DataAnalyzer V1.0

By Jean-Pierre Ribreau (JPR) – 4D Trainer and Consultant

Technical Note 15-05

Table of Contents

Abstract

This Technical Note presents DataAnalyzer, a 4D Tool targeted to 4D Files detailed Analysis. DataAnalyzer opens files created or used by 4D & Wakanda in brute force, and analyses their contents to find possible inconsistencies, and to help to understand the structure of these files. It draws a color Map showing the data from global view to detailed content.

Introduction

The first version of DataAnalyzer has been made in 1996, for 4D versions 5.0 to 6.5. This version is a totally new version, redone from scratch, and able to analyze every files used by 4D and Wakanda. This version 1.0 handles the 4D Data File (.4DD) and Wakanda Data File (.waData). It is a compiled pure 4D application, which opens directly the file, and analyses it block after block. It creates different color Maps, and displays the Data by Objects of every type, from the Allocation Tables to the content of each Record. The next versions will perform analysis on other files (Index files, Structure files), as well as provide more statistics on data itself.

Simplified view of Files Structure

4th Dimension data files, index files, structure files, Wakanda data files and index files share the same internal structure. DataAnalyzer is a Tool that has been made to process autopsies of DB4D Files.

DB4D is the open source data engine used by 4D & Wakanda. In this Technical Note, Data File will refer to any file created and handled by DB4D.

In the rest of this document, we will discuss of 4D data files (.4DD) and Wakanda data files (.waData), for it is the ones presently supported by DataAnalyzer V1.0. Because the internal structure has changed since 4D V11, all these explanations will apply only on data files since 4D V11, and not on previous versions (2004 and previous)

Each data file is composed of a whole number of 128-byte ($80) data blocks.

Note: *Depending on the physical allocation block size of your hard drive, the physical size of a data segment might not be a multiple of 128. Whatever the case, 4D only works with data blocks of 128 bytes (the trailer bytes at the end of the file, if any, are and have to be ignored.*

The maximum size for a Data file (or Index file, or Structure file) is equal to 65 536 * 8 192 * 16 384 * 128 bytes, equivalent to 1 048 576 Gb (1024 Tb or 1 PetaByte or 2^50 bytes)

A data file is composed of the following parts:

- Header(s)
- Block Allocation tables
- Record Address Table tree
- BLOB Address Table tree
- The actual records

Note: *These different parts are allocated dynamically in the data file, depending on the operations performed on the database.*

- The Page Header contains information like addresses of Primary Address Tables, UUIDs, Consistency information, and so on.

- The Block Allocation tables is telling 4D which blocks are actually in use or available. It uses 1 bit per 128-byte block.

- The Record Address Tables exist for each 4D Table actually created. A 4D database can contain up to 32 767 tables (referred to as 4D tables in this section). For each 4D table, there is a tree with primary, secondary (and possibly more) Address Tables stored dynamically in the data file. Each one has the same structure of a tree. It starts with a page with 1024 entries. Up to 1024 records, one page is enough. Then secondary pages are created up to 1024 * 1024 records. A third level is then added, which pushes the number of records per table to 1024 * 1024 * 1024 (around a billion records).

A Record Address Table has a header of 128 bytes, followed by 1024 addresses (Record or Secondary Table) with an actual address (8 bytes = 64bits) + a Record length (4 bytes). So it takes 12,288 bytes + header = 97 blocks.

The Record Address Tables of a 4D table describes the exact location of each record within the file. The term Address refers to the physical address (offset from the beginning of the file) of the first byte of a 128-byte block. 8 bytes = 64 bits address = addressing 2^64 bytes, which is 16 Exabytes (exa = 2^60 = 10^17 = mega-terabytes).

When you create a new 4D table in the Structure window, 4D automatically creates one Address Table for the first 1024 records, and it takes less than 16K. When the first Address Tag is full, 4D creates a Primary Address Table, and the previously created Address Table becomes the first Secondary Address Table, which contains the addresses of 1024 records.

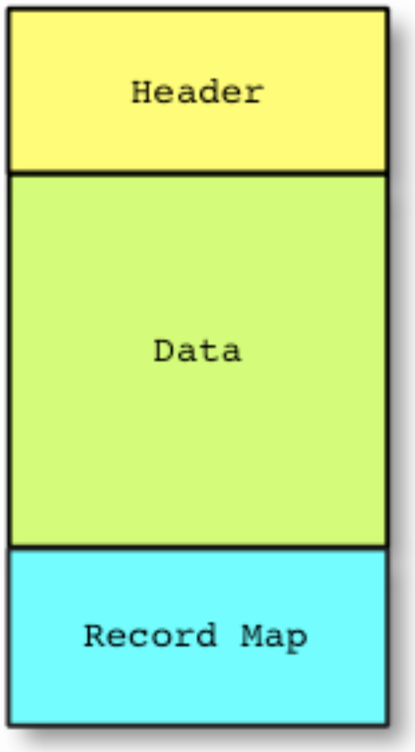
The position of a record in the Address Table is known as the "record number." This number can range from zero up to position 2^30 (1 Giga Rec) depending on how many entries are in the Address Table.

Because records can be added, modified and deleted (as well as indexes and all the necessary internal Address Tables), the 4D database engine keep tracks of the data block use by the means of a **Datablock Allocation Bitmap**. Within the bitmap, a bit is set to 1 (one) when the block is in use; it is set to 0 (zero) when the block is free.

In order to retrieve each allocation bitmap, the 4D database engine maintains an **Allocation Bitmap Address Table** composed of 4,096 entries. Each entry is the 32-bit address to the corresponding allocation bitmap within the data segment. The first entry refers to the first allocated bitmap, the second entry the second allocated bitmap and so on. An entry is equal to $00000000 (zero) when the allocation bitmap has not been created yet.

A Record on disk consists of 3 parts:

- A block 32-byte Header
- A block of the values (i.e. the contents of each field)
- A block of the Record Map, which describes the structure of this particular record: the type of contents and the offset from the beginning of the Data part.



Record Structure

The Record Header contains:

(uLONG) Length:	Size of raw data
(uLONG8) timestamp:	Updated every times the record is saved
(uLONG) checksum:	To check the consistency of the whole record
(sLONG) pos:	Position in the address table
(sLONG) parent:	Table number
(sLONG) nbfields:	Number of fields of this record

The data is always fixed size and takes, for each field:

1 byte for a Boolean

2 bytes for an integer

4 bytes for a Longint

8 bytes for a 64-bit Integer

8 bytes for a Real

8 bytes for a Float

8 bytes for a Date

8 bytes for a Time

4 + (2 * String Length) bytes for a String

4 + (2 * Text Length) bytes for a Text saved inside the record

Or 4 bytes for a Text saved in the Data file or outside the Data file

4 + (Picture size) bytes for a Picture saved inside the record

Or 4 bytes for a Picture saved in the Data file or outside the Data file

4 + (Blob size) bytes for a Blob saved inside the record

Or 4 bytes for a Blob saved in the Data file or outside the Data file

The Record Map is saved at the end of the record itself. In this map, each field uses 8 bytes:

- An offset from the beginning of the data block
- An integer to specify the type of the field.

File Header

At least a minimum understanding of the content of the File Header is recommended to understand the way DalaAnalyzer works.

Each data file starts with a 256-byte **header block** (block 0 + 1). The header block is followed by the First Address Table (#0) of **Datablock Allocation Bitmap** (Start Address $0100)

Content: 8192 * 8 (addresses) ($10000 = 65536) followed by 8192 * 2 (lengths) ($4000 = 16384)

This is an example of Data File Header with its interpretation:

```
0244 0144 uLONG tag;    //4401 4402 always
0000 0000 sLONG lastoper;     //0 or -1  cache status
FFFF FFFF sLONG lastparm;     //always -1
0100 0000 sLONG nbsegdata;    //(4D or Wakanda file))
0100 0000 0300 0000    uLONG8 VersionNumber for Data;
85C4 91CC
8FE8 4FA7
B8C8 93B6
C077 F8D5 VUUIDBuffer ID;
   //16 bytes for synchro data <-> struct
809D 4800 0000 0000    DataAddr4D finfic;  //Logical EOF(sLONG8)
0000 0000 0000 0000    DataAddr4D limitseg;       //Maximum Size for a segment
0000 0000 0000 0000    DataAddr4D AddPagePrim;
   //Primary Block Allocation Address Table
   //Only above 17 Gb)(BitTables)
8000 0000 sLONG sizeblockseg; //Size of Minimum blocks
0700 0000 sLONG ratio;        //Shift ratio bits
8069 0200 0000 0000    DataAddr4D DataTables_addrtabaddr;
   //Address of Address Tables of 4D Tables
FFFF FFFF FFFF FFFF    DataAddr4D addrmultisegheader;   //Always -1
0000 0000 sLONG lenmultisegheader;   //Length of Segment Headers
9CDA E102 sLONG filfil;
   //Random value to link with indexes
5302 0000 sLONG fStamp;//Count header Modifications
FFFF FFFF FFFF FFFF    sLONG8 lastaction;  //Last Action in Log
0000 0000 sLONG countlog;     //Nb of Log Files
F1C5 536B sLONG LastFlushRandomStamp;       //Duplicate Indexes detection
0400 0000 sLONG nbDataTable;
   //Nb of Address Tables of 4D Tables
00  uBOOL doischangerfilfil;          //1 in case repair is needed
01  uBOOL backupmatchlog;             //not used
00  uBOOL IsAStruct;          //0=Data, 1=Structure
01  uBOOL WithSeparateIndexSegment; //Always 1
```

```
006C CA88 FFFF FFFF     DataAddr4D TableDef_debuttrou;
   //position of 1st hole; or -2 billions if no hole
0087 0100 0000 0000     DataAddr4D TableDef_addrtabaddr;
   //address of table primary table (TDEF)
1B00 0000 sLONG nbTableDef;
0000 0000 sLONG nbRelations;         //if structure file
006C CA88 FFFF FFFF     DataAddr4D Relations_debuttrou;
0000 0000 0000 0000     DataAddr4D Relations_addrtabaddr;
0300 0000 sLONG nbSeqNum;            //if data, seq nb
006C CA88 FFFF FFFF     DataAddr4D SeqNum_debuttrou;
004A 0100 0000 0000     DataAddr4D SeqNum_addrtabaddr;
1400 0000 sLONG nbIndexDef;          //index definition in data
          //tag iDEF header d'index
006C CA88 FFFF FFFF     DataAddr4D IndexDef_debuttrou;
0060 3A00 0000 0000     DataAddr4D IndexDef_addrtabaddr;
0000 0000 sLONG nbIndexDefInStruct;  //not used in data
006C CA88 FFFF FFFF     DataAddr4D IndexDefInStruct_debuttrou;
0000 0000 0000 0000     DataAddr4D IndexDefInStruct_addrtabaddr;
0039 3900 0000 0000     DataAddr4D ExtraAddr;
   //extra properties per table & field
FC00 0000 sLONG ExtraLen;
006C CA88 FFFF FFFF     DataAddr4D DataTables_debuttrou;
D700 0000 sLONG countFlush;
0C04 0000 uLONG dialect;// 0 or -4444 (system)
01  uBYTE CollatorOptions collatorOptions;
   //bit 1: withICU,
   //bit 2: ignoreWildCharInMiddle
```

Before attempting to read and load records from a data file, you must have successfully read and validated the contents of the following objects:

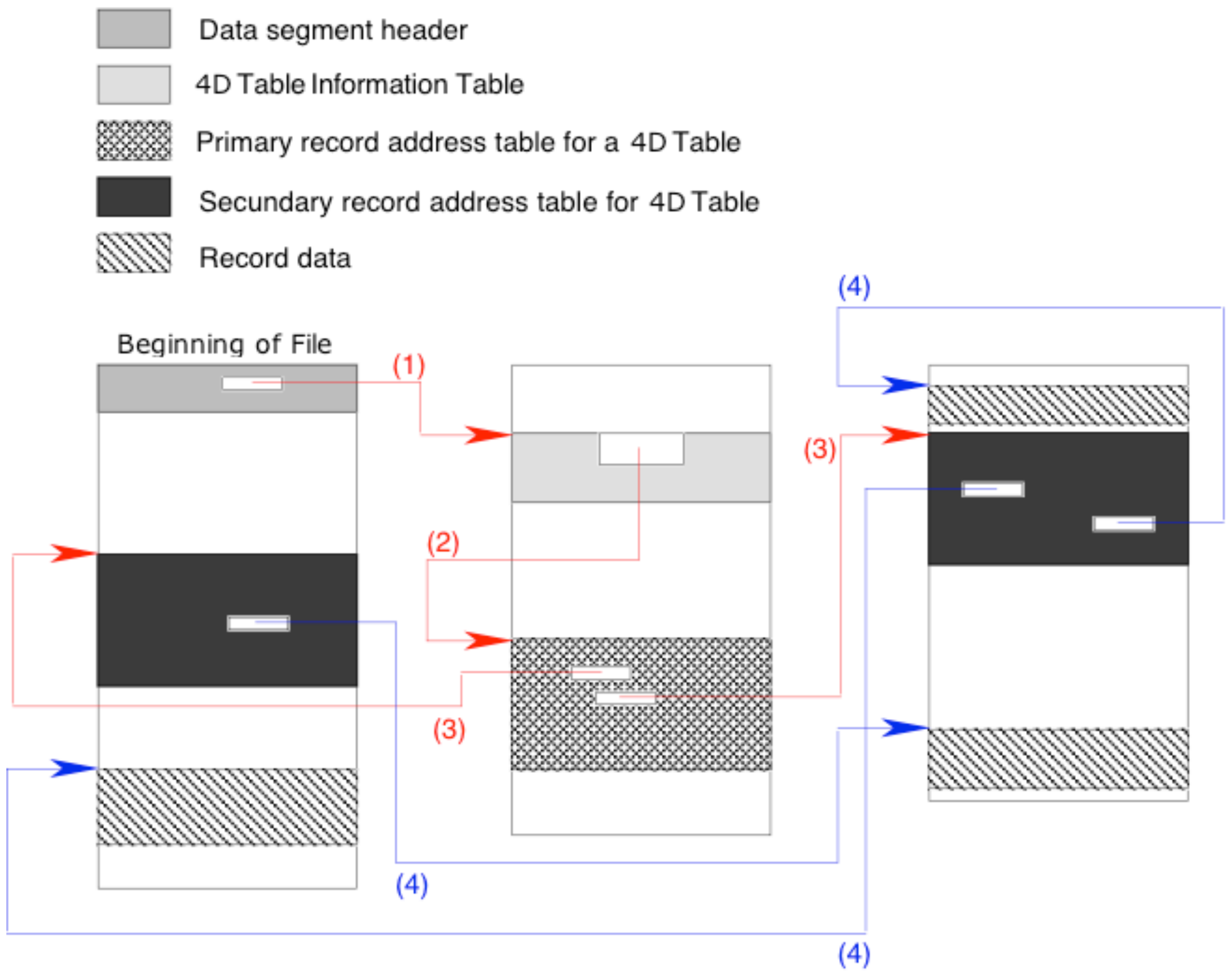
- Data File Header
- Block Allocation Address Tables
- Address Tables of 4D Tables

At this point, you have all the necessary information for accessing the records of a particular file:

- The number of records addresses for the file
- The address (file position) of the Primary Record Address (PRA) table
- The number of Secondary Record Address (SRA) tables listed in the PRA
- The record number of the last deleted record.

To access to an object, it is necessary to understand that 4D keeps the physical address of every objects inside Address Tables, which hold a pack of addresses (from 1024 to 8096) in one object, for instance a TableAddress (TabA). When the number of final Objects (Records rec1, Blobs blob, etc.) exceed the size of the AddressTable, a Secondary Address Table becomes necessary, and you get 2 levels: Primary Address Table -> Secondary Address Table -> Object. If needed, more levels can be added up to 4 levels.

This can be represented with the picture shown below:


Data segment header
4D Table Information Table
Primary record address table for a 4D Table
Secundary record address table for 4D Table
Record data
Beginning of File
(1)
(2)
(3)
(3)
(4)
(4)
(4)

Notes about the figure above

1. The File Header gives the address (file position) of the 4D Table Information Table.
2. The entry for a 4D Table gives the address (file position) of the Primary Record Address (PRA) table.
3. The PRA Table gives the addresses (file positions) of the Secondary Record Address (SRA) tables.
4. The SRA tables give the addresses (file positions) of the records.

DataAnalyzer: The way it works

DataAnalyzer opens any file and reads directly the data on disk. Then, it tries to interpret the data by the book, i.e. the way it is supposed to be. It will read each block, and check if there are inconsistencies between the data read and the data supposed to be. It will also rebuild all the trees representing the cascading addresses, and compare everything as much as possible, in order to get information and statistics that 4D doesn't return up to now.

From all this data, DataAnalyzer can create Maps, compare Maps, find possible errors in checksums, compare timestamps, etc.

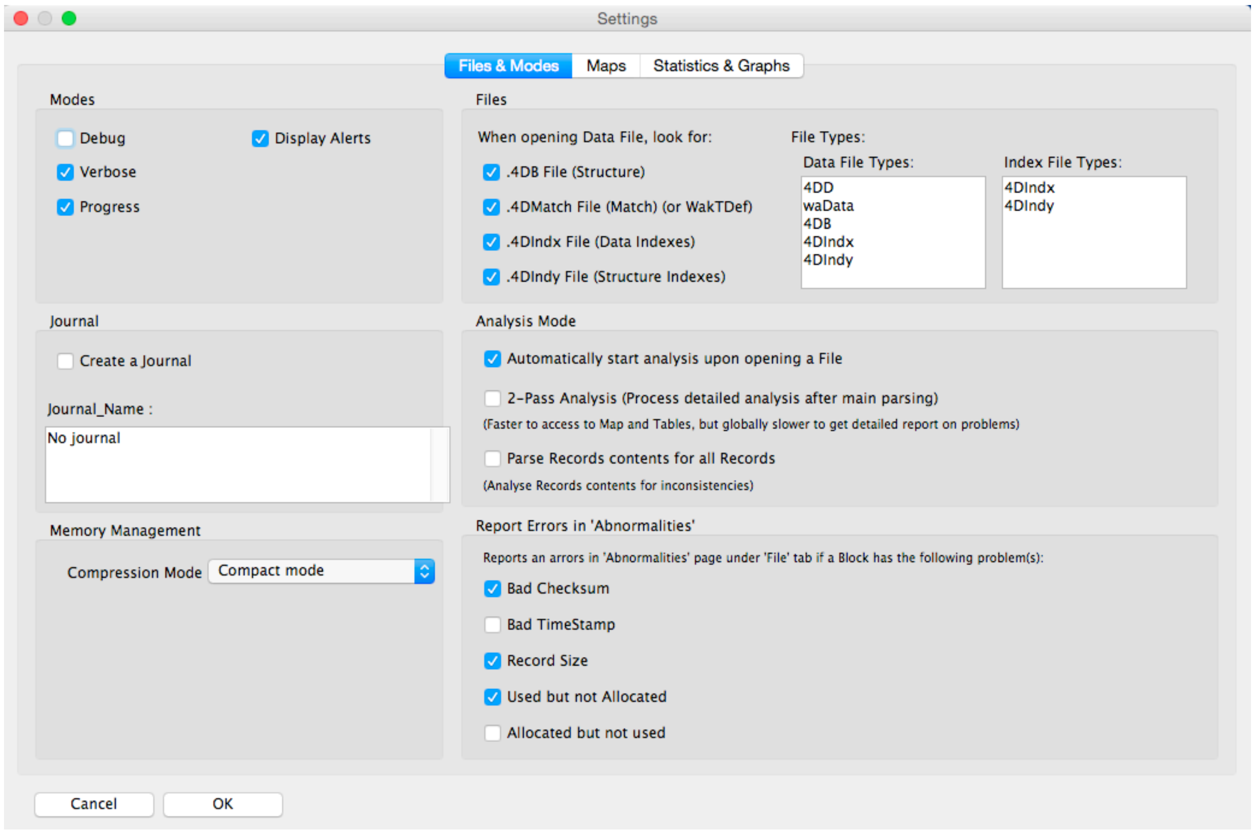
DataAnalyzer has only one menu. Let's see how to use it.

DataAnalyzer: Settings...

This page allows you to customize the way DataAnalyzer will proceed. In V1.0, Settings has 2 Pages.

Note: *On this first version, some of these checkboxes are not used yet.*

- **Page 1** (Files & Modes) handles the behavior:



Modes

- [ ] Debug
- [✓] Display Alerts
- [✓] Verbose
- [✓] Progress

Files

When opening Data File, look for:

- [✓] .4DB File (Structure)
- [✓] .4DMatch File (Match) (or WakTDef)
- [✓] .4DIndx File (Data Indexes)
- [✓] .4DIndy File (Structure Indexes)

File Types:

Data File Types: 4DD, waData, 4DB, 4DIndx, 4DIndy

Index File Types: 4DIndx, 4DIndy

Journal

- [ ] Create a Journal

Journal_Name : No journal

Analysis Mode

- [✓] Automatically start analysis upon opening a File
- [ ] 2-Pass Analysis (Process detailed analysis after main parsing)
  (Faster to access to Map and Tables, but globally slower to get detailed report on problems)
- [ ] Parse Records contents for all Records
  (Analyse Records contents for inconsistencies)

Memory Management

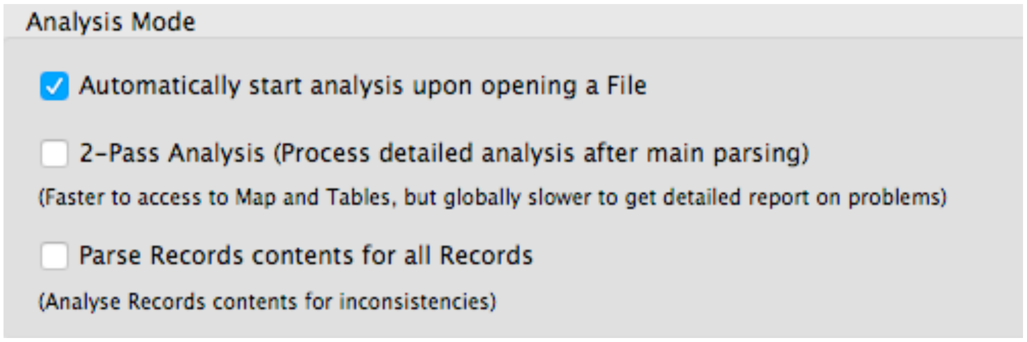
Compression Mode: Compact mode

Report Errors in 'Abnormalities'

Reports an arrors in 'Abnormalities' page under 'File' tab if a Block has the following problem(s):

- [✓] Bad Checksum
- [ ] Bad TimeStamp
- [✓] Record Size
- [✓] Used but not Allocated
- [ ] Allocated but not used

- Modes: Modifies the way the analysis is done. This part is mainly used for debugging.
- Journal: DataAnalyzer can create a detailed Journal of everything happening during the File Analysis.
- Memory Management: DataAnalyzer may have to handle big amounts of data in memory. This data will be compressed. This choice defines the compression mode, according to the different choices of the COMPRESS BLOB 4D Command.
- Files: Select which kind of files you want DataAnalyzer to attempt to open automatically when opening a Data File.
- Analysis Mode: Depending on which information you want to get, you can ask DataAnalyzer to perform a faster or a deeper analysis. The second checkbox allows automatic launch of a second pass of analyze. The first pass is faster, but the total time spent is longer. The third checkbox starts a detailed internal analysis of each record of each table, in order to detect possible internal record inconsistencies.


Analysis Mode

✓ Automatically start analysis upon opening a File

☐ 2-Pass Analysis (Process detailed analysis after main parsing)

(Faster to access to Map and Tables, but globally slower to get detailed report on problems)

☐ Parse Records contents for all Records

(Analyse Records contents for inconsistencies)

- Report Errors in 'Abnormalities': Select what kind of inconsistencies will be reported in the Abnormalities report. For instance, to data files created before V13, it is possible to have a meaningless Time Stamp for each record. In this case, it is useless to report it.

- **Page 2** (Maps) allows you to customize the way the Map is drawn:

On the left part ('Map') you will see a list of all the possible Objects that can be found inside a 4D File. Each of these objects can use from 1 to N 128-byte blocks. The first 4 bytes of any Object is the Tag (or Identifier, or Signature) of the objet.

The first 8 lines do not correspond to an object, but to a specific property of a Block.

- Line 1: Empty Block. This block is nor presently used by 4D.
- Line 2: Claimed Block. This block is used by 4D. Or, at least, 4D has marked it as used in the Datablock Allocation Bitmap.
- Line 3: Recoverable Block. This block has a structure of a block header, but is not marked as used in the Datablock Allocation Bitmap. It might be recoverable in case of a Repair by Tag operation.

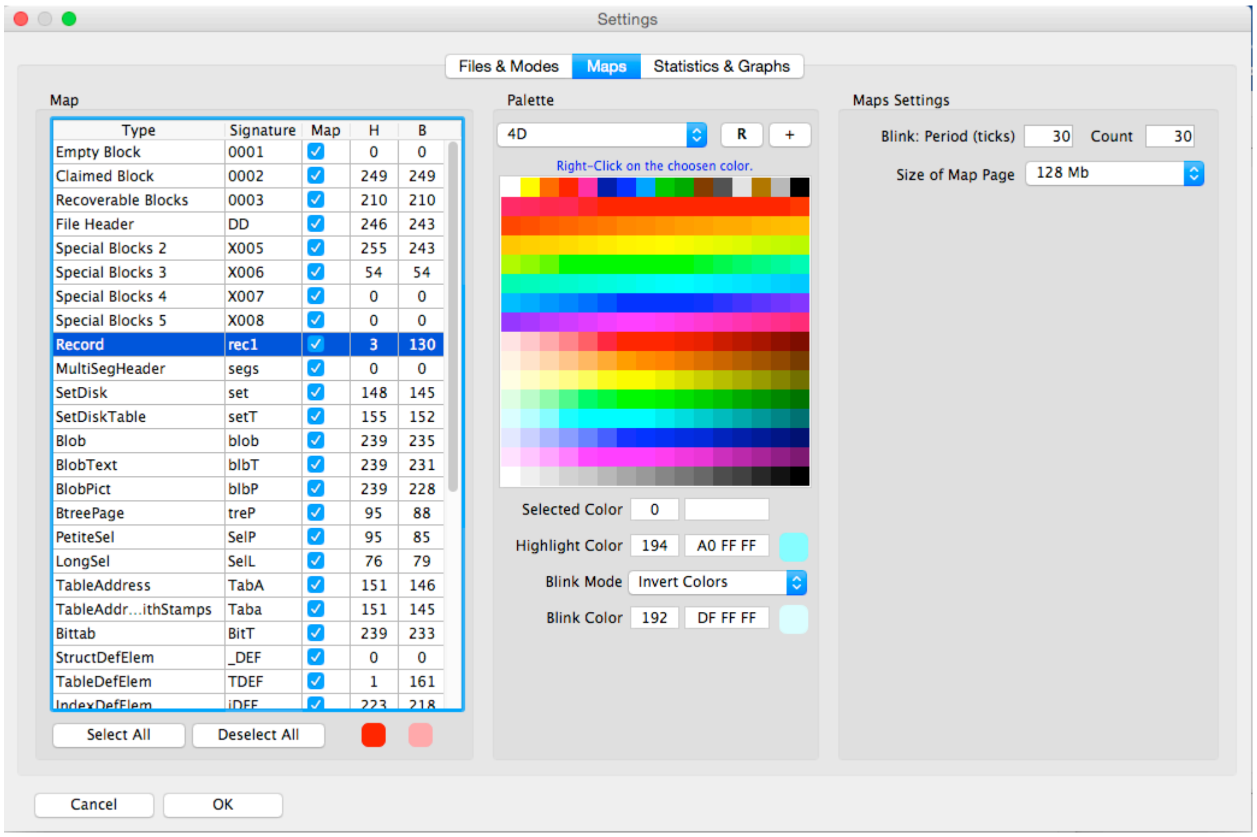
- Line 4: File Header. The 2 first Blocks of a file.
- Line 5 to 8: Special Blocks 2 to 5. Reserved for a future use in the next versions of DataAnalyzer.

The next lines correspond to 4D Objects. You get the Object Name, the Tag (in text, more readable than the 4 bytes hex value. Do not forget that the tag is inverted on disk, due to the use of Little-endian by the Intel CPUs. For instance, 'rec1' tag (for Record) is '1cer' on disk.

The last 6 lines correspond to normally unused Objects, but may be found in old versions of Data Files or Structure Files.

Check the Checkbox in the Map column if you want to map a type of Object.

The 2 columns H & B allow you to select which color you want to use for each kind of object. H is for Header and B is for Block. You can select a different color for the Object Header and for the following blocks of the same Object. The number in each column corresponds to an index in a Palette of 256 possible colors. When you select a line, the colors used by the corresponding Object are displayed at the bottom of the column.



Type	Signature	Map	H	B
Empty Block	0001	✓	0	0
Claimed Block	0002	✓	249	249
Recoverable Blocks	0003	✓	210	210
File Header	DD	✓	246	243
Special Blocks 2	X005	✓	255	243
Special Blocks 3	X006	✓	54	54
Special Blocks 4	X007	✓	0	0
Special Blocks 5	X008	✓	0	0
Record	rec1	✓	3	130
MultiSegHeader	segs	✓	0	0
SetDisk	set	✓	148	145
SetDiskTable	setT	✓	155	152
Blob	blob	✓	239	235
BlobText	blbT	✓	239	231
BlobPict	blbP	✓	239	228
BtreePage	treP	✓	95	88
PetiteSel	SelP	✓	95	85
LongSel	SelL	✓	76	79
TableAddress	TabA	✓	151	146
TableAddr...ithStamps	Taba	✓	151	145
Bittab	BitT	✓	239	233
StructDefElem	_DEF	✓	0	0
TableDefElem	TDEF	✓	1	161
IndexDefElem	iDEF	✓	223	218

Palette: 4D

Right-Click on the choosen color.

Selected Color: 0

Highlight Color: 194 A0 FF FF

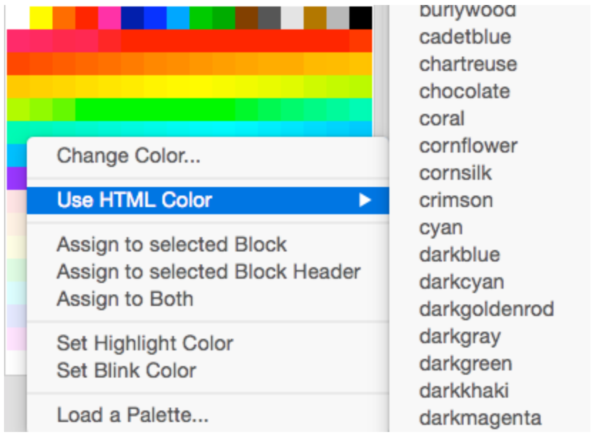
Blink Mode: Invert Colors

Blink Color: 192 DF FF FF

Blink: Period (ticks) 30 Count 30

Size of Map Page: 128 Mb

The middle part ('Palette') allows you to customize the Palette used by the Map. DataAnalyzer uses the Palette index, and not the direct color, but can save a different choice of indexes along with each Palette, so you can switch palettes without losing your selected indexes. A right-click on a color in the Palette opens the following Menu:



- Change Color... opens the Color Picker to choose the color to affect to the particular entry under the Mouse.
- Use HTML Color... allows the selection of a Color by it's name.
- Assign to the selected Block Header assigns the selected color to the Block Header of the selected line.
- Assign to selected Block assigns the selected color to the following Blocks of the selected line.
- Assign to Both assigns the selected color to both the Block Header and the following Blocks of the selected line.
- Set Highlight Color allows the choice of the color, which will used to highlight Objects on the Map.
- Set Blink Color allows the choice of the color, which will be used to Blink selected Objects on the Map.
- Load a Palette... to use to load a new Palette in DataAnalyzer.

Below the Palette is a popup menu to select which way will used to blink a part of the Map. The possibility of blinking blocks of a kind makes easier to find it instantly even on big maps.

The middle part ('Palette') allows you to customize the Palette used by the Map. There are 3 choices:

- Invert Colors will use the complement color for each block: Black for White, orange for green, etc.
- Use Blink Color will use the Blink Color.
- Use Complement to Blink Color will use a mix between Blink Color and Dot Color.
- DataAnalyzer: Analyze Data File

The right part (Maps Settings) allows you to select different parameters:

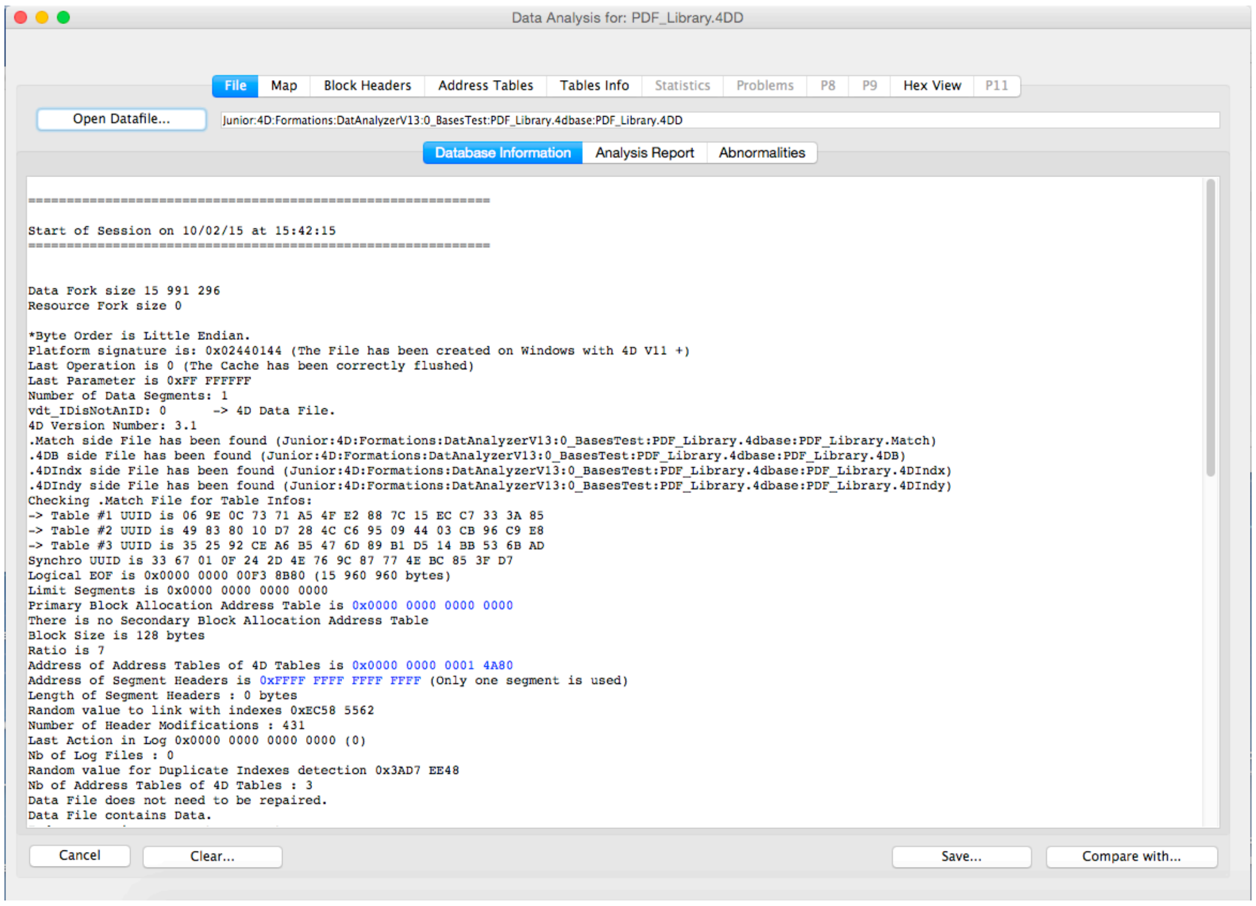
- Blink period in Ticks (1/60th s.) gives the blinking frequency.
- Blink Count will control the number of blinks before it stops.
- Size of Map Page allows you to select the size of Data, which will be displayed inside a single page. It goes from 128Mb to 1280Mb of Data File, i.e. from 1Mb to 10Mb of picture size, for the picture uses 1 dot for each 128-byte Block on disk. When the file size exceeds this value, DataAnalyzer creates as many pages as necessary to show the complete Map.

DataAnalyzer: Analyze Data File...

When choosing this menu, you enter the main DataAnalyzer screen. It has several pages, which will be accessible as soon as the progress of the analysis allows it.

Page 1 (File)

This page allows you to select the File you want to analyze. Click on the 'Open DataFile...' button, and select a 4D Data File (.4DD). A thermometer will show on top of the window, for the analysis is done asynchronously. As the thermometers end, the different tabs of the main Tab Bar will become accessible. As soon as one of these become selectable, you can use it. For instance, you can access to the Map before the end of Records analysis.



```
============================================================

Start of Session on 10/02/15 at 15:42:15
============================================================


Data Fork size 15 991 296
Resource Fork size 0

*Byte Order is Little Endian.
Platform signature is: 0x02440144 (The File has been created on Windows with 4D V11 +)
Last Operation is 0 (The Cache has been correctly flushed)
Last Parameter is 0xFF FFFFFF
Number of Data Segments: 1
vdt_IDisNotAnID: 0      -> 4D Data File.
4D Version Number: 3.1
.Match side File has been found (Junior:4D:Formations:DatAnalyzerV13:0_BasesTest:PDF_Library.4dbase:PDF_Library.Match)
.4DB side File has been found (Junior:4D:Formations:DatAnalyzerV13:0_BasesTest:PDF_Library.4dbase:PDF_Library.4DB)
.4DIndx side File has been found (Junior:4D:Formations:DatAnalyzerV13:0_BasesTest:PDF_Library.4dbase:PDF_Library.4DIndx)
.4DIndy side File has been found (Junior:4D:Formations:DatAnalyzerV13:0_BasesTest:PDF_Library.4dbase:PDF_Library.4DIndy)
Checking .Match File for Table Infos:
-> Table #1 UUID is 06 9E 0C 73 71 A5 4F E2 88 7C 15 EC C7 33 3A 85
-> Table #2 UUID is 49 83 80 10 D7 28 4C C6 95 09 44 03 CB 96 C9 E8
-> Table #3 UUID is 35 25 92 CE A6 B5 47 6D 89 B1 D5 14 BB 53 6B AD
Synchro UUID is 33 67 01 0F 24 2D 4E 76 9C 87 77 4E BC 85 3F D7
Logical EOF is 0x0000 0000 00F3 8B80 (15 960 960 bytes)
Limit Segments is 0x0000 0000 0000 0000
Primary Block Allocation Address Table is 0x0000 0000 0000 0000
There is no Secondary Block Allocation Address Table
Block Size is 128 bytes
Ratio is 7
Address of Address Tables of 4D Tables is 0x0000 0000 0001 4A80
Address of Segment Headers is 0xFFFF FFFF FFFF FFFF (Only one segment is used)
Length of Segment Headers : 0 bytes
Random value to link with indexes 0xEC58 5562
Number of Header Modifications : 431
Last Action in Log 0x0000 0000 0000 0000 (0)
Nb of Log Files : 0
Random value for Duplicate Indexes detection 0x3AD7 EE48
Nb of Address Tables of 4D Tables : 3
Data File does not need to be repaired.
Data File contains Data.
```

This Page can display 3 different kind of information: Database Information, Analysis Report, and Abnormalities. Each of the Reports named above can be saved as Text files for a further study (Button 'Save...'). In a next version, you will be able to compare with a previously saved report (of the same kind, obviously)

Database Information

This is the detailed content of the 256-byte header block. The text in Courier is the report, in which text in blue corresponds to important addresses. The text in *italic* is the comments. Let's have a look on it's content:

```
Data Fork size 15 991 296
    Physical size of the file on disk
Resource Fork size 0
    Not used anymore
*Byte Order is Little-endian.
    Always (until a next platform ?)
Platform signature is: 0x02440144 (The File has been created on Windows with 4D
V11 +)
    Windows means Intel CPU (Little-endian)
Last Operation is 0 (The Cache has been correctly flushed)
    If not 0, the file is suspected to have lost its integrity
```

```
Last Parameter is 0xFF FFFFFF
Number of Data Segments: 1
   Always 1
vdt_IDisNotAnID: 0      -> 4D Data File.
   Can be Wakanda File
4D Version Number: 3.1
   Last version used when the File has been closed

   Then DataAnalyzer will check for other files associated with this one, as
specified in Settings. If found, It will cross-reference information between
files:
.Match side File has been found
(Junior:4D:Formations:DatAnalyzerV13:0_BasesTest:PDF_Library.4dbase:PDF_Library.
Match)
.4DB side File has been found
(Junior:4D:Formations:DatAnalyzerV13:0_BasesTest:PDF_Library.4dbase:PDF_Library.
4DB)
.4DIndx side File has been found
(Junior:4D:Formations:DatAnalyzerV13:0_BasesTest:PDF_Library.4dbase:PDF_Library.
4DIndx)
.4DIndy side File has been found
(Junior:4D:Formations:DatAnalyzerV13:0_BasesTest:PDF_Library.4dbase:PDF_Library.
4DIndy)

   For instance, it will use information from the .Match file:
Checking .Match File for Table Infos:
-> Table #1 UUID is 06 9E 0C 73 71 A5 4F E2 88 7C 15 EC C7 33 3A 85
-> Table #2 UUID is 49 83 80 10 D7 28 4C C6 95 09 44 03 CB 96 C9 E8
-> Table #3 UUID is 35 25 92 CE A6 B5 47 6D 89 B1 D5 14 BB 53 6B AD
   There are 3 4D Tables
Synchro UUID is 33 67 01 0F 24 2D 4E 76 9C 87 77 4E BC 85 3F D7
Logical EOF is 0x0000 0000 00F3 8B80 (15 960 960 bytes)
   Logical end of file. Data after this address is garbage.

Limit Segments is 0x0000 0000 0000 0000
   There is no size limit for segments
Primary Block Allocation Address Table is 0x0000 0000 0000 0000
   Means there is no Secundary Block Allocation Address Table
There is no Secondary Block Allocation Address Table
   Confirmed here!
Block Size is 128 bytes
   Usual block size since V6.0
Ratio is 7
   The 7 LSBits are not used for the Address
Address of Address Tables of 4D Tables is 0x0000 0000 0001 4A80
   Where we will find the Addresses of the 3 4D Tables
Address of Segment Headers is 0xFFFF FFFF FFFF FFFF (Only one segment is used)
   -1 means Only one segment
Length of Segment Headers : 0 bytes
   No segment Headers
Random value to link with indexes 0xEC58 5562
   This value is used to synchronize Data and Indexes (.4Dindx)
Number of Header Modifications : 431
   Number of times the Header has been modified
Last Action in Log 0x0000 0000 0000 0000 (0)
   Means No Log in use
Nb of Log Files : 0
Random value for Duplicate Indexes detection 0x3AD7 EE48
Nb of Address Tables of 4D Tables : 3
   3 4D Tables in this Data File
Data File does not need to be repaired.
Data File contains Data.
Indexes are in a separate segment.
```

```
First Hole Position in Address Table 0xFFFF FFFF 88CA 6C00 (-2 000 000 000) = No
Hole
   No 4D Table has been deleted
4D Tables Headers Address Table 0x0000 0000 0001 7C00
   Where we will find the beginning of Table Headers
Nb of 4D Tables : 3

   The following is not used in a Data File
Nb of Relations : 0 (In case of Structure File)
First Hole Position in Relations Table 0xFFFF FFFF 88CA 6C00 (-2 000 000 000)
(In case of Structure File) = No Hole
Relations Table Address 0x0000 0000 0000 0000 (In case of Structure File)

   Informations about the Table of Sequence Numbers
Nb of Sequence Numbers : 1 (In case of Data File)
First Hole Position in Sequence Numbers Table 0xFFFF FFFF 88CA 6C00 (-2 000 000
000) (In case of Data File) = No Hole
Sequence Numbers Table Address 0x0000 0000 0007 5280 (In case of Data File)

   Informations regarding Indexes. Not used here.
Nb of Indexes : 8
First Hole Position in Index Definitions Table 0xFFFF FFFF 88CA 6C00 (-2 000 000
000) (In case of Structure File) = No Hole
Index Definitions Table Address 0x0000 0000 0006 DC00 (In case of Structure
File)
Nb of Indexes defined in structure : 0
First Hole Position in Index Definitions Table (in Structure) 0xFFFF FFFF 88CA
6C00 (-2 000 000 000) (In case of Structure File) = No Hole
Index Definitions Table Address (in Structure) 0x0000 0000 0006 DC00 (In case of
Structure File)

   Fields may have Extra Properties
Extra Properties Table Address 0x0000 0000 0000 0000 (0)
Extra Properties Size : 0 bytes.
First Hole Position in Address Tables Table 0xFFFF FFFF 88CA 6C00 (-2 000 000
000) = No Hole

Number of Data Flushes : 284 times.
   Number of times Data has been flushed
Dialect : 1036 = FRENCH
   Dialect used is French. Important because Indexes are based on it
With ICU : True
   Uses ICU (International Components for Unicode)
Ignore WildChar in middle : False
Uses secondary collation strength when matching strings : False
Uses quaternary collation strength when sorting strings : False
Hiragana code points will sort before everything else on the quaternary level :
False
Uses language neutral deadchar algorithm in FindWord and GetWordBoundaries. Else
uses ICU BreakIterator. : False
Use traditional-style sorting : False
   All this concerns Language Settings

   Then some statistics on this Data File
Start of Bit Allocation Table Analysis on 10/02/15 at 14:43:01

*** There is only one Allocation Bitmap Address Tables

*** Analyzing Allocation Bitmap Address Table #1
8 Bitmap(s) found on 8192 possible.
  Table #1 has 0 Records
  Table #2 has 138 Records
  Table #3 has 0 Records
  Table #1 has 0 Blobs
  Table #2 has 0 Blobs
  Table #3 has 0 Blobs
Total Number of Records = 138
Total Number of Blobs = 0
```

Analysis Report

This is the report describing the analysis of the Data File, with the time used for each phase, and what was the purpose of this phase:

```
============================================================
Start of Analysis on 10/02/15 at 15:42:15
============================================================
Start of Raw Blocks Analysis on 10/02/15 at 15:42:16

Start of Block Address Tables Analysis on 10/02/15 at 15:42:17
- Start of DTab Address Tables Analysis on 10/02/15 at 15:42:17
- Start of TDEF Address Tables Analysis on 10/02/15 at 15:42:17
- Start of iDEF Address Tables Analysis on 10/02/15 at 15:42:17
- Start of Seq1 Address Tables Analysis on 10/02/15 at 15:42:17
End of Block Address Tables Analysis on 10/02/15 at 15:42:17 in 0 days and
00:00:00s.
Start of Records & Blobs Address Tables Analysis on 10/02/15 at 15:42:17
- Start of Table 1 Records Tables parsing on 10/02/15 at 15:42:17
  Found 3 ATs and 0 Records in 00:00:00
- Start of Table 2 Records Tables parsing on 10/02/15 at 15:42:17
  Found 1 ATs and 138 Records in 00:00:00
- Start of Table 3 Records Tables parsing on 10/02/15 at 15:42:17
  Found 3 ATs and 0 Records in 00:00:00
- Start of Table 1 Blobs Tables parsing on 10/02/15 at 15:42:17
  Found 3 ATs and 0 Blobs in 00:00:00
- Start of Table 2 Blobs Tables parsing on 10/02/15 at 15:42:17
  Found 3 ATs and 0 Blobs in 00:00:00
- Start of Table 3 Blobs Tables parsing on 10/02/15 at 15:42:17
  Found 3 ATs and 0 Blobs in 00:00:00
End of Records & Blobs Address Tables Analysis on 10/02/15 at 15:42:17 in 0 days
and 00:00:00s.

Start of Records Information Analysis on 10/02/15 at 15:42:17
End of Records Information Analysis on 10/02/15 at 15:42:17 in 0 days and
00:00:00s.
```

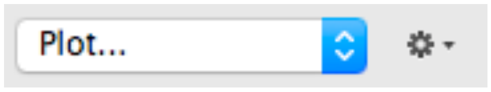
Abnormalities

Each time DataAnalyzer finds something 'strange' (i.e. not following the rules), it will be reported in this Report. For instance, in this Data File, coming from a V12 version; the TimeStamps have not been correctly calculated:

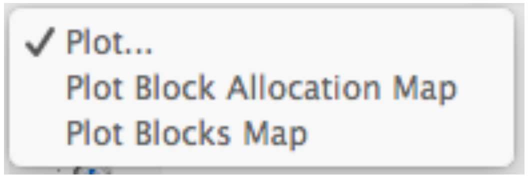
```
Table #2 Record #135 Block Nb 3 980 -> TimeStamp is not correct!
    The Record # is the one returned by 4D Function Record number
    To get the address from the blocl number, multiply by 128

Table #2 Record #136 Block Nb 4 014 -> TimeStamp is not correct!
Table #2 Record #140 Block Nb 4 029 -> TimeStamp is not correct!
Table #2 Record #141 Block Nb 4 055 -> TimeStamp is not correct!
Table #2 Record #142 Block Nb 4 096 -> TimeStamp is not correct!
Table #2 Record #156 Block Nb 4 160 -> TimeStamp is not correct!
Table #2 Record #225 Block Nb 4 210 -> TimeStamp is not correct!
Table #2 Record #219 Block Nb 4 261 -> TimeStamp is not correct!
```

Page 2 (Map)

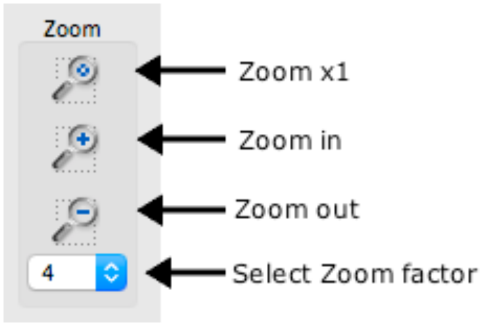
On this page will be displayed Maps of the opened file. To create a Map, use the 'Plot...' menu at the top-left corner.



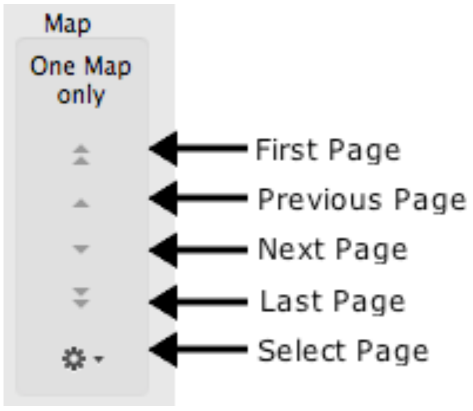
In DataAnalyzer V1, there are 2 kinds of Maps:



On each of these Maps, you can use the Toolbar on the left side of the Map:



Zoom Tool: Allows you to zoom up to 64 times. The Map represents blocks, one dot for one 128-byte block. Zooming allows you to display it from 1 pixel per block to 64*64 pixels per block.



Pages Tool: When a File is too big to be represented by a single Page (See Settings), it will be cut into multiple pages, and this tool allows you to navigate between the pages.

(The Map index is the line number of the block list. See Settings, 'Maps' tab)

Block Allocation Map

- It will display the content of the Block Allocation Tables. 4 colors can be used
- Empty Blocks (Map index 1): Blocks available (not in use) according to 4D.
- Claimed Blocks (Map index 2): Blocks allocated (in use) according to 4D.
- Blocks Allocated but not used (Blue): These blocks are supposed to be used, but the deep analysis didn't find them in any Object.
- Blocks used but not allocated (Red): These blocks are not supposed to be used, but the deep analysis found them used by at least one Object. This situation can be damaging, in case 4D will attempt to reuse it, therefore deleting the present data. A compress or repair of the Data file is strongly recommended.

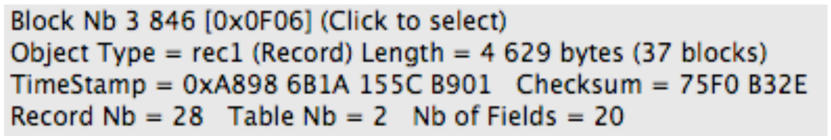
By using the Tool (*) menu on the right of the 'Plot...' menu, you can select to highlight the Recoverable Blocks.

- Recoverable Blocks (Map index 3): Blocks not allocated (not in use), but whose content is consistent with an Object. For instance, this is the case when a Record has been deleted. In case of Repair by Tags, these blocks can be 'undeleted' (or retrieved) by 4D.

Blocks Map

It will display the content of the File. Each dot (i.e. each Block) will receive a color according to its type and the corresponding Map index. Because each block has been read, a block with a valid Tag will be considered as the first block of an object (Object Header). In this header is a Length field, allowing DataAnalyzer to compute how many of the following blocks belong to this Object.

While moving the mouse over a Block, information on this block is displayed on top of the Map. If you let the mouse over a Block for a short while, more information will be displayed

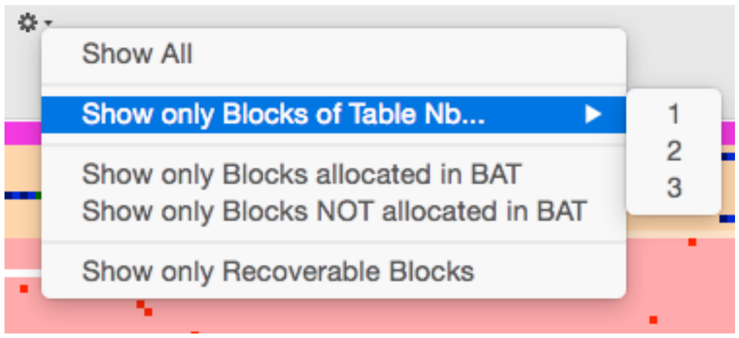


```
Block Nb 3 846 [0x0F06] (Click to select)
Object Type = rec1 (Record) Length = 4 629 bytes (37 blocks)
TimeStamp = 0xA898 6B1A 155C B901   Checksum = 75F0 B32E
Record Nb = 28   Table Nb = 2   Nb of Fields = 20
```

This is how to read this example:

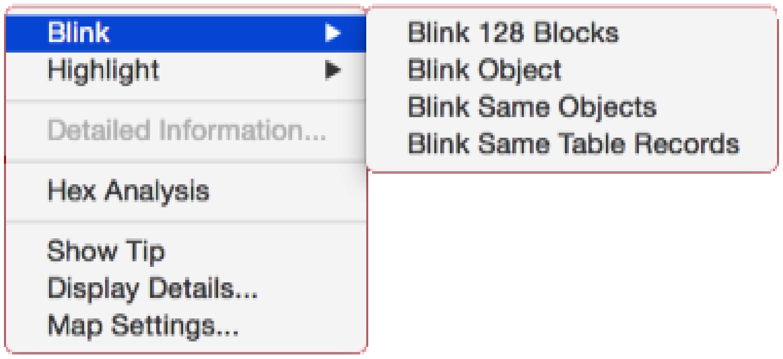
- This is the block number 3 846.
- It is the Header block of an Object of type 'rec1', thus a record, whose length is 37 blocks, and size is 4 629 bytes.
- It is the Record number 28 of Table number 2, and it has 20 Fields.

The Tool (*) menu on the right of the 'Plot...' menu displays now more options:


Show All
Show only Blocks of Table Nb...
1
2
3
Show only Blocks allocated in BAT
Show only Blocks NOT allocated in BAT
Show only Recoverable Blocks

- 'Show All' will display all the Blocks.
- 'Show only Blocks of Table Nb X' will display only Objects related to this Table (Records, Blobs of any type, etc.)
- 'Show only Blocks allocated in BAT' or 'Show only Blocks NOT allocated in BAT' will display (or not) Blocks that are considered as 'in use' by 4D.
- 'Show only recoverable Blocks' will display only the Objects that will be recovered in case of a 'Repair by Tags' operation.

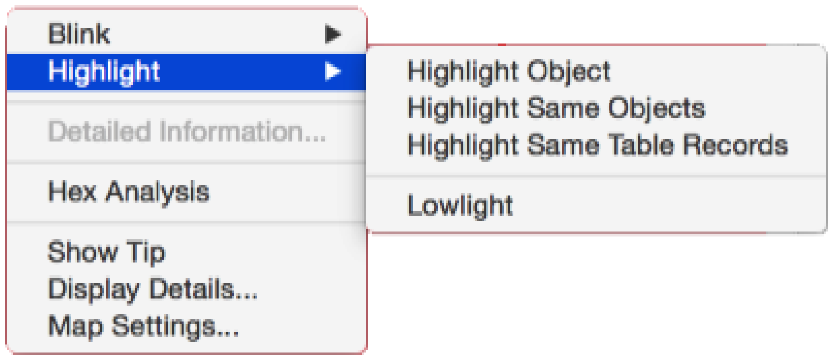
A right-click on any Block of the Map will open a contextual menu allowing you to have a better view on what is represented:



'Blink' will start blinking what you select:

- 'Blink 128 Blocks' will blink a set of 128 consecutive blocks starting with the one under the cursor.
- 'Blink Object' will blink all the blocks belonging to this Object.
- 'Blink Same Object' will blink all the Objects of the same kind.
- 'Blink Same Table Records' will blink all the records, which belong to the same Table.

Note: *Blink Settings (Color, duration, can be set in the 'Settings...' Menu).*



'Highlight' will highlight what you select:

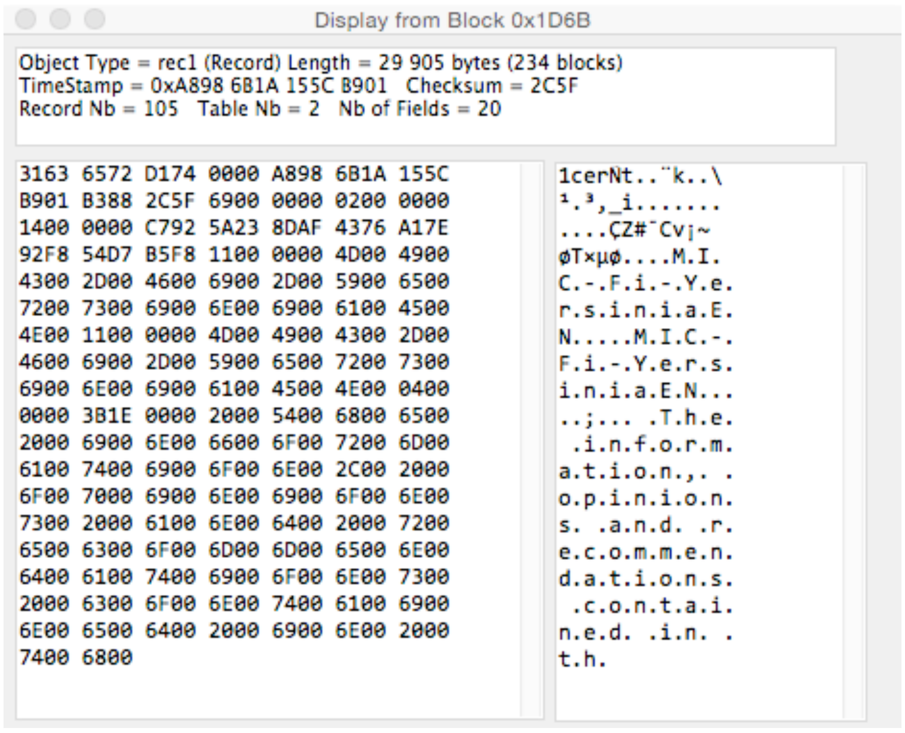
- 'Highlight Object' will highlight all the blocks belonging to this Object.
- 'Highlight U will highlight all the Objects of the same kind.
- 'Highlight Same Table Records' will highlight all the records, which belong to the same Table.
- 'Stop Highlight' will remove the highlight.

Note: *Highlight Settings (Color, mode, can be set in the 'Settings...' Menu).*

- 'Hex Analysis' will bring you on the page 'Hex View' directly on the Block you have clicked on.
- 'Show/Hide Tip' will display Block information as a Tip.
- 'Display Detail' will open a Palette Window to display more information regarding the Block under the Mouse pointer, as well as the Hexadecimal view of the content of the complete Block.(Similar as the Magnifying Glass button).
- 'Map Settings...' will open the Customization Page for this Map.

The 'Selected Block' field displays the number of the last block clicked.

The 'Magnifying Glass' Button will toggle the Hexadecimal view window. This window displays the block, which is currently under the Mouse Pointer.



```
Object Type = rec1 (Record) Length = 29 905 bytes (234 blocks)
TimeStamp = 0xA898 6B1A 155C B901   Checksum = 2C5F
Record Nb = 105   Table Nb = 2   Nb of Fields = 20
```

```
3163 6572 D174 0000 A898 6B1A 155C    1cerNt..¨k..\
B901 B388 2C5F 6900 0000 0200 0000    ¹.³,_i.......
1400 0000 C792 5A23 8DAF 4376 A17E    ....ÇZ#¯Cv¡~
92F8 54D7 B5F8 1100 0000 4D00 4900    øT×µø....M.I.
4300 2D00 4600 6900 2D00 5900 6500    C.-.F.i.-.Y.e.
7200 7300 6900 6E00 6900 6100 4500    r.s.i.n.i.a.E.
4E00 1100 0000 4D00 4900 4300 2D00    N.....M.I.C.-.
4600 6900 2D00 5900 6500 7200 7300    F.i.-.Y.e.r.s.
6900 6E00 6900 6100 4500 4E00 0400    i.n.i.a.E.N...
0000 3B1E 0000 2000 5400 6800 6500    ..;... .T.h.e.
2000 6900 6E00 6600 6F00 7200 6D00     .i.n.f.o.r.m.
6100 7400 6900 6F00 6E00 2C00 2000    a.t.i.o.n.,. .
6F00 7000 6900 6E00 6900 6F00 6E00    o.p.i.n.i.o.n.
7300 2000 6100 6E00 6400 2000 7200    s. .a.n.d. .r.
6500 6300 6F00 6D00 6D00 6500 6E00    e.c.o.m.m.e.n.
6400 6100 7400 6900 6F00 6E00 7300    d.a.t.i.o.n.s.
2000 6300 6F00 6E00 7400 6100 6900     .c.o.n.t.a.i.
6E00 6500 6400 2000 6900 6E00 2000    n.e.d. .i.n. .
7400 6800                             t.h.
```

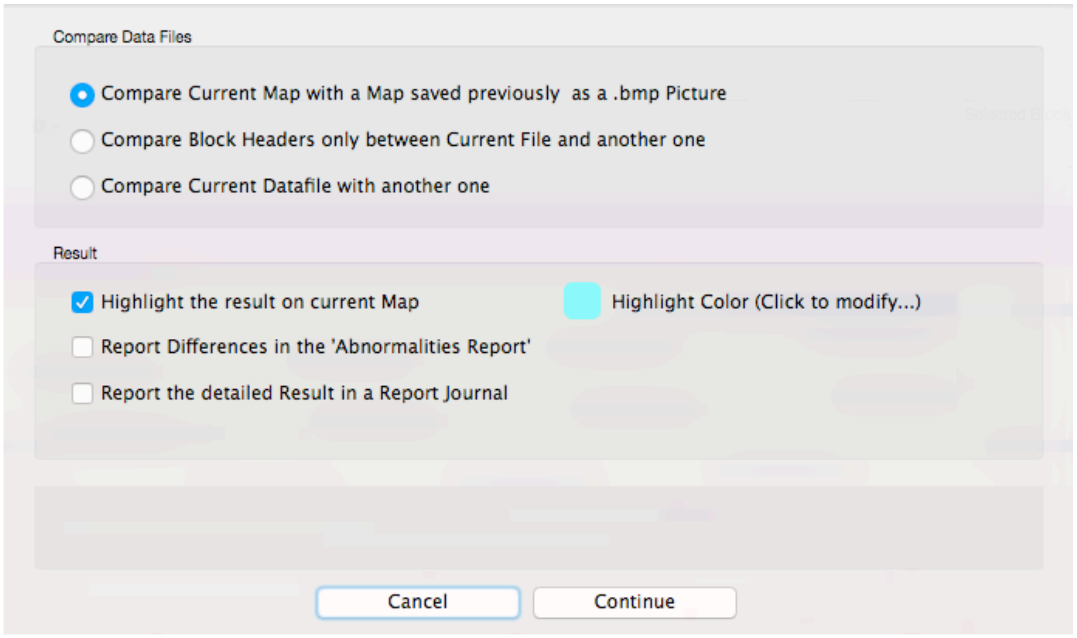
On top is information about the block itself. Below, you will find on left a hexadecimal view of the block content, as well as the next one. On right, there is an ASCII view of the same data. In this example, we can see that this is the block header of an Object of type 'rec1' (Record), the first 4 bytes are '1cer', which is 'rec1' in Little-endian. This record is the Record number 105 of the Table number 2, and it has 20 fields.

The 'Copy' Button puts the current Map into the Clipboard, with the type of a Picture, and with the present Zoom.

The 'Save' Button saves into a File, the Map itself, plus information regarding the content of the Data File. The type of this File is '.4DAnzer'. It can be used later to compare with another Map.

The 'Compare' Button allows you to compare the present Map, with an older version that has been saved using the 'Save' Button. This is very useful when looking for the result of a specific action. You can get a Map of the Data File, then save it, then start the application, perform the action, and quit. You can reopen that Data File with Data Analyzer, get the Map, and compare it with the previously saved one.

This comparison can be customized thru the following Dialog:



Compare Data Files

- [✓] Compare Current Map with a Map saved previously as a .bmp Picture
- [ ] Compare Block Headers only between Current File and another one
- [ ] Compare Current Datafile with another one

Result

- [✓] Highlight the result on current Map
- Highlight Color (Click to modify...)
- [ ] Report Differences in the 'Abnormalities Report'
- [ ] Report the detailed Result in a Report Journal

'Compare Current Map with a Map saved previously as a .bmp Picture'

You will select a .bmp file and DataAnalyzer will compare the 2 pictures bit per bit, then Highlight or Report the differences.

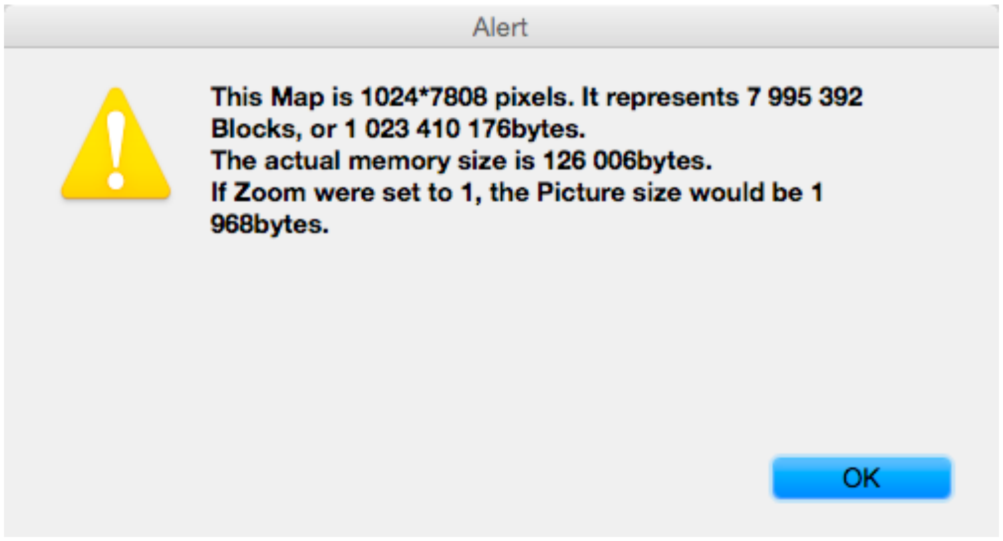
'Compare Block Headers only between Current File and another one'

You will open a .4DAnzer file, and DataAnalyzer will compare the 2 pictures bit per bit, as well as information regarding all the Block Headers, then Highlight or Report the differences.

'Compare Current Datafile with another one'

You can open a Data File, and DataAnalyzer will compare it with the currently opened one. If a .4DAnzer file exists, you will be asked to open it, for it will process the analysis faster. Otherwise, the Data File itself will be scanned in order to find every difference between header blocks.

The 'Help' Button will give some useful information about the Size of the Map if you want to save it, or to use the Map Picture, depending of the present Zoom factor:

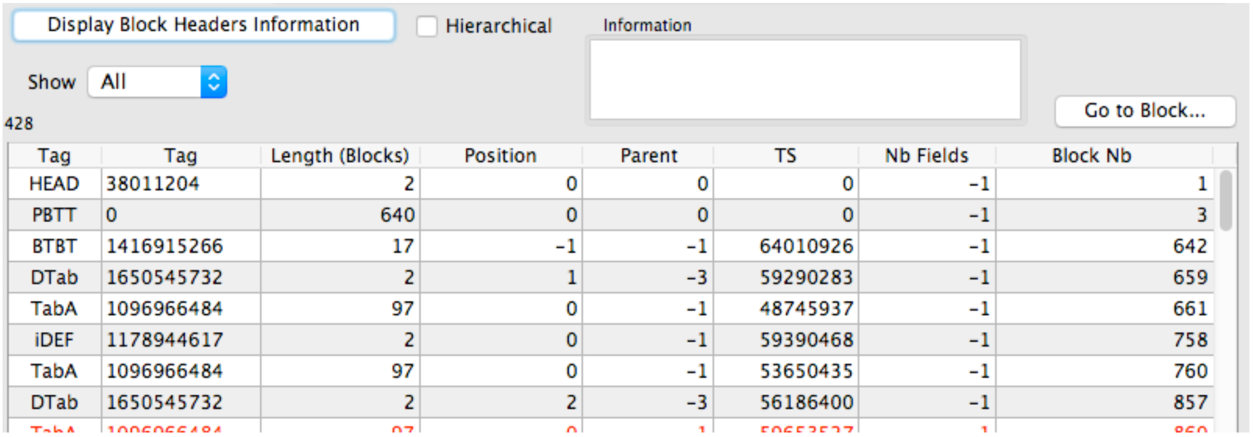


Alert

This Map is 1024*7808 pixels. It represents 7 995 392 Blocks, or 1 023 410 176bytes.
The actual memory size is 126 006bytes.
If Zoom were set to 1, the Picture size would be 1 968bytes.

Page 3 (Block Headers)

This page displays a list of all the Blocks that have been identified as possible Headers by Tag Analysis. Click on the 'Display Block Headers Information' button, and the list will be displayed. The red lines correspond to valid headers, which are not recognized as 'in use' (Claimed) by the Bit Allocation Tables.



428

Tag	Tag	Length (Blocks)	Position	Parent	TS	Nb Fields	Block Nb
HEAD	38011204	2	0	0	0	-1	1
PBTT	0	640	0	0	0	-1	3
BTBT	1416915266	17	-1	-1	64010926	-1	642
DTab	1650545732	2	1	-3	59290283	-1	659
TabA	1096966484	97	0	-1	48745937	-1	661
iDEF	1178944617	2	0	-1	59390468	-1	758
TabA	1096966484	97	0	-1	53650435	-1	760
DTab	1650545732	2	2	-3	56186400	-1	857

You can view the list as a whole, or through a Hierarchical view, by checking the box 'Hierarchical'.

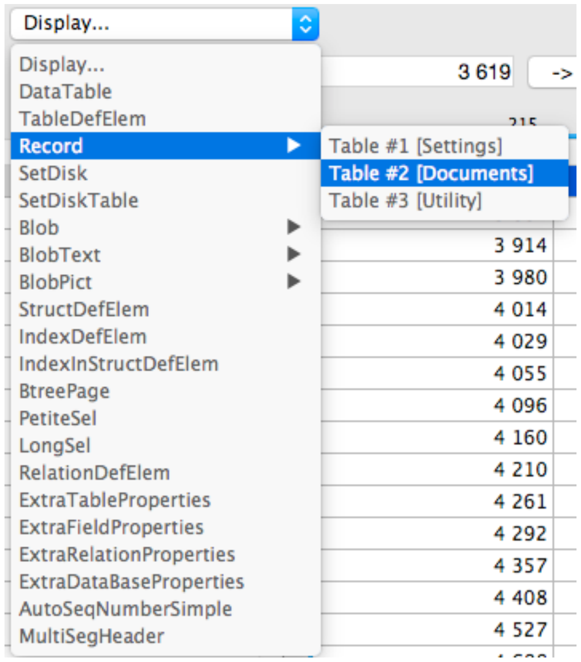
The 'Show' Menu allows you to select what you want to see.

When you select a line, the 'Information' Field displays information about the type of Block header you have selected.

When a line is selected, the 'Go to Block...' Button brings you directly to the 'Hex View' Page with the Block that has been selected.

Page 4 (Address Tables)

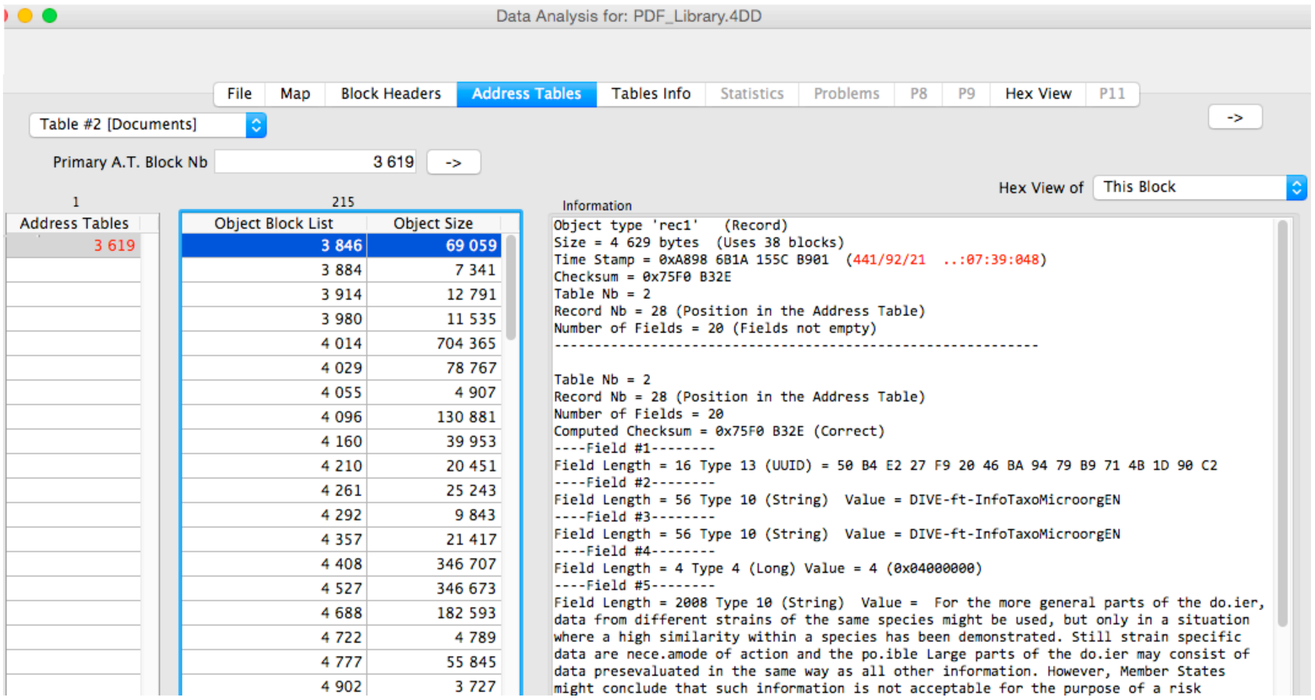
After page 3, the information displayed on the different pages doesn't come from the Blocks analysis, but from the Objects analysis. Page 4 displays a list of every type of Objects that have been found in the Data File.



The top-left Menu ('Display...') allows you to select which type of Object you want to see. Selecting 'Record, for instance, and then Table #2, will bring you to the following screen.

On the left, you can see the list of the Address Tables. The Primary Address Table Block Number is displayed on top of the lists.

On the middle part, you get the list on the Block Headers of every Object of this type, with its size.



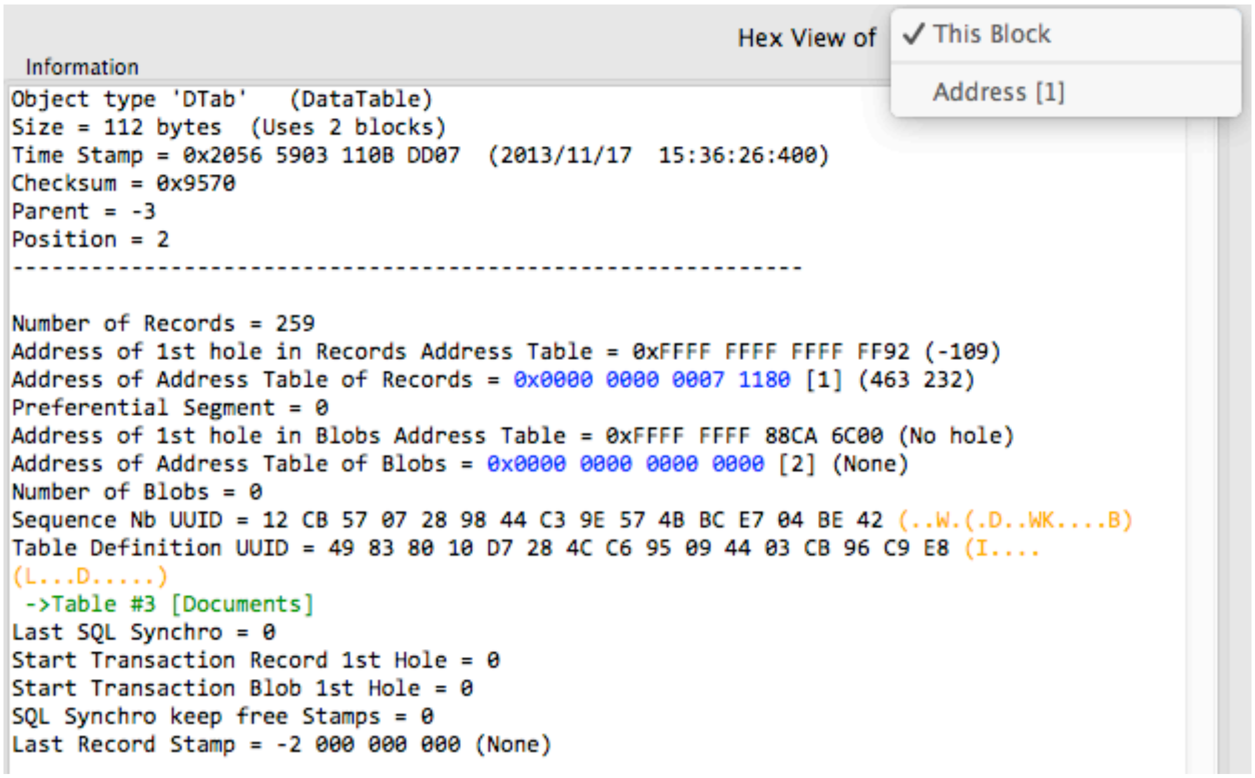
Address Tables
3 619

Object Block List	Object Size
3 846	69 059
3 884	7 341
3 914	12 791
3 980	11 535
4 014	704 365
4 029	78 767
4 055	4 907
4 096	130 881
4 160	39 953
4 210	20 451
4 261	25 243
4 292	9 843
4 357	21 417
4 408	346 707
4 527	346 673
4 688	182 593
4 722	4 789
4 777	55 845
4 902	3 727

```
Object type 'rec1'   (Record)
Size = 4 629 bytes  (Uses 38 blocks)
Time Stamp = 0xA898 6B1A 155C B901  (441/92/21  ..:07:39:048)
Checksum = 0x75F0 B32E
Table Nb = 2
Record Nb = 28 (Position in the Address Table)
Number of Fields = 20 (Fields not empty)
------------------------------------------------------------

Table Nb = 2
Record Nb = 28 (Position in the Address Table)
Number of Fields = 20
Computed Checksum = 0x75F0 B32E (Correct)
----Field #1--------
Field Length = 16 Type 13 (UUID) = 50 B4 E2 27 F9 20 46 BA 94 79 B9 71 4B 1D 90 C2
----Field #2--------
Field Length = 56 Type 10 (String)  Value = DIVE-ft-InfoTaxoMicroorgEN
----Field #3--------
Field Length = 56 Type 10 (String)  Value = DIVE-ft-InfoTaxoMicroorgEN
----Field #4--------
Field Length = 4 Type 4 (Long) Value = 4 (0x04000000)
----Field #5--------
Field Length = 2008 Type 10 (String)  Value =  For the more general parts of the do.ier,
data from different strains of the same species might be used, but only in a situation
where a high similarity within a species has been demonstrated. Still strain specific
data are nece.amode of action and the po.ible Large parts of the do.ier may consist of
data presevaluated in the same way as all other information. However, Member States
might conclude that such information is not acceptable for the purpose of a risk
```

The right part shows the information regarding this Object. It begins with the information from the Block Header. When something is not normal, it is displayed in red (For instance, in this example, the Timestamp is incorrect).

Then DataAnalyzer tries to 'understand' the content of the Object. In the case of a Record, you can see information regarding the Record itself, followed by the list of the Fields, with their Type and Content.



```
Object type 'DTab'   (DataTable)
Size = 112 bytes  (Uses 2 blocks)
Time Stamp = 0x2056 5903 110B DD07  (2013/11/17  15:36:26:400)
Checksum = 0x9570
Parent = -3
Position = 2
------------------------------------------------------------

Number of Records = 259
Address of 1st hole in Records Address Table = 0xFFFF FFFF FFFF FF92 (-109)
Address of Address Table of Records = 0x0000 0000 0007 1180 [1] (463 232)
Preferential Segment = 0
Address of 1st hole in Blobs Address Table = 0xFFFF FFFF 88CA 6C00 (No hole)
Address of Address Table of Blobs = 0x0000 0000 0000 0000 [2] (None)
Number of Blobs = 0
Sequence Nb UUID = 12 CB 57 07 28 98 44 C3 9E 57 4B BC E7 04 BE 42 (..W.(.D..WK....B)
Table Definition UUID = 49 83 80 10 D7 28 4C C6 95 09 44 03 CB 96 C9 E8 (I....
(L...D.....)
 ->Table #3 [Documents]
Last SQL Synchro = 0
Start Transaction Record 1st Hole = 0
Start Transaction Blob 1st Hole = 0
SQL Synchro keep free Stamps = 0
Last Record Stamp = -2 000 000 000 (None)
```

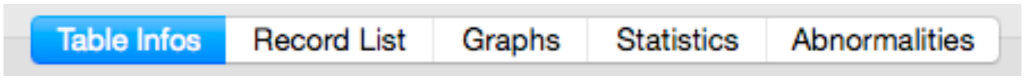
The screen above shows the analysis of a Data Table ('DTab') Object.

The top-right Menu allows you to go directly to the 'Hex View' page (at the beginning of the Object). When the Object contains an Address (in blue), and when this address makes sense, it is displayed also in the Menu and you can go directly to this address. For instance, in the example above, the address index [1] (Address of the Address Table of Records) is valid, and you can go directly to a Hex View of it by choosing Address [1] in the menu.

Page 5 (Tables Info)

While page 4 was displaying information regarding every type of Blocks, the page 5 is dedicated to 4D Tables. This page applies only to 4D or Wakanda Data Files.

This page has 5 subpages, but only the first 3 are working in this version.



All these subpages share the same 'Table' menu, allowing you to select which 4D Table you want to work on.

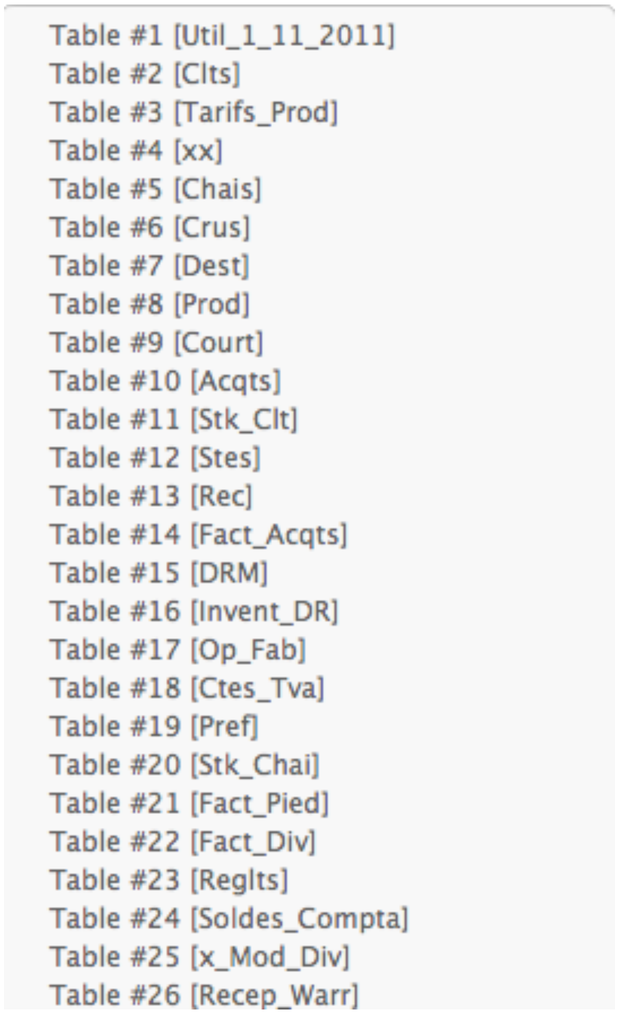
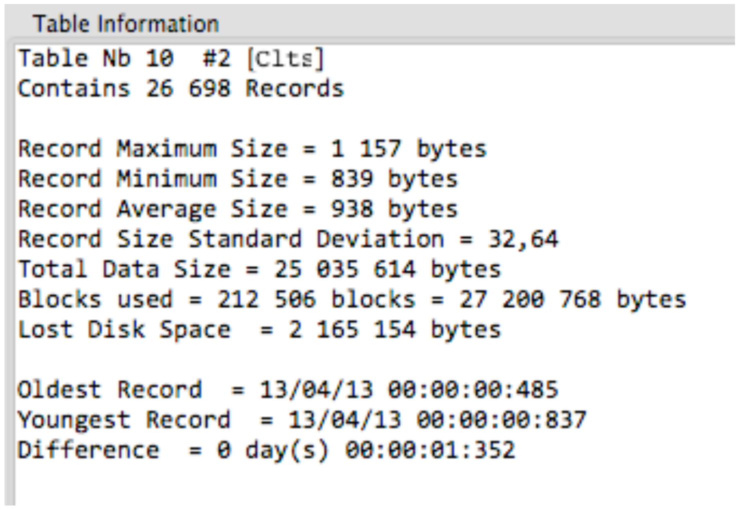


Table #1 [Util_1_11_2011]
Table #2 [Clts]
Table #3 [Tarifs_Prod]
Table #4 [xx]
Table #5 [Chais]
Table #6 [Crus]
Table #7 [Dest]
Table #8 [Prod]
Table #9 [Court]
Table #10 [Acqts]
Table #11 [Stk_Clt]
Table #12 [Stes]
Table #13 [Rec]
Table #14 [Fact_Acqts]
Table #15 [DRM]
Table #16 [Invent_DR]
Table #17 [Op_Fab]
Table #18 [Ctes_Tva]
Table #19 [Pref]
Table #20 [Stk_Chai]
Table #21 [Fact_Pied]
Table #22 [Fact_Div]
Table #23 [Reglts]
Table #24 [Soldes_Compta]
Table #25 [x_Mod_Div]
Table #26 [Recep_Warr]

The 'Table Infos' subpage gives you information regarding the specific selected Table, as well as Statistics, and Abnormalities:



```
Table Nb 10  #2 [Clts]
Contains 26 698 Records

Record Maximum Size = 1 157 bytes
Record Minimum Size = 839 bytes
Record Average Size = 938 bytes
Record Size Standard Deviation = 32,64
Total Data Size = 25 035 614 bytes
Blocks used = 212 506 blocks = 27 200 768 bytes
Lost Disk Space  = 2 165 154 bytes

Oldest Record  = 13/04/13 00:00:00:485
Youngest Record  = 13/04/13 00:00:00:837
Difference  = 0 day(s) 00:00:01:352
```

After the Table Number and Name, you will find the number of Records.

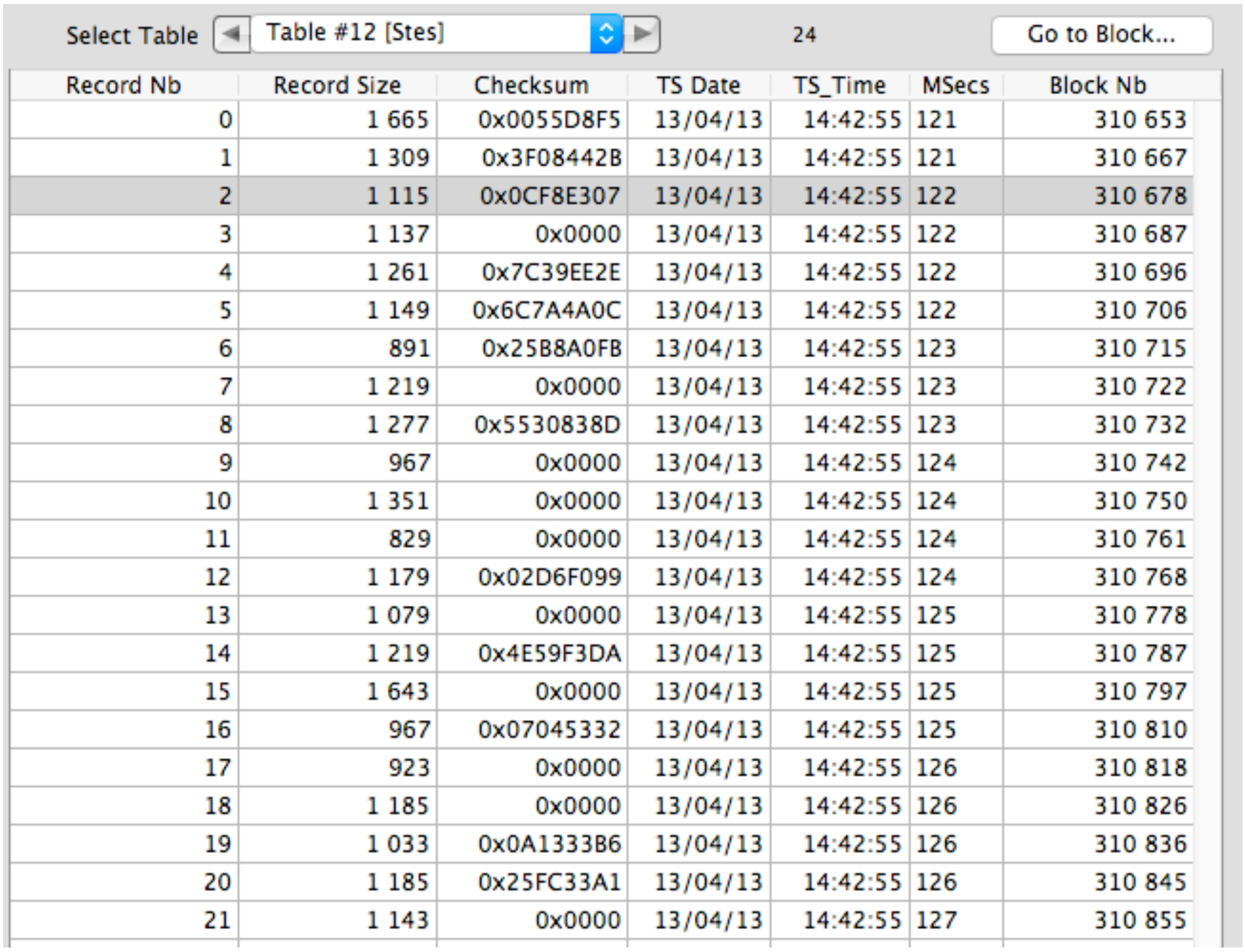
Then you will get the Maximum, Minimum, and Average size of the Records, the total size used by the records, and the actual size used on disk. The Lost Disk Space is the total of the size lost because the last block of each record is not completely filled. This value is not important in case of a small number of big records, but can be very big in case of big number of small records (For instance, relational tables used in N-to-N relations.

Then, there is information regarding the Records Age: The age of a record is calculated with the difference between the oldest and the youngest Timestamp. With versions prior to V14, this Timestamp was reset every time you compacted the Data File. From V14 and later versions, you can trust the Timestamp, which is the last time the record has been saved on disk. Old records mean records that are seldom modified. Young records have just been created, or saved. This information may be important for establishing a Backup strategy.

In the example above, the analysis of ages show that the records have been created automatically, or imported all at once, or the data file has been compacted, for there are the only ways to get 27,000 records updated in less than 2 seconds...

If abnormalities are found in this Table (Incorrect Timestamp, incorrect Checksum, 'strange' fields, etc.), it will be displayed in red after the statistics.

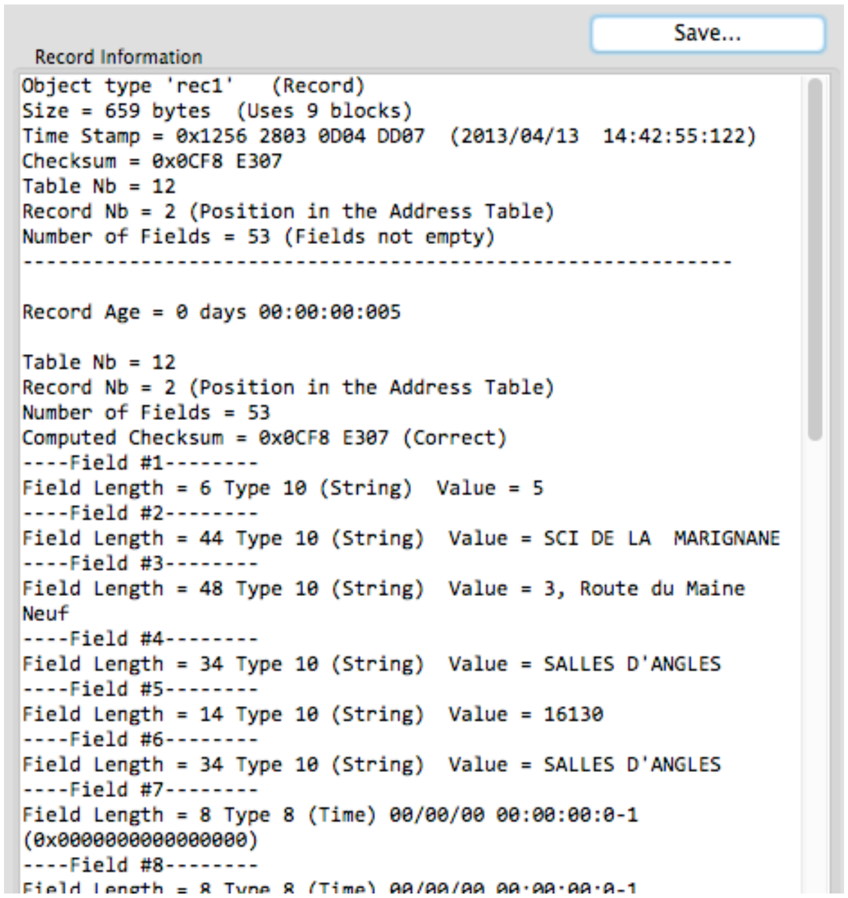
The 'Record List' subpage gives you the list of this Table's Records. On the left part, you will find the list itself:



Record Nb	Record Size	Checksum	TS Date	TS_Time	MSecs	Block Nb
0	1 665	0x0055D8F5	13/04/13	14:42:55	121	310 653
1	1 309	0x3F08442B	13/04/13	14:42:55	121	310 667
2	1 115	0x0CF8E307	13/04/13	14:42:55	122	310 678
3	1 137	0x0000	13/04/13	14:42:55	122	310 687
4	1 261	0x7C39EE2E	13/04/13	14:42:55	122	310 696
5	1 149	0x6C7A4A0C	13/04/13	14:42:55	122	310 706
6	891	0x25B8A0FB	13/04/13	14:42:55	123	310 715
7	1 219	0x0000	13/04/13	14:42:55	123	310 722
8	1 277	0x5530838D	13/04/13	14:42:55	123	310 732
9	967	0x0000	13/04/13	14:42:55	124	310 742
10	1 351	0x0000	13/04/13	14:42:55	124	310 750
11	829	0x0000	13/04/13	14:42:55	124	310 761
12	1 179	0x02D6F099	13/04/13	14:42:55	124	310 768
13	1 079	0x0000	13/04/13	14:42:55	125	310 778
14	1 219	0x4E59F3DA	13/04/13	14:42:55	125	310 787
15	1 643	0x0000	13/04/13	14:42:55	125	310 797
16	967	0x07045332	13/04/13	14:42:55	125	310 810
17	923	0x0000	13/04/13	14:42:55	126	310 818
18	1 185	0x0000	13/04/13	14:42:55	126	310 826
19	1 033	0x0A1333B6	13/04/13	14:42:55	126	310 836
20	1 185	0x25FC33A1	13/04/13	14:42:55	126	310 845
21	1 143	0x0000	13/04/13	14:42:55	127	310 855

This list shows the Record Number (in 4D, the result of **`Record number`** function), the Record size, the Checksum, the Timestamp date, time, and milliseconds, and the Block number. If you select a line and click on the 'Go to Block...' button, you will go to the 'Hex View' page at the beginning of this record.

On the right part of the page, as soon as you select a line in the list, you get the Record content interpreted in text, header and fields. For each field, you get the field number, the length (size of the field in the record), the field type, and the value (field content).

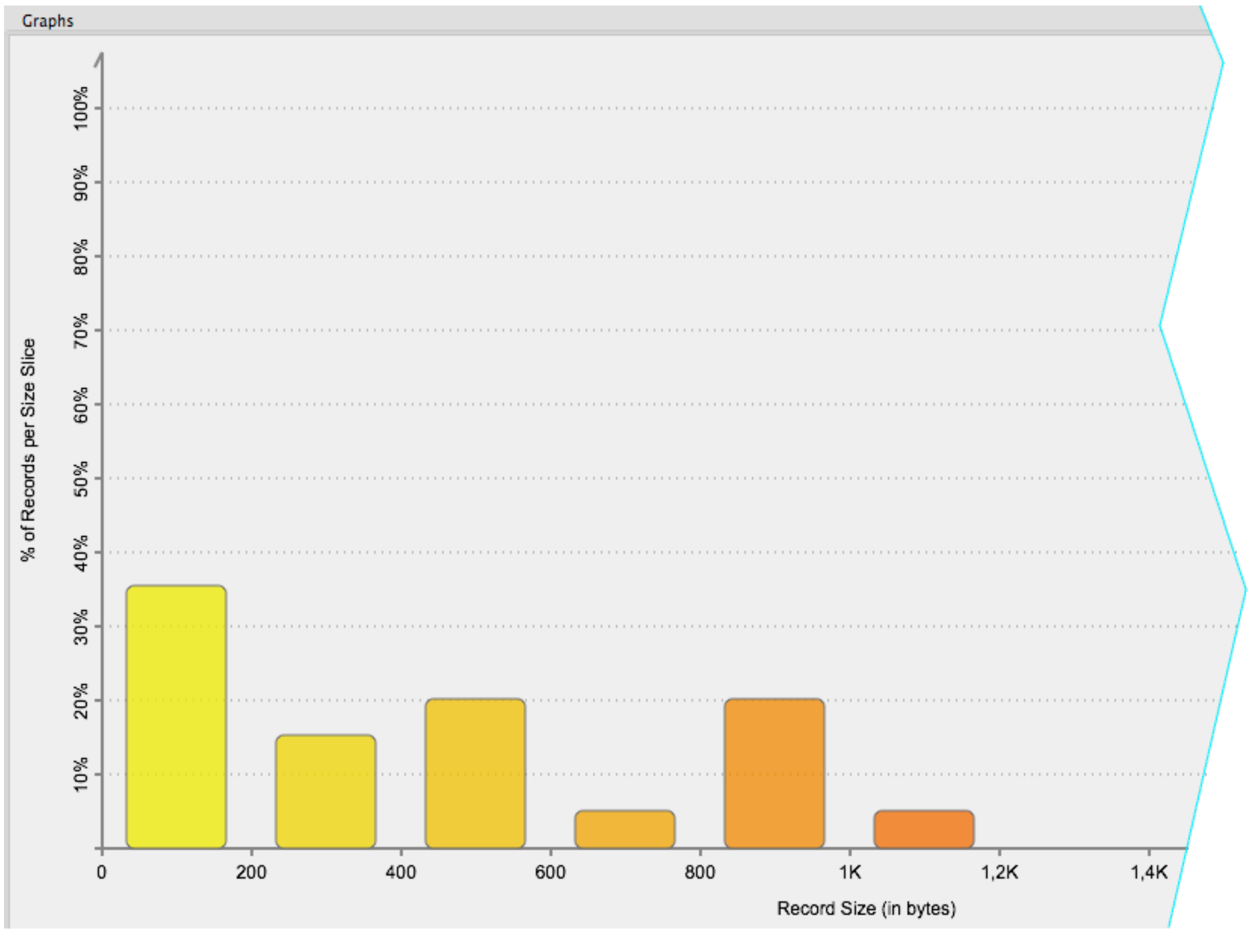


```
Object type 'rec1'   (Record)
Size = 659 bytes  (Uses 9 blocks)
Time Stamp = 0x1256 2803 0D04 DD07  (2013/04/13  14:42:55:122)
Checksum = 0x0CF8 E307
Table Nb = 12
Record Nb = 2 (Position in the Address Table)
Number of Fields = 53 (Fields not empty)
------------------------------------------------------------

Record Age = 0 days 00:00:00:005

Table Nb = 12
Record Nb = 2 (Position in the Address Table)
Number of Fields = 53
Computed Checksum = 0x0CF8 E307 (Correct)
----Field #1--------
Field Length = 6 Type 10 (String)  Value = 5
----Field #2--------
Field Length = 44 Type 10 (String)  Value = SCI DE LA  MARIGNANE
----Field #3--------
Field Length = 48 Type 10 (String)  Value = 3, Route du Maine
Neuf
----Field #4--------
Field Length = 34 Type 10 (String)  Value = SALLES D'ANGLES
----Field #5--------
Field Length = 14 Type 10 (String)  Value = 16130
----Field #6--------
Field Length = 34 Type 10 (String)  Value = SALLES D'ANGLES
----Field #7--------
Field Length = 8 Type 8 (Time) 00/00/00 00:00:00:0-1
(0x0000000000000000)
----Field #8--------
Field Length = 8 Type 8 (Time) 00/00/00 00:00:00:0-1
```

If inconsistencies are found, it will be displayed in red.

The 'Graphs' subpage is the beginning of what will become an extensive statistics view of the Data File. Beside the Table menu, there are 2 Graph menus.

This first one allows selection of what you want to see. On this version, only 'Records size' is available. It gives the percentage of records for each slice of size. Moving the mouse over the columns displays more information on top of the graph.



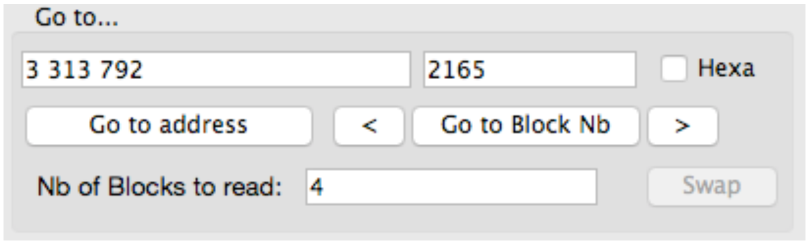
This graph allows you to visualize the dispersion of sizes. If the records have very variable sizes, and if they are subject to many modifications, it increases the risk of data fragmentation, for instance.

The second menu will allow you, in later versions, to choose the kind of graph you want to get.

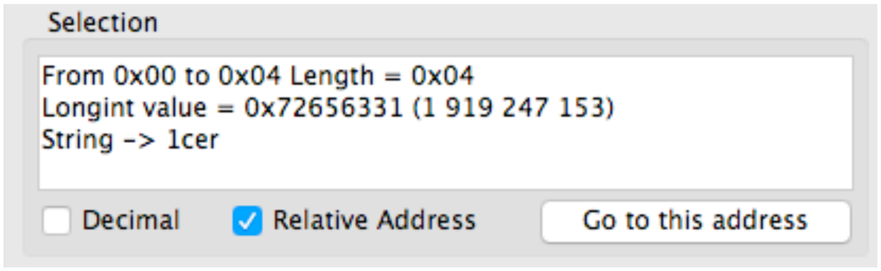
Page 10 (Hex View)

This page is a really low level microscope, which allows you to read the data byte per byte if necessary. It has to be used only in case of real trouble caused by a non-obvious cause. Use it suppose to have a deep knowledge of the way DB4D accesses the disk, and of the fine structure of the DB4D Files.

Basically, this page shows two Areas, the left one shows Hexadecimal values, and the right one ASCII text. On top of these, there are several parts handling the bottom areas.



The top-right 'Go to...' Groupbox indicates which block to read. You can give the Address or the Block number, and click on one of the 2 buttons 'Go to address' or 'Go to Block Nb'. Address or Block number can be entered in Decimal or Hexadecimal (if the 'Hexa' checkbox is checked). The 2 arrows beside the 'Go to Block Nb' button allow you to read previous or next block. The 'Nb of Blocks to read' area indicates how many consecutive blocks should be read at once. If you Copy-Paste an address or block number in Hexa, the 'Swap' button allows you to byte-swap it, because addresses on disk are in Little-endian, but we usually use Big-endian when talking about addresses.



```
From 0x00 to 0x04 Length = 0x04
Longint value = 0x72656331 (1 919 247 153)
String -> 1cer
```

The top-left 'Selection' Groupbox is related to the text selected in either of areas below it. When you select a text in either area (Hex or ASCII), the corresponding text is automatically highlighted in the other one. Thus, the 2 selections are corresponding. DataAnalyzer tries to interpret this selection, depending on how many bytes have been selected:

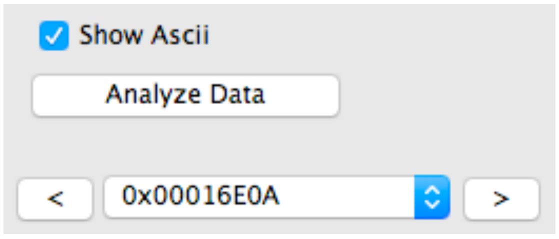
- 1 byte: Byte value represented as hexadecimal, decimal, and character.
- 2 bytes: Integer value represented as hexadecimal, decimal, and string.
- 3-4 bytes: Longint value represented as a hexadecimal, decimal, and string.
- 8 bytes: Long8 value represented as a hexadecimal, decimal, and string.
- >8 bytes: Data. In this case, only the length is reported: (From 0x02 to 0x0A = 0x08)

Up to 8 bytes, the data read on disk is automatically byte swapped to convert it from Little-Endian to Big-Endian, before numeric evaluation.

The 'Decimal' checkbox displays the values in Decimal.

The 'Relative Address' gives the From and To addresses relative to the beginning of Block. If unchecked, it gives these addresses relatively to the beginning of file.

If the length of the selection is compatible with an address on disk, the 'Go to this address' button is enabled and can be used to jump to this address.

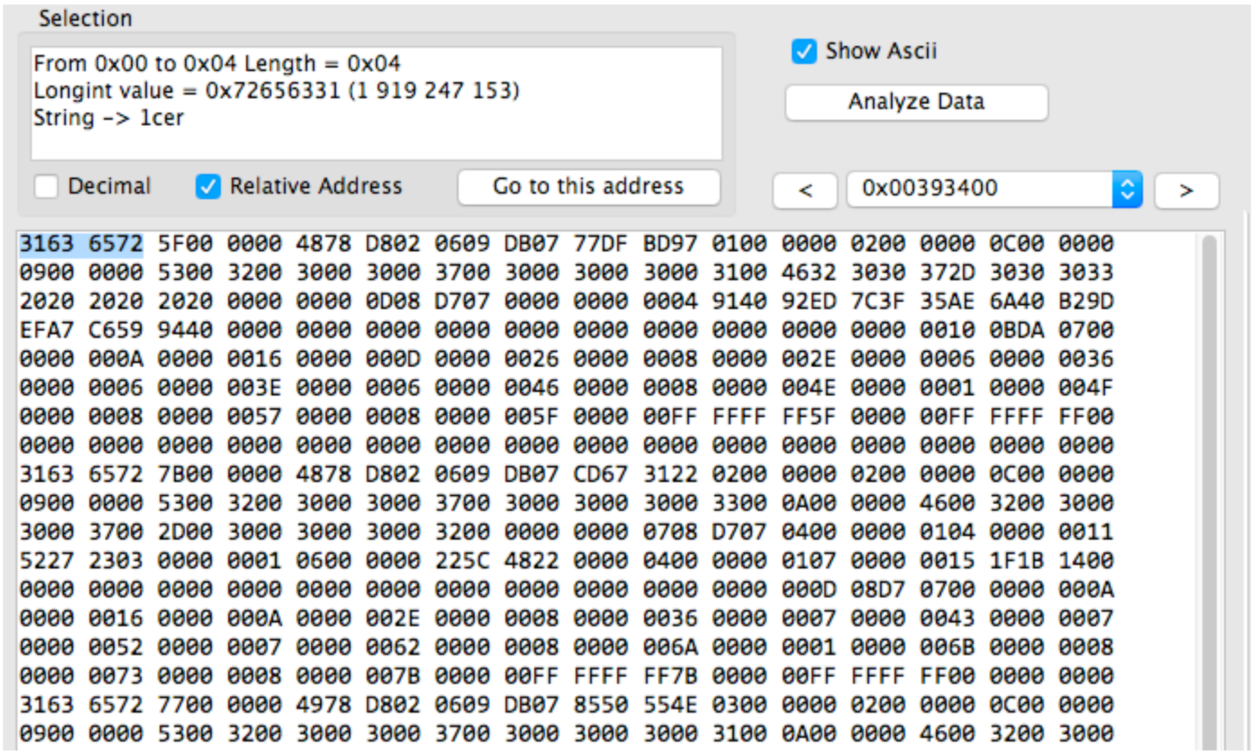


At the top of the middle part, the 'Show Ascii' checkbox controls the presence of the ASCII display.

The 'Analyze Data' button is not used yet.

At the bottom of this part, the Menu shows the list of every address, which has been accessed during this session. It allows you to return easily to one of the previous places. This is very useful when you 'drill in' the data, for instance when you start from a primary address table to a secondary address table, then to an object, and you want to go back to the next object or address table. It is a kind of 'Open recent' menu.

The left text area is the hexadecimal representation of the data that has been load:



```
From 0x00 to 0x04 Length = 0x04
Longint value = 0x72656331 (1 919 247 153)
String -> 1cer
```

Show Ascii: ✓ · Decimal: ☐ · Relative Address: ✓ · 0x00393400

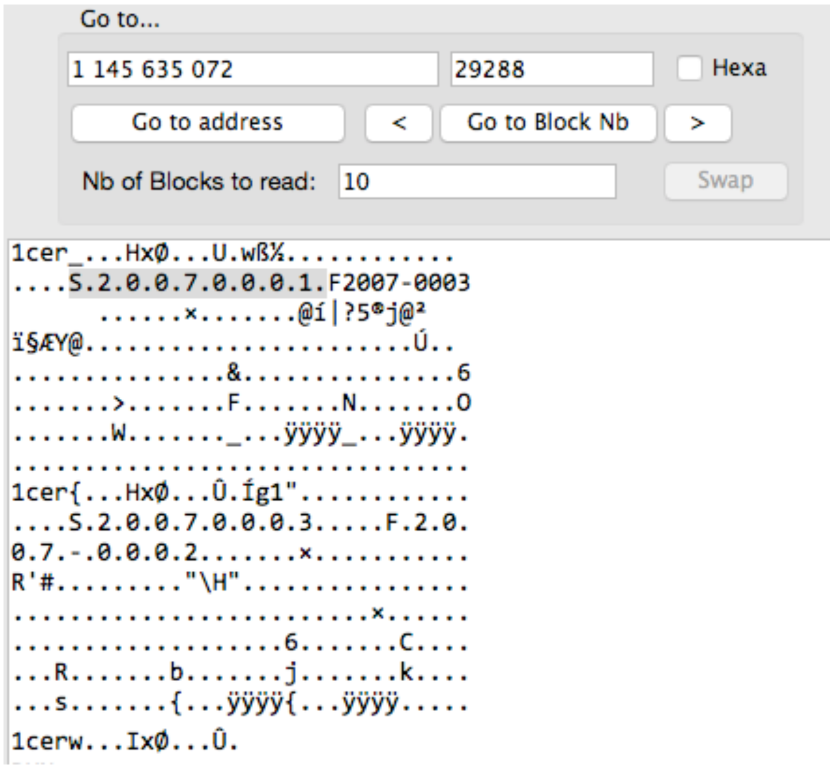
```
3163 6572 5F00 0000 4878 D802 0609 DB07 77DF BD97 0100 0000 0200 0000 0C00 0000
0900 0000 5300 3200 3000 3000 3700 3000 3000 3000 3100 4632 3030 372D 3030 3033
2020 2020 2020 0000 0000 0D08 D707 0000 0000 0004 9140 92ED 7C3F 35AE 6A40 B29D
EFA7 C659 9440 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0010 0BDA 0700
0000 000A 0000 0016 0000 000D 0000 0026 0000 0008 0000 002E 0000 0006 0000 0036
0000 0006 0000 003E 0000 0006 0000 0046 0000 0008 0000 004E 0000 0001 0000 004F
0000 0008 0000 0057 0000 0008 0000 005F 0000 00FF FFFF FF5F 0000 00FF FFFF FF00
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
3163 6572 7B00 0000 4878 D802 0609 DB07 CD67 3122 0200 0000 0200 0000 0C00 0000
0900 0000 5300 3200 3000 3000 3700 3000 3000 3000 3300 0A00 0000 4600 3200 3000
3000 3700 2D00 3000 3000 3000 3200 0000 0000 0708 D707 0400 0000 0104 0000 0011
5227 2303 0000 0001 0600 0000 225C 4822 0000 0400 0000 0107 0000 0015 1F1B 1400
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 000D 08D7 0700 0000 000A
0000 0016 0000 000A 0000 002E 0000 0008 0000 0036 0000 0007 0000 0043 0000 0007
0000 0052 0000 0007 0000 0062 0000 0008 0000 006A 0000 0001 0000 006B 0000 0008
0000 0073 0000 0008 0000 007B 0000 00FF FFFF FF7B 0000 00FF FFFF FF00 0000 0000
3163 6572 7700 0000 4978 D802 0609 DB07 8550 554E 0300 0000 0200 0000 0C00 0000
0900 0000 5300 3200 3000 3000 3700 3000 3000 3000 3100 0A00 0000 4600 3200 3000
```

If you know the structure of the blocks you are examining, you can recognize the data. For instance, in this screenshot, you can see that the highlighted part is `3163 6572`. In the Selection groupbox above, you can see that it can be read as '1cer'. By byte swapping it, you will read 'rec1' which is the tag, which begins a Record. At the line #9, and the line #17, you will find the same tag, meaning these are beginnings of the 2 next records.

After `3163 6572`, you will find `5F00 0000 (95)` which is the record length. Then `4878 D802 0609 DB07, which` is the Timestamp (8 bytes), followed by the Checksum `77DF BD97`. Then you will find `0100 0000` (Record number 1), `0200 0000` (Table number 2), `0C00 0000` (which has 12 not empty fields), etc.

When you highlight a part in this area, it will automatically adjust your selection to an exact number of bytes, and highlight the corresponding part in the ASCII area. And if you highlight a part in the ASCII area, it will automatically adjust your selection to an exact number of bytes, and highlight the corresponding part in the Hexadecimal area.

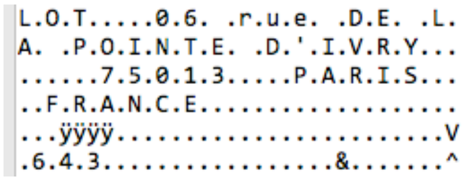
ASCII area makes it easier to read at least the text parts of the record.



```
1 145 635 072    29288
Nb of Blocks to read: 10
```

```
1cer_...HxØ...U.wß½............
....S.2.0.0.7.0.0.0.1.F2007-0003
      .......×........@í|?5®j@²
ï§ÆY@.......................Ú..
...............&...............6
.......>.......F.......N.......O
.......W......._...ÿÿÿÿ_...ÿÿÿÿ.
................................
1cer{...HxØ...Û.Íg1"............
....S.2.0.0.7.0.0.0.3.....F.2.0.
0.7.-.0.0.0.2........×..........
R'#.........."\H"...............
.........................×......
...................6.......C....
...R.......b.......j.......k....
...s.......{...ÿÿÿÿ{...ÿÿÿÿ.....
1cerw...IxØ...Û.
```

For instance, this part of a record (below) is easily readable. The dot between letters shows that this text is in UTF16 format (2 bytes per character)



```
L.O.T.....0.6. .r.u.e. .D.E. .L.
A. .P.O.I.N.T.E. .D.'.I.V.R.Y...
......7.5.0.1.3.....P.A.R.I.S...
..F.R.A.N.C.E...................
...ÿÿÿÿ........................V
.6.4.3.................&.......^
```

Conclusion

DataAnalyzer V1.0 is a preliminary version. The next version will be more complete in terms of statistics and information more directly usable for database understanding and optimization. There are 2 main targets: Debugging DB4D (4D & Wakanda data engine), and troubleshooting Developers applications.

This technical note explained how DataAnalyzer works. The next one will be more oriented toward what you, Developers, can get from data file analysis.