4D SUMMIT 2011 PRE-CLASS

Advanced 4D Administration and In-Depth Troubleshooting

Presented by

Thomas Maul, Managing Director, Northern Europe
Sergiy Temnikov, Team Leader, SQL Development Group
Josh Fletcher, Senior 4D Technical Services Team Member

ABSTRACT

Welcome to the 4D Summit 2011 Pre-Class! This documentation is divided into two sections to match two different parts of the pre-class:

- The “Best Practices” section is all about best practices for 4D administration and deployment.
- The “In Depth Problem Analysis” section is all about troubleshooting real-world problems using the powerful tools in 4D.

TABLE OF CONTENTS

Abstract 2
Table of Contents 2
BEST PRACTICES 4
Database settings 4
Compiler 4
Database 5
Web 5
SQL 7
Compatibility 8
Best Practices 8
Understanding Unicode Mode 9
What is the ICU? 9
What is Unicode? 9
How does it work in 4D? 9
Best Practices 10
Backup and Restore 11
4D Backup Scheduling 11
4D Restore 12
Log Management 15
Best Practices 21
Maintenance 22
Standard Mode versus Maintenance Mode 22
Access Rights 23
Verify 24
4D Language Support 25
Best Practices 27
Operating System 27
Windows 27
Mac OS X 28
Best Practices 28
Hardware 29
Memory 29
Solid-State Drives 39
Best Practices 57
4D and Desktop virtualization 57
RDPCLIP 57
Printer Driver 57
Client Load 58
Things to Avoid 58
Database Settings 58
IN DEPTH PROBLEM ANALYSIS 59
4D Cache Log Analyzer 59
Cache Log Recorder and Analyze 59
4D Cache Log Recorder 61
I. Introduction 61
II. Installation 61
III. Opening the interface 61
IV. Setting the options 62
V. Clearing the cache 63
VI. Starting the information recording 63
IX.Debug logs 65
X.Sending the generated data 65
XI. Uninstalling the component 65

4D Cache Log Analyzer 66
Statistics Overview 66
Used memory 67
LOG EVENT 70
Need Bytes 71
Flush from 73
Flush duration 74
Flush by task 75
Process list 76
Statistics Details 78
Used Cache Size 79
Free Memory 79
Used physical memory and Used virtual memory 79
Used Mem and Free Mem 80
Small and big blocks 80
Big blocks 80
Small blocks 80
Examples 81
Acknowledgment 89
4D Info Report Component 90
This information component is mainly designed to provide in readable text documents a description of: 93
Supported version of 4D: 93
Some information on this component: 93
Confidentiality 94
Supported language for the interface and the content of the reports): 94
Supported platform and OS: 94
Naming conventions: 94
List of the shared methods of the component (by category) 94
Creation of reports 94
Display of existing reports 94
Display of dialogs 94
Utility 94
Details on the shared methods of the component 95
Description of the main parameter for the creation of reports: 98
Discover the Component run in stand-Alone: 99
The Format and content of the generated reports: 105
1/ All generated files have a structured name: 105
2/ Each document is generated with a UTF-8 format, with all lines ending with CR+LF. 105
3/ The content of the report is allows the component to find searched items. 105
A/ The Header: (general description of the Computer): 105
B/ Description of the 4D Application: (version, build, 64-bit?): 105
C/ Description of the Database (size, location, plugins, components, settings): 106
D/ Infos on the Tables (num, name, nb records, fields, indexed fields, subtables, triggers): 108
Deployment of the component: 108
Accessing the code of the component from the Host database 109
Acknowledgment 110
4D POP DATA ANALYZER 111
Overview 111
Link informations 116
Usage 117
Acknowledgment 118
Debug log 118
4D NETWORK LOG ANALYZER 120
Recall about the network log 120
Overview 120
Statistic 122
Using network log analyzer - Creating and importing Logs 123
Advanced Usage – Combination with 4D Debug Log 123
Advanced Usage – Including Query path 125
Advanced Usage – Collecting debug logs from client computers 126
Network packet names 126
Backup log – Log Tools 127
Windows Resource Monitor 130

BEST PRACTICES

This portion of the 4D Summit 2011 Pre-Class is all about "best practices" for 4D. In particular the intended focus is on best practices for deployment and administration of 4D databases, as opposed to development, though there may be a few development-related tips sprinkled throughout.

DATABASE SETTINGS

4D has many settings that can be configured not only during development but for deployment and even at runtime throughout the life of the database. This chapter addresses the best-practices for many of these settings.

Note that this chapter does not cover every possible setting. The goal is to provide pertinent information for those settings that might have a "best practice" rather than document every setting. Probably the largest omission from this chapter is any discussion about timeouts. This is for a few reasons:

- Timeouts are a large topic, in general, and would require a significant portion of the pre-class to cover.
- There is already an excellent Technical Note covering all 4D timeouts in detail:
  - Mastering 4D Timeouts & Connectivity Settings
- The second part of the pre-class will cover network troubleshooting and, therefore, touch on the issue of timeouts.

Backup settings will be discussed in a separate section.

Reminder: this chapter is specifically about "Database Settings" in 4D v12. In previous versions of 4D these settings appeared in the "4D Preferences" dialog. In 4D v12 the settings that are application-specific are still accessible via the 4D Preferences dialog but the settings that are database-specific are accessed in Database Settings dialog:

Figure 1 - 4D Preferences

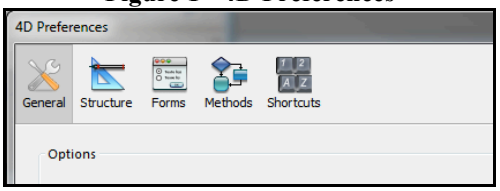
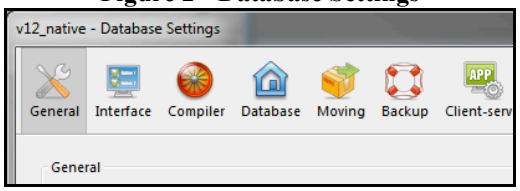
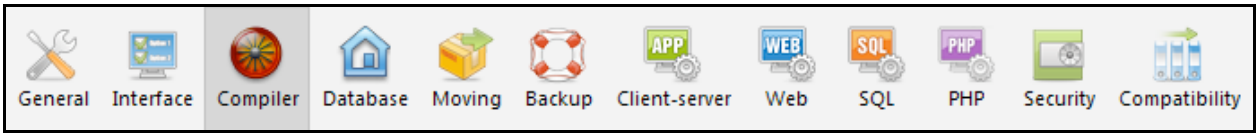


Figure 2 - Database Settings



The following sections are organized by "Theme". The name of each section corresponds to the name of the tab in the Database Settings dialog.

Compiler

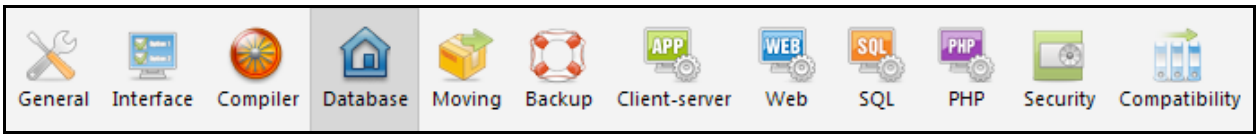


Compiler settings are, of course, typically a design-time consideration. However there are two settings that can be important from a deployment and maintenance perspective.

- Set the Compilation Path to "All Variables are Typed".
  - This setting tells the 4D compiler that all variables in the database have been explicitly typed and, therefore, to return errors for any variables where typing information is not found. This is especially useful for runtime because it avoids typing conflicts **before** they happen. If typing conflicts can be detected at compile time, the developer can eliminate them before they make it into production.
- Understand what "Also compile for 64-bit processors" means and set it accordingly.
  - The name of this setting is perhaps a little misleading. This setting is not required to run the database on a machine with a 64-bit processor. If this were the case the setting would certainly need to be on by default because it is difficult to find any modern computer without

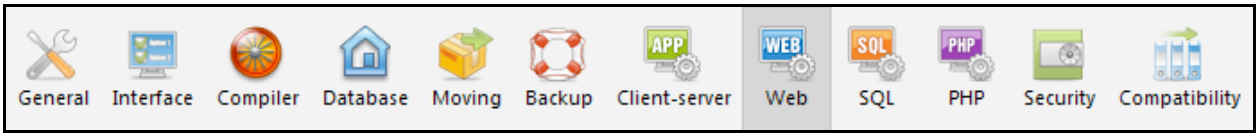
a 64-bit processor. This setting is required to run the database using **64-bit 4D Server**. Only enable this setting if the database will be deployed with 64-bit 4D Server. It is ok to leave it enabled in all cases but it unnecessarily increases the size of the database by adding 64-bit compiled code.

Database



- Understand what "Calculation of adaptive cache" is useful for.
  - A common misconception is that adaptive cache means the size of the cache can change at runtime; i.e. the cache is resized dynamically. This is false.
  - Another common assumption is that adaptive cache takes runtime resources into consideration when calculating the cache size. I.e. that the cache size might be based on the free memory in the machine or how many programs are running. This is false. Unless the hardware on the machine is modified (i.e. adding or removing RAM) the cache size **will not** change.
  - The purpose of adaptive cache calculation is very simple: this setting is useful for "shrink-wrap" products where the developer has no control over the system on which the database might be installed. In this situation it is prudent to try to set a cache size that will not dominate the machine memory. Since the developer has no direct control over the hardware used, the adaptive cache setting allows for a cache of varying sizes depending on the hardware.
  - If the database is deployed on a single machine, or on several machines where the developer has direct control over the cache size (whether by setting it directly, or via training and documentation), the adaptive cache feature offers **no advantage** and, furthermore, makes configuring the cache size unnecessarily complex. The best practice in this situation is to turn the adaptive cache off and simply set the value directly.
- Leave "Keep cache in physical memory" disabled as it could cause unknown side-effects. This setting is primarily useful for detecting memory leaks because it forces 4D to commit the entire cache allocation. It makes it easier to measure memory usage because with the setting disabled, the memory used will increase dynamically as items are added to the cache.
- The topic of setting cache size is discussed later.

Web



In general 4D is not used to serve only static Web content; that is to say 4D is not generally used purely as a Web server but, rather, a Web application platform involving dynamic requests and access to data. Because of this, performance is highly variable and depends too much on database design and client load to lend itself to a great deal of "best practices". But certainly there are some important points to consider.

Because Web application performance is so variable, one of the most important best practices for 4D Web applications is to load test them. There are many automated load-testing tools for Web applications. After choosing such a tool, the following methodology can be used:

- Decide a suitable target load.
  - This is probably the most important step. It is absolutely critical to understand the intended usage of the Web application and adjust the testing to match those expectations.
- Test that load.
  - Of course it is prudent to test at a load higher than anticipated to ensure fault tolerance.
- Tweak as needed.

- See the following sections for more details.

An alternative method is to test for the upper-limit:

- Decide on server hardware; this will determine the upper boundary of suitable performance.
- Load test the server to the maximum until it is no longer responsive.
- Tweak as needed to prevent the server reaching that point.

If load against the 4D application is a concern consider techniques such as proxying, load-balancing, rewriting, etc.:

- Proxying or rewriting can be used to eliminate the need for the 4D database to serve any static content at all, for example. This frees the database to handle only requests for dynamic Web content and, of course, means that 4D can be integrated into other Web application environments to act as the Web-based database server.
- Load-balancing should be considered in any case where a single instance of the 4D database is not enough to handle the load. There are two primary options:
  - Host multiple copