VISTEC

# *LadMan*

# A Large Data Management System

**Walter Schmeing**
VISTEC Software GmbH
Justus-von-Liebig-Str. 7
12489 Berlin, Germany
Fon: +49 30/6392-6060
Fax: +49 30/6392-6062
w.schmeing@vissoft.de
http://www.vissoft.de

**Abstract**

More and more of our customers have to deal with very large datasets like elevation data and digital roadmaps covering Europe or even the entire world, very large images e.g. from satellites or CT-Scans both medical and industrial. In this paper we describe how we tried to solve the problems of storing and especially of accessing those large datasets by developing *LadMan,* a Large Data Management System. We will explain *LadMans's* design and architecture, we will give a short overview of *LadMan's* API and will look at the implementation of *LadMan* as an important part of RUVIS, a Broadcasting Planning and Visualization System developed for Deutsche Telekom by VISTEC Software GmbH.

# 1 Introduction

Today's science, engineering and business is producing increasingly large datasets from different sources which include:

- CFD-, FEA- and simulation packages using large-scale models
- medical scanners using high resolutions (CAT, MRI, ...)
- remote sensing devices (radar, satellites ...)
- GIS-Applications
- Statistics-Applications (political, economic, financial data ...)
- etc.

To save such large datasets, so they become accessible for further processing or visualization from various tools and applications, one needs to invest in ever increasing storage capacity (disk space). Even worse many of the access tools are only capable of reading the complete dataset; very quickly one reaches the limits of physical and virtual memory in atempting to process these large datasets.

Almost all data are compressible thereby reducing their size dramatically. One can use standard compression tools like compress, pack or gzip to save the large data files. This reduces the required disk space but then requires decompression of the complete file to access the data for further processing even if only a small part of it is needed.

We defined the following requirements to be implemented by a Large Data Management System:

- most important is an easy and fast access to the data
- subsampling like cropping, downsizing, zooming, slicing etc. should be done while reading, to make accessing possible independently of the available memory
- the data should be stored in compressed and platform independent form
- there should be no limit in the dimensionality (1D, 2D,3D,4D,..nD, time varying) and any dimension should be dynamically extensible.
- the System should support "all common" data types (char, short, int, long, float, double) (scalar, vector, tensor)

To reach these goals we designed *LadMan,* a Large Data Management System, that divides the original datasets into smaler pieces and stores these regions in compressed format. Smart readers access and decompress only those regions of interest at any time. The access tools have to keep in memory only one region at a time parsing the data region by region to the application.
This approach leads to applications preserving disk and memory resources making them available for examining extremely large datasets.

# 2 *LadMan:* A Large Data Management System

## 2.1 Overview

*LadMan* permits the storage of huge datasets of any dimension, type and size in compressed format and gives you access to those data via smart readers. *LadMan* divides large n-dimensional datasets into smaller pieces and saves the generated regions in compressed and system independent format to the storage media.
The regions have the same number of dimensions as the complete dataset itself. The user can define the size of the regions (region dimensions) separately for each region or more globally for each dimension. While accessing the data *LadMan* only reads or writes one region at a time.
Suppose one has a large 3D-dataset of 1024x1024x1024 elements, one could divide it into regions of 32x32x32 in size to let you easily generate isosurfaces by reading in only a few interesting regions. If examining this data means to you slicing through the 3'rd dimension, you better would divide it into regions of 1024x1024x1 in size for *LadMan* gets one complete z-slice while reading one region.
By default the system uses for each region the compression algorithm with the best compression rate out of the five already integrated algorithms. *LadMan* allows the user to select his favorite one or he can even link in a new algorithm or bind to a hardware compression chip.

*LadMan* can manage the following data types:

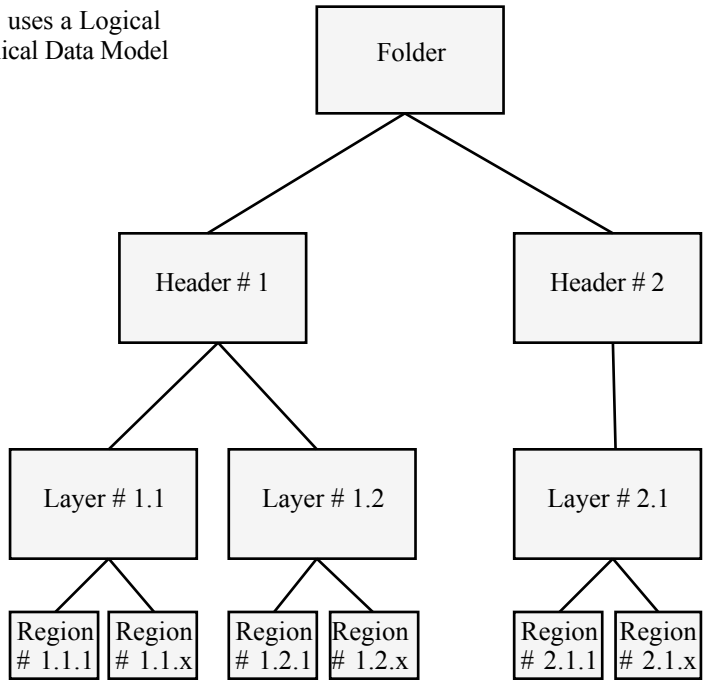
- char, unsigned char
- short, unsigned short
- int, unsigned int
- long, unsigned long
- float
- double

combined as scalar, vector or tensor data.

The simple *LadMan*-API provides access to the compressed data via smart readers capable of doing subsampling like cropping, downsizing, interpolating, mirroring, stretching, slicing, etc. while reading.
Using *LadMan* one can have a quick look at a complete large dataset reading it in at low resolution. Then ome can decide which small part of the data one wanst to see in detail by reading it in at higher resolution.

**Data Model**

*LadMan* uses a Logical Hierarchical Data Model

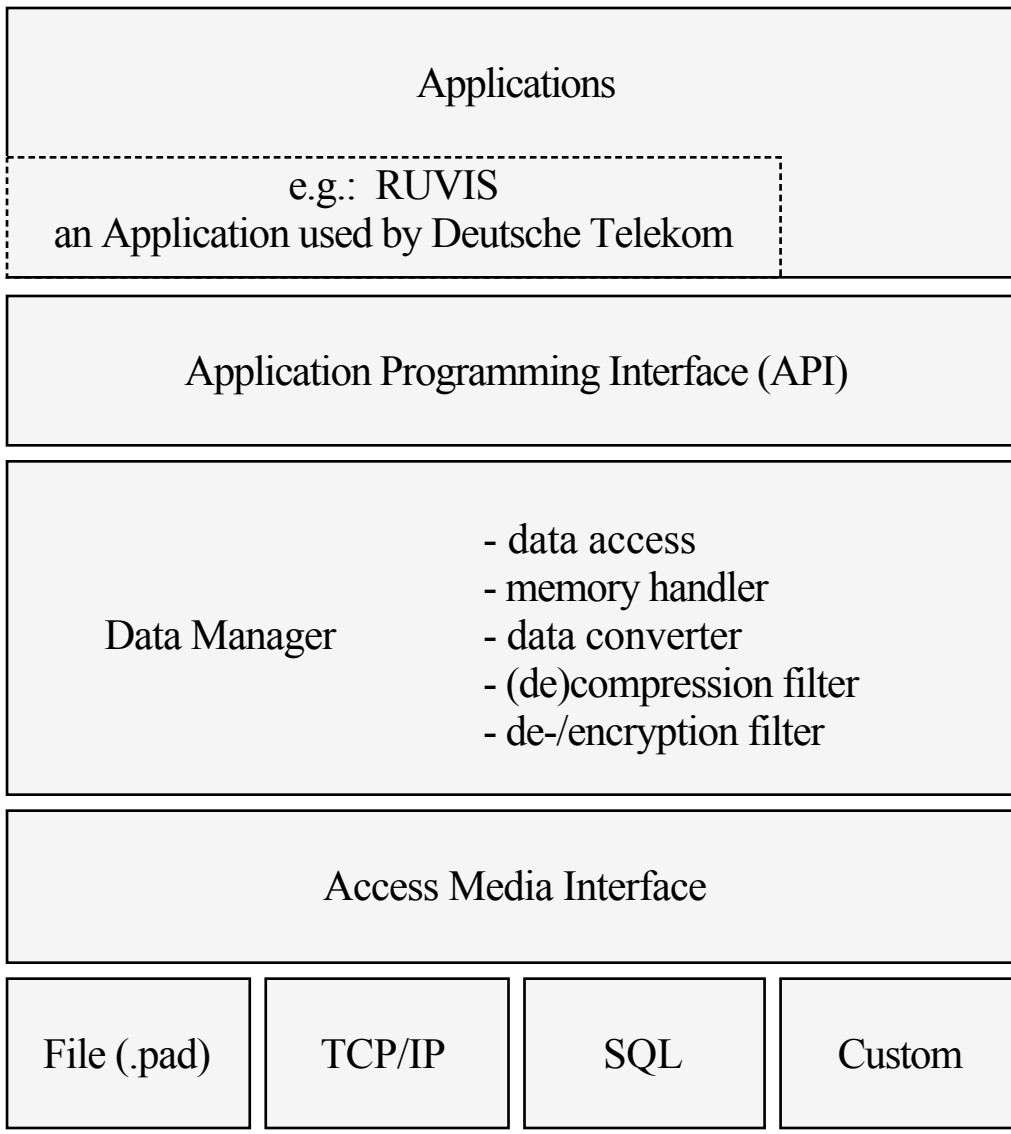



*LadMan* is like a library holding books of different sizes and content. The books have pages of the same paper size with text or images.

| | | |
|---|---|---|
| Folder | The Folder describes the relation of the different data | Library |
| Header | The Header determines the data and coord space (dimensions, resolutions) | Books |
| Layer | The Layer defines the data structure (type, veclen, etc.) | Pages |
| Region | The Region holds parts of the data in compressed format | Text/Images |

**Architecture**

The kernel of *LadMan* is the Data Manager with open interfaces to the storage media and the applications.



## 2.2 The Data Manager

The Data Manager is the kernel of *LadMan* responsible for accessing the requested regions of the complete data set. It accesses only one region at a time sending the region data through the proper de/encryption and (de)compression filter and the data converter to make it available in a system usable format.

In read mode the Data Manager has to perform the following steps for each requested region:

- read in the compressed data of the current region
- decrypt the data using the proper user defined password
- decompress the data using the corresponding algorithm
- convert the data from *LadMan* representation to the native machine one
- copy the data into the user data buffer (doing subsampling if necessary)

The compression filter of *LadMan* uses five built-in run length compression algorithms. One can explicitly define the algorithm *LadMan* should use or let the Data Manager optimize the compression rate by dynamically testing the result of the different algorithms. One can easily integrate another particular algorithm not built-in or use any hardware compression chip.
The Data Manager stores the data in a system independent format. This allows one to access the same *LadMan*-Databases from every supported system.
A memory manager allows the user to define how many megabytes of memory *LadMan* shall use. The Data Manager keeps as many region data in memory as it can hold in the user defined memory limit. If the system reaches this limit it will free the oldest region.

## 2.3 The Access Media Interface

*LadMan* has an open interface to access the storage media. By default it uses the built-in Pad-File. Pad stands for Packed Data and the Pad-File-Media stores *LadMan*-Data to disk files.
You can store one *LadMan*-Database into one or more disk files even on different filesystems.
Due to the simple interface one can define and use a different media (e.g DAT-Tapes). Already integrated is an TCP/IP-Interface, so one can install LadMan-Server and -clients and a SQL-Interface, that lets you store the *LadMan* data into a commercial data base.
One *LadMan* data base can be stored on several media of different types.

The Data Manager uses the following four pseudo functions only:

- OpenMedia ()
- CloseMedia ()
- ReadMediaData ()
- WriteMediaData ()

In the case of the integrated Pad-File-Media the interface sets these functions as follows:

- OpenPadFile ()
- ClosePadFile ()
- ReadPadFile ()
- WritePadFile ()

and internally they use the following System-Calls:

- open ()
- close ()
- lseek ()
- read ()
- write ()

To add a different media one writes new versions of the four interface functions in consideration of the defined parameter list and media-structure.

