle.

Several graphics primitives control how text appears. These primitive operations set the attributes of text and render character strings.

textfont (*font name*)
sets the font that will be used for rendering the string. The *font name* must be an Adobe Type 1 or TrueType font that is available for the device being used.

textsize (*width height*)
sets the size of characters when text is drawn. The dimensions are relative to the scaling in effect when the text string is drawn.

textspacing (*spacing*)
sets the spacing between characters in a string. This is measured as a distance in the same way as `textsize`, not as a fraction of the character size.

textslant (*slant*)
sets the angle to slant characters in a text string. Fonts do not normally require slanting, as many specifically designed italic fonts are available. The *slant* is an angle in degrees, characters are slanted to the left by this angle.

textalign (*alignment*)
sets the alignment of text relative to the current point.

textrotation (*rotation*)
sets a *rotation* angle for each character. Each character is rotated individually by this angle. Rotation is in degrees and is measured in a counter-clockwise direction.

textpath (*angle*)
specifies the *angle* at which a string is drawn. Text is normally drawn horizontally, at an angle of 0.

textdraw (*string*)
renders a *string* at the current point as defined by the path primitive. The characters are drawn along a straight path, given the current drawing state.

Appendix V. TIFF Specification

This appendix contains sections of the TIFF specification relevant to this project. The full specification is included on the disk enclosed with this project as a Adobe PDF file, or can be downloaded from: http://www.adobe.com/Support/TechNotes.html

TIFF Revision 6.0

Final - June 3, 1992

Part 1: Baseline TIFF

The TIFF specification is divided into two parts. Part 1 describes *Baseline TIFF*. Baseline TIFF is the core of TIFF, the essentials that all mainstream TIFF developers should support in their products.

Section 1: Notation

Decimal and Hexadecimal

Unless otherwise noted, all numeric values in this document are expressed in decimal. (".H" is appended to hexidecimal values.)

Compliance

Is and *shall* indicate mandatory requirements. All compliant writers and readers must meet the specification.

Should indicates a recommendation.

May indicates an option.

Features designated TMnot recommended for general data interchange' are considered extensions to Baseline TIFF. Files that use such features shall be designated "Extended TIFF 6.0" files, and the particular extensions used should be documented. A Baseline TIFF 6.0 reader is not required to support any extensions.

Section 2: TIFF Structure

TIFF is an image file format. In this document, a *file* is defined to be a sequence of 8-bit bytes, where the bytes are numbered from 0 to N. The largest possible TIFF file is 2**32 bytes in length.

A TIFF file begins with an 8-byte *image file header* that points to an *image file directory* (IFD). An image file directory contains information about the image, as well as pointers to the actual image data.

The following paragraphs describe the image file header and IFD in more detail. See Figure 1.

Image File Header

A TIFF file begins with an 8-byte image file header, containing the following information:

Bytes 0-1:

The byte order used within the file. Legal values are:

"II" (4949.H)

"MM" (4D4D.H)

In the "II" format, byte order is always from the least significant byte to the most significant byte, for both 16-bit and 32-bit integers This is called *little-endian* byte order. In the "MM" format, byte order is always from most significant to least significant, for both 16-bit and 32-bit integers. This is called *big-endian* byte order.

Bytes 2-3

An arbitrary but carefully chosen number (42) that further identifies the file as a TIFF file.

The byte order depends on the value of Bytes 0-1.

Bytes 4-7

The offset (in bytes) of the first IFD. The directory may be at any location in the file after the header but *must begin on a word boundary*. In particular, an Image File Directory may follow the image data it describes. Readers must follow the pointers wherever they may lead.

The term *byte offset* is always used in this document to refer to a location with respect to the beginning of the TIFF file. The first byte of the file has an offset of 0.

Figure 1

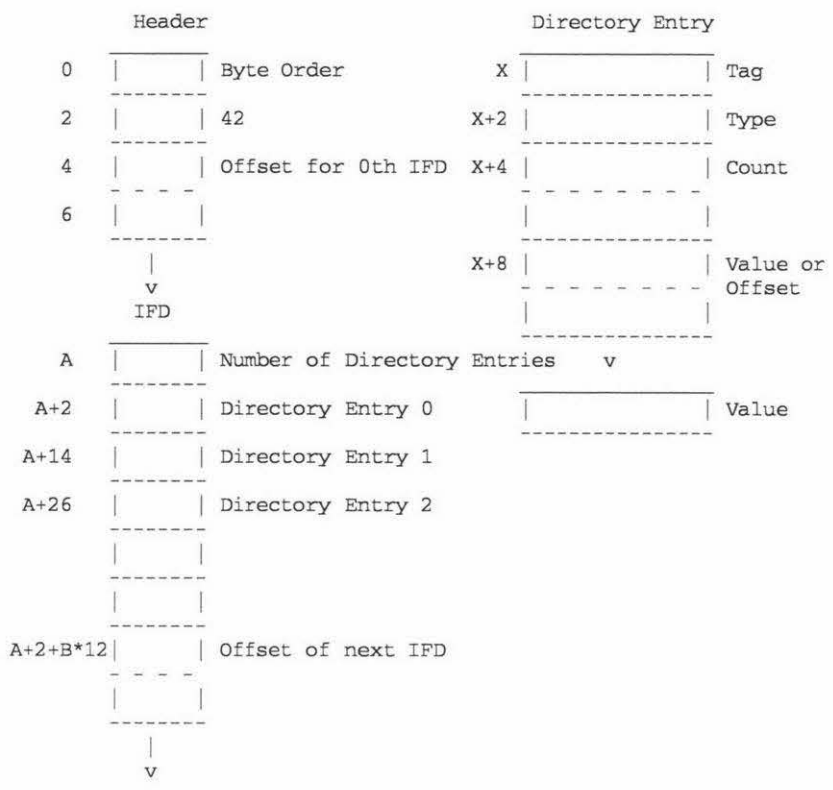


Image File Directory

An *Image File Directory* (IFD) consists of a 2-byte count of the number of directory entries (i.e., the number of fields), followed by a sequence of 12-byte field entries, followed by a 4-byte offset of the next IFD (or 0 if none). (Do not forget to write the 4 bytes of 0 after the last IFD.)

There must be at least 1 IFD in a TIFF file and each IFD must have at least one entry. See Figure 1.

IFD Entry

Each 12-byte IFD entry has the following format:

Bytes 0-1

The Tag that identifies the field.

Bytes 2-3

The field Type.

Bytes 4-7

The number of values, *Count* of the indicated Type.

Bytes 8-11

The Value Offset, the file offset (in bytes) of the Value for the field. The Value is expected to begin on a word boundary; the correspond- ing Value Offset will thus be an even number. This file offset may point anywhere in the file, even after the image data.

IFD Terminology

A *TIFF field* is a logical entity consisting of TIFF tag and its value. This logical concept is implemented as an *IFD Entry*, plus the actual value if it doesn't fit into the value/offset part, the last 4 bytes of the IFD Entry. The terms *TIFF field* and *IFD entry* are interchangeable in most contexts.

Sort Order

The entries in an IFD must be sorted in ascending order by Tag. Note that this is not the order in which the fields are described in this document. The Values to which directory entries point need not be in any particular order in the file.

Value/Offset

To save time and space the Value Offset contains the Value instead of pointing to the Value *if and only if the Value fits into 4 bytes.* If the Value is shorter than 4 bytes, it is left-justified within the 4-byte Value Offset, i.e., stored in the lower-numbered bytes. Whether the Value fits within 4 bytes is determined by the Type and Count of the field.

Count

Count--called *Length* in previous versions of the specification--is the number of values. Note that Count is not the total number of bytes. For example, a single 16-bit word (SHORT) has a Count of 1; not 2.

Types

The field types and their sizes are:

1 = BYTE

8-bit unsigned integer.

2 = ASCII

8-bit byte that contains a 7-bit ASCII code; the last byte must be NUL (binary zero).

3 = SHORT

16-bit (2-byte) unsigned integer.

4 = LONG

32-bit (4-byte) unsigned integer.

5 = RATIONAL

Two LONGs: the first represents the numerator of a fraction; the second, the denominator.

The value of the Count part of an ASCII field entry includes the NUL. If padding is necessary, the Count does not include the pad byte. Note that there is no initial "count byte" as in Pascal-style strings.

Any ASCII field can contain multiple strings, each terminated with a NUL. A single string is preferred whenever possible. The Count for multi-string fields is the number of bytes in all the strings in that field plus their terminating NUL bytes. Only one NUL is allowed between strings, so that the strings following the first string will often begin on an odd byte.

The reader must check the type to verify that it contains an expected value. TIFF currently allows more than 1 valid type for some fields. For example, ImageWidth and ImageLength are usually specified as having type SHORT. But images with more than 64K rows or columns must use the LONG field type.

TIFF readers should accept BYTE, SHORT, or LONG values for any unsigned integer field. This allows a single procedure to retrieve any integer value, makes reading more robust, and saves disk space in some situations.

In TIFF 6.0, some new field types have been defined:

6 = SBYTE

An 8-bit signed (twos-complement) integer.

7 = UNDEFINED

An 8-bit byte that may contain anything, depending on the definition of the field.

8 = SSHORT

A 16-bit (2-byte) signed (twos-complement) integer.

9 = SLONG

A 32-bit (4-byte) signed (twos-complement) integer.

10 = SRATIONAL

Two SLONG's: the first represents the numerator of a fraction, the second the denominator.

11 = FLOAT

Single precision (4-byte) IEEE format.

12 = DOUBLE

Double precision (8-byte) IEEE format.

These new field types are also governed by the byte order (II or MM) in the TIFF header.

Warning: It is possible that other TIFF field types will be added in the future. Readers should skip over fields containing an unexpected field type.

Fields are arrays

Each TIFF field has an associated Count. This means that all fields are actually one-dimensional arrays, even though most fields contain only a single value.
For example, to store a complicated data structure in a single private field, use the UNDEFINED field type and set the Count to the number of bytes required to hold the data structure.

Section 3: Bilevel Images

Now that the overall TIFF structure has been described, we can move on to filling the structure with actual fields (tags and values) that describe raster image data.
To make all of this clearer, the discussion will be organized according to the four Baseline TIFF image types: bilevel, grayscale, palette-color, and full-color images. This section describes bilevel images.
Fields required to describe bilevel images are introduced and described briefly here. Full descriptions of each field can be found in Section 8.

Color

A bilevel image contains two colors--black and white. TIFF allows an application to write out bilevel data in either a white-is-zero or black-is-zero format. The field that records this information is called PhotometricInterpretation.

Compression

Data can be stored either compressed or uncompressed.

Rows and Columns

An image is organized as a rectangular array of pixels. The dimensions of this array are stored in the following fields:

ImageLength

ImageWidth

Physical Dimensions

Applications often want to know the size of the picture represented by an image. This information can be calculated from ImageWidth and ImageLength given the following resolution data:

ResolutionUnit

Location of the Data

Compressed or uncompressed image data can be stored almost anywhere in a TIFF file. TIFF also supports breaking an image into separate strips for increased editing flexibility and efficient I/O buffering. The location and size of each strip is given by the following fields:

RowsPerStrip

The number of rows in each strip (except possibly the last strip.)

For example, if ImageLength is 24, and RowsPerStrip is 10, then there are 3 strips, with 10 rows in the first strip, 10 rows in the second strip, and 4 rows in the third strip. (The data in the last strip is not padded with 6 extra rows of dummy data.)

StripOffsets

For each strip, the byte offset of that strip.

StripByteCounts

For each strip, the number of bytes in that strip *after any compression.*

Putting it all together (along with a couple of less-important fields that are discussed later), a sample bilevel image file might contain the following fields:

```
A Sample Bilevel TIFF FileOffset Description                Value
(hex)                                (numeric values are expressed
                                     in hexadecimal notation)
-------------------------------------------------------------------
Header:
0000   Byte Order                    4D4D
0002   42                            002A
0004   1st IFD offset                00000014IFD:
0014   Number of Directory Entries   000C
0016   NewSubfileType                00FE 0004 00000001 00000000
0022   ImageWidth                    0100 0004 00000001 000007D0
002E   ImageLength                   0101 0004 00000001 00000BB8
003A   Compression                   0103 0003 00000001 8005 0000
0046   PhotometricInterpretation     0106 0003 00000001 0001 0000
0052   StripOffsets                  0111 0004 000000BC 000000B6
005E   RowsPerStrip                  0116 0004 00000001 00000010
006A   StripByteCounts               0117 0003 000000BC 000003A6
0076   XResolution                   011A 0005 00000001 00000696
0082   YResolution                   011B 0005 00000001 0000069E
008E   Software                      0131 0002 0000000E 000006A6
009A   DateTime                      0132 0002 00000014 000006B6
00A6   Next IFD offset               00000000
Values longer than 4 bytes:
00B6   StripOffsets                  Offset0, Offset1, ... Offset187
03A6   StripByteCounts               Count0, Count1, ... Count187
0696   XResolution                   0000012C 00000001
069E   YResolution                   0000012C 00000001
06A6   Software                      "PageMaker 4.0"
06B6   DateTime                      "1988:02:18 13:59:59"Image
Data:
00000700                             Compressed data for strip 10
xxxxxxxx                             Compressed data for strip 179
xxxxxxxx                             Compressed data for strip 53
xxxxxxxx                             Compressed data for strip 160..
End of example
```

Comments on the Bilevel Image Example

- The IFD in this example starts at 14h. It could have started anywhere in the file providing the offset was an even number greater than or equal to 8 (since the TIFF header is always the first 8 bytes of a TIFF file).
- With 16 rows per strip, there are 188 strips in all.
- The example uses a number of optional fields such as DateTime. TIFF readers must safely skip over these fields if they do not understand or do not wish to use the information. Baseline TIFF readers must not require that such fields be present.
- To make a point, this example has highly-fragmented image data. The strips of the image are not in sequential order. The point of this example is to illustrate that strip offsets must not be ignored. Never assume that strip N+1 follows strip N on disk. It is not required that the image data follow the IFD information.

Required Fields for Bilevel Images

Here is a list of required fields for Baseline TIFF bilevel images. The fields are listed in numerical order, as they would appear in the IFD. Note that the previous example omits some of these fields. This is permitted because the fields that were omitted each have a default and the default is appropriate for this file.

TagName	Decimal	Hex	Type	Value
ImageWidth	256	100	SHORT or LONG	
ImageLength	257	101	SHORT or LONG	
Compression	259	103	SHORT	1, 2 or 32773
PhotometricInterpretation	262	106	SHORT	0 or 1
StripOffsets	273	111	SHORT or LONG	
RowsPerStrip	278	116	SHORT or LONG	
StripByteCounts	279	117	LONG or SHORT	
XResolution	282	11A	RATIONAL	
YResolution	283	11B	RATIONAL	
ResolutionUnit	296	128	SHORT	1, 2 or 3

Section 4: Grayscale Images

Grayscale images are a generalization of bilevel images. Bilevel images can store only black and white image data, but grayscale images can also store shades of gray. To describe such images, you must add or change the following fields. The other required fields are the same as those required for bilevel images.

Differences from Bilevel Images

Compression

BitsPerSample

The number of bits per component.

Allowable values for Baseline TIFF grayscale images are **4** and **8**, allowing either 16 or 256 distinct shades of gray.

Required Fields for Grayscale Images

These are the required fields for grayscale images (in numerical order):

TagName	Decimal	Hex	Type	Value
ImageWidth	256	100	SHORT or LONG	
ImageLength	257	101	SHORT or LONG	
BitsPerSample	258	102	SHORT	4 or 8
Compression	259	103	SHORT	1 or 32773
PhotometricInterpretation	262	106	SHORT	0 or 1
StripOffsets	273	111	SHORT or LONG	
RowsPerStrip	278	116	SHORT or LONG	
StripByteCounts	279	117	LONG or SHORT	
XResolution	282	11A	RATIONAL	
YResolution	283	11B	RATIONAL	
ResolutionUnit	296	128	SHORT	1, 2 or 3

Section 5: Palette-color Images

Palette-color images are similar to grayscale images. They still have one compo- nent per pixel, but the component value is used as an index into a full RGB-lookup table. To describe such images, you need to add or change the following fields. The other required fields are the same as those for grayscale images.

Differences from Grayscale Images

PhotometricInterpretation = 3 (Palette Color).

ColorMap

Tag = 320 (140.H)

Type = SHORT

N = 3 * (2**BitsPerSample)

This field defines a Red-Green-Blue color map (often called a lookup table) for palette color images. In a palette-color image, a pixel value is used to index into an RGB-lookup table. For example, a palette-color pixel having a value of 0 would be displayed according to the 0th Red, Green, Blue triplet.

In a TIFF ColorMap, all the Red values come first, followed by the Green values, then the Blue values. In the ColorMap, black is represented by 0,0,0 and white is represented by 65535, 65535, 65535.

Required Fields for Palette Color Images

TagName	Decimal	Hex	Type	Value
ImageWidth	256	100	SHORT or LONG	
ImageLength	257	101	SHORT or LONG	
BitsPerSample	258	102	SHORT	4 or 8
Compression	259	103	SHORT	1 or 32773
PhotometricInterpretation	262	106	SHORT	3
StripOffsets	273	111	SHORT or LONG	
RowsPerStrip	278	116	SHORT or LONG	
StripByteCounts	279	117	LONG or SHORT	
XResolution	282	11A	RATIONAL	
YResolution	283	11B	RATIONAL	
ResolutionUnit	296	128	SHORT	1, 2 or 3
ColorMap	320	140	SHORT	

Section 6: RGB Full Color Images

In an RGB image, each pixel is made up of three components: red, green, and blue.

There is no ColorMap.

To describe an RGB image, you need to add or change the following fields and values.

The other required fields are the same as those required for palette-color images.

Differences from Palette Color Images

BitsPerSample = 8,8,8. Each component is 8 bits deep in a Baseline TIFF RGB image.

PhotometricInterpretation = 2 (RGB).

There is no **ColorMap**.

SamplesPerPixel

The number of components per pixel. This number is 3 for RGB images, unless extra samples are present. See the ExtraSamples field for further information.

Required Fields for RGB Images

TagName	Decimal	Hex	Type	Value
ImageWidth	256	100	SHORT or LONG	
ImageLength	257	101	SHORT or LONG	
BitsPerSample	258	102	SHORT	8,8,8
Compression	259	103	SHORT	1 or 32773
PhotometricInterpretation	262	106	SHORT	2
StripOffsets	273	111	SHORT or LONG	
SamplesPerPixel	277	115	SHORT	3 or more
RowsPerStrip	278	116	SHORT or LONG	
StripByteCounts	279	117	LONG or SHORT	
XResolution	282	11A	RATIONAL	
YResolution	283	11B	RATIONAL	
ResolutionUnit	296	128	SHORT	1, 2 or 3

Section 7: Additional Baseline TIFF Requirements

This section describes characteristics required of all Baseline TIFF files.

General Requirements

- **Options.** Where there are options, TIFF writers can use whichever they want. Baseline TIFF readers must be able to handle all of them.
- **Defaults.** TIFF writers may, but are not required to, write out a field that has a default value, if the default value is the one desired. TIFF readers must be pre- pared to handle either situation.
- **Other fields.** TIFF readers must be prepared to encounter fields other than those required in TIFF files. TIFF writers are allowed to write optional fields such as Make, Model, and DateTime, and TIFF readers may use such fields if they exist. TIFF readers must not, however, refuse to read the file if such optional fields do not exist.

  TIFF readers must also be prepared to encounter and ignore private fields not described in the TIFF specification.

- **'MMM' and 'MII' byte order.** TIFF readers must be able to handle both byte orders. TIFF writers can do whichever is most convenient or efficient.
- **Multiple subfiles.** TIFF readers must be prepared for multiple images (subfiles) per TIFF file, although they are not required to do anything with images after the first one. TIFF writers are required to write a long word of 0 after the last IFD (to signal that this is the last IFD), as described earlier in this specification.

  If multiple subfiles are written, the first one must be the full-resolution image. Subsequent images, such as reduced-resolution images, may be in any order in the TIFF file. If a reader wants to use such images, it must scan the corresponding IFD's before deciding how to proceed.

- **TIFF Editors.** Editors--applications that modify TIFF files--have a few additional requirements:
- TIFF editors must be especially careful about subfiles. If a TIFF editor edits a full-resolution subfile, but does not update an accompanying reduced-resolution subfile, a reader that uses the reduced-resolution subfile for screen display will display the wrong thing. So TIFF editors must either create a new reduced-resolution subfile when they alter a full-resolution subfile or they must delete any subfiles that they aren?t prepared to deal with.
- A similar situation arises with the fields in an IFD. It is unnecessary--and possibly dangerous--for an editor to copy fields it does not understand because the editor might alter the file in a way that is incompatible with the unknown fields.
- **No Duplicate Pointers.** *No data should be referenced from more than one place. TIFF readers and editors are under no obligation to detect this condition and handle it properly. This would not be a problem if TIFF files were read-only entities, but they are not. This warning covers both TIFF field value offsets and fields that are defined as offsets, such as StripOffsets.*
- **Point to real data.** *All strip offsets must reference valid locations. (It is not legal to use an offset of 0 to mean something special.)*

- **Beware of extra components.** *Some TIFF files may have more components per pixel than you think. A Baseline TIFF reader must skip over them gracefully, using the values of the SamplesPerPixel and BitsPerSample fields. For example, it is possible that the data will have a PhotometricInterpretation of RGB but have 4 SamplesPerPixel. See ExtraSamples for further details.*
- **Beware of new field types.** *Be prepared to handle unexpected field types such as floating-point data. A Baseline TIFF reader must skip over such fields gracefully. Do not expect that BYTE, ASCII, SHORT, LONG, and RATIONAL will always be a complete list of field types.*
- **Beware of new pixel types.** *Some TIFF files may have pixel data that consists of something other than unsigned integers. If the SampleFormat field is present and the value is not 1, a Baseline TIFF reader that cannot handle the SampleFormat value must terminate the import process gracefully.*

Notes on Required Fields

- **ImageWidth, ImageLength.** Both "SHORT" and "LONG" TIFF field types are allowed and must be handled properly by readers. TIFF writers can use either type. TIFF readers are not required to read arbitrarily large files however. Some readers will give up if the entire image cannot fit into available memory. (In such cases the reader should inform the user about the problem.) Others will probably not be able to handle ImageWidth greater than 65535.
- **RowsPerStrip.** SHORT or LONG. Readers must be able to handle any value between 1 and 2**32-1. However, some readers may try to read an entire strip into memory at one time. If the entire image is one strip, the application may run out of memory. Recommendation: Set RowsPerStrip such that the size of each strip is about 8K bytes. Do this even for uncompressed data because it is easy for a writer and makes things simpler for readers. Note that extremely wide high-resolution images may have rows larger than 8K bytes; in this case, RowsPerStrip should be 1, and the strip will be larger than 8K.
- **StripOffsets.** SHORT or LONG.
- **StripByteCounts.** SHORT or LONG.
- **XResolution, YResolution.** RATIONAL. Note that the X and Y resolutions may be unequal. A TIFF reader must be able to handle this case. Typically, TIFF pixel- editors do not care about the resolution, but applications (such as page layout programs) do care.
- **ResolutionUnit.** SHORT. TIFF readers must be prepared to handle all three values for ResolutionUnit.

Section 15: Tiled Images

Introduction

Motivation

This section describes how to organize images into tiles instead of strips.

For low-resolution to medium-resolution images, the standard TIFF method of breaking the image into strips is adequate. However high-resolution images can be accessed

more efficiently--and compression tends to work better--if the image is broken into roughly square tiles instead of horizontally-wide but vertically-narrow strips.

Relationship to existing fields

When the tiling fields described below are used, they replace the StripOffsets, StripByteCounts, and RowsPerStrip fields. Use of tiles will therefore cause older TIFF readers to give up because they will have no way of knowing where the image data is or how it is organized. **Do not** use both strip-oriented and tile-oriented fields in the same TIFF file.

Padding

Tile size is defined by TileWidth and TileLength. All tiles in an image are the same size; that is, they have the same pixel dimensions.

Boundary tiles are padded to the tile boundaries. For example, if TileWidth is 64 and ImageWidth is 129, then the image is 3 tiles wide and 63 pixels of padding must be added to fill the rightmost column of tiles. The same holds for TileLength and ImageLength. It doesn't matter what value is used for padding, because good TIFF readers display only the pixels defined by ImageWidth and ImageLength and ignore any padded pixels. Some compression schemes work best if the padding is accomplished by replicating the last column and last row instead of padding with 0's.

The price for padding the image out to tile boundaries is that some space is wasted. But compression usually shrinks the padded areas to almost nothing. Even if data is not compressed, remember that tiling is intended for large images. Large images have lots of comparatively small tiles, so that the percentage of wasted space will be very small, generally on the order of a few percent or less. The advantages of padding an image to the tile boundaries are that implementations can be simpler and faster and that it is more compatible with tile-oriented compression schemes such as JPEG. See Section 22.

Tiles are compressed individually, just as strips are compressed. *That is, each row of data in a tile is treated as a separate "scanline" when compressing.* Compression includes any padded areas of the rightmost and bottom tiles so that all the tiles in an image are the same size when uncompressed.

All of the following fields are required for tiled images:

Fields

TileWidth

The tile width in pixels. This is the number of columns in each tile.

Assuming integer arithmetic, three computed values that are useful in the following field descriptions are:

TilesAcross = (ImageWidth + TileWidth - 1) / TileWidth

TilesDown = (ImageLength + TileLength - 1) / TileLength

TilesPerImage = TilesAcross * TilesDown

These computed values are not TIFF fields; they are simply values determined by the ImageWidth, TileWidth, ImageLength, and TileLength fields.

TileWidth and ImageWidth together determine the number of tiles that span the width of the image (TilesAcross). TileLength and ImageLength together determine the number of tiles that span the length of the image (TilesDown).

We recommend choosing TileWidth and TileLength such that the resulting tiles are about 4K to 32K bytes before compression. This seems to be a reasonable value for most applications and compression schemes.

TileWidth must be a multiple of 16. This restriction improves performance in some graphics environments and enhances compatibility with compression schemes such as JPEG.

Tiles need not be square.

Note that ImageWidth can be less than TileWidth, although this means that the tiles are too large or that you are using tiling on really small images, neither of which is recommended. The same observation holds for ImageLength and TileLength.

No default. See also TileLength, TileOffsets, TileByteCounts.

TileLength

The tile length (height) in pixels. This is the number of rows in each tile.

TileLength must be a multiple of 16 for compatibility with compression schemes such as JPEG.

Replaces RowsPerStrip in tiled TIFF files.

No default. See also TileWidth, TileOffsets, TileByteCounts.

TileOffsets

= SamplesPerPixel * TilesPerImage for PlanarConfiguration = 2

For each tile, the byte offset of that tile, as compressed and stored on disk. The offset is specified with respect to the beginning of the TIFF file. Note that this implies that each tile has a location independent of the locations of other tiles.

Offsets are ordered left-to-right and top-to-bottom. For PlanarConfiguration = 2, the offsets for the first component plane are stored first, followed by all the offsets for the second component plane, and so on.

No default. See also TileWidth, TileLength, TileByteCounts.

TileByteCounts

= SamplesPerPixel * TilesPerImage for PlanarConfiguration = 2

For each tile, the number of (compressed) bytes in that tile.

See TileOffsets for a description of how the byte counts are ordered.

No default. See also TileWidth, TileLength, TileOffsets.