## 2.4 The Application Programming Interface (API)

*LadMan* has a simple Application Programming Interface (API).
In the following you will find an excerpt of the important calls:

- data_id = DBFLD_**OpenData** ( media_id, header_name, data_name, password)
  opens data and returns a unique data_id used by the API Access Routines

- DBFLD_**ReadData** (data_id, data, minIndices, maxIndices)
  reads data of data_id from min- to maxIndices into data buffer (every value)

- DBFLD_**GetData** (data_id, data, minIndices, maxIndices, dimensions)
  reads data of data_id from min- to maxIndices into data buffer of dimensions

- DBFLD_**ExtractData** (data_id, data, minCoords, maxCoords, dimensions)
  reads data of data_id from min- to maxCoords into data buffer of dimensions,
  where min- and maxCoords reside in the User Coordinate System.

- DBFLD_**WriteData** (data_id, data, minIndices, maxIndices)
  writes data of data_id from min- to maxIndices to a *LadMan*-Database

- DBFLD_**FillData** (data_id, data, minCoords, maxCoords)
  writes data of data_id from min- to maxCoords to a *LadMan*-Database ,
  where min- and maxCoords reside in the User Coordinate System.

- DBFLD_**CloseData** (data_id)
  closes and frees the data and its associated structures

To understand the access mechanism we give a more detailed description of :

DBFLD_**GetData** (data_id, data, minIndices, maxIndices, dimensions)

minIndices, maxIndices and dimensions are integer buffers with entries for each dimension.
By defining minIndices and maxIndices one crops the data.
Setting the size value for each dimension lets one downsize or interpolate the data.
If the size of a dimension equals to one *LadMan* cuts e.g. a time step or a slice or a line.
If maxIndex is less than minIndex *LadMan* mirrors the data in this dimension.

The application programmer can use a single function call to access the data with all possible subsampling features.

## 2.5 Extended features of *LadMan* (in a nutshell)

| | |
|---|---|
| User Coordinates: | The user can link each dimension to his coordinates to realize mapping of indices into user coordinate space (e.g. lattitude and longitude) |
| Data Encryption: | All data can be encrypted by a user defined password preventing unauthorized users to access or ”debug” the data<br>To accelerate the access speed *LadMan* lets one encrypt the higher level hierarchies but leaves the data itself uncrypted. |
| Data Storage: | *LadMan* stores its data in a platform independent way so a data base is identical on all platforms and accessible from each platform. |
| Memory Handler: | The user can define the amount of memory *LadMan* should use causing the system to keep already read data in memory till the defined limit is reached.<br>Then the oldest data are freed. |
| LOD’s:<br>(level of details) | *LadMan* lets you store a dataset in differnt resolutions and automatically reads the data from the best fitting LOD accelerating the access speed dramatically and minimally increasing the storage capacity. |

## 3 Experiences using *LadMan*

Using the simple Application Programming Interface (API) it has been simple to write interfaces or converter to create *LadMan*-Databases from different input formats. (e.g.: TIFF, JPEG, DICOM, ...)
Developing such a converter one should take in consideration that *LadMan* always stores a complete region to the data base, even if the data to be store covers only a part of that region. (*LadMan* first reads the old region data, if already present)
It is recommend to develop a converter so that it writes a region only once to the newly created data base avoiding updating of this region.
If the new size of a compressed region exceeds the old one, *LadMan* has to append the region data to the end of the data base increasing the data base size and leaving gaps in the data base.
Suppose you want to store an image and have defined the region size to 50x50 pixel. If your converter would store the image pixel by pixel to a data base, *LadMan* has to write this region 2500 times and probably has to expand the data base each time.
So one better fills the region data buffer first pixel by pixel and then writes the completely filled region data buffer to the data base.
If it cannot be avoided to produce gaps one can of course use the REORG-utility to pack a data base filling the gaps.

If a user creates a *LadMan*-Database he should have in mind, how he wants to access the data, in order to define the optimal region sizes.
e.g.: to store a 3D-dataset for slicing through the Z-dimension one should set the region size of that dimension to 1 for *LadMan* to read a complete Z-slice with one database access.
But if instead one wants to read a complete X-slice from this data base, *LadMan* has to access every region in the data base.
So the user should carefully plan the definable region sizes.
One can also store the same data set with different region sizes to one data base.
We applied this for example to the Data of the National Library of Medicine's Visible Human Project, which we stored in four different data layers each one optimized for different access (X-slice, Y-slice, Z-slice, 3D-access for e.g. isosurface).
Although we stored the same data four times, the size of the data base is still only less than 70% of the original data due to the compression.

In almost all cases *LadMan*-Access is faster than "traditional" readers (like file access) probably because *LadMan* reads compressed data and therefore far less data from disk.

Using *Ladman* in different projects we have developed a higher level API, that automatically reads the best fitting level of detail or the right "slice layer" or reads in advance the regions the application probably will demand next while panning through a dataset.

# 4 *LadMan* as part of the RUVIS-Application

RUVIS is a VHF/UHF Broadcasting Planning and Visualization System developed by VISTEC for Deutsche Telekom AG (German Telecom) to simplify and accelerate the planning of broadcasting transmitters.
The system helps to optimize technical parameters of the transmitter station like power and frequency and to find its best location.

RUVIS supports the planning engineer by providing access to several different data:

- elevation data
- land usage data
- digital road maps in various resolutions
- vector data like state and district borders
- political data like inhabitants or households etc.
- economic data
- coverage data generated by a wave propagation model

The RUVIS-Application makes these data visible to the engineer.
RUVIS can blend different data like transmitter coverage with district borders and road maps generating images which will be visualized in a 2D-display or as textures mapped over the terrain.

The data are available as 2D-*LadMan*-Fields using geographical latitude and longitude as user coordinates.
Most of the data cover a large part of Europe (at least the complete area of Germany).

The digital road maps alone would consume more than 6 GB's of disk space if stored in uncompressed format. Using *LadMan* the same data require less then 800 MB's saving more than 5.5 GB's of storage capacity.

Furthermore the smart readers of *LadMan* in conjunction with RUVIS provide the planning engineer easy access to the available data. It allows him data to be read at different resolutions into buffers of the same size used for blending. Zooming in and out for getting new views or switching between road maps of different scale is a very easy task now.

# 5 Conclusions

Scientists and engineers are increasingly concerned with large datasets.
They want to be able to examine these datasets.
Tools are necessary for storing and accessing their large data.
Such tools must be easily integrated into existing applications.

*LadMan* implements these demands by its design:

- large n-dimensional datasets become divided into smaller regions
- these regions are being stored in compressed format saving much disk space
- smart readers give easy access to the data (regions)
- a simple API allows a fast integration into applications
- higher level interfaces (e.g. to AVS or DX) make it immediately available to the user
- new storage media can be added very easily

This open design lets *LadMan* manage very different data supporting scientists and engineers working in various fields.

The integration of *LadMan* into an existing Visualization-Application (RUVIS of Deutsche Telekom) has proved useful to the planning engineers and the application programmers.

# 6 References

[1] Lloyd A. Treinish
Unifying principles of data management for scientific visualization.
In *Animation and Scientific Visualization Tools & Applications*,
pp. 141-170, Academic Press, 1993

[2] Nielsen, G., Brunet, P., Gross, M., Hagen, H. et al.
Chapter III: Visualizing Large Data Sets
In *Scientific Visualization Advances and Challenges*, L. Rosenblum et al. (Eds)
pp. 143-198, Academic Press, 1994

[3] ACM Siggraph94 Course #27:
Visualizing and Examining Large Scientific Data Sets.
Chair: Theresa Marie Rhyne
Instructors: Bill Hibbard, Lloyd Treinish, Kevin Hussey, 1994

[4] Samet, Hanan
Applications of Spatial Data Structures.
Computer Graphics, Image Processing, and GIS
Addison-Wesley Publishing, 1990

[5] Frank, Andrew U., Kuhn, Werner (Eds.)
Spatial Information Theory: A theoretical Basis for GIS.
International Conference COSIT'95, Semmering, Austria, September 1995
Proceedings, Springer-Verlag, 1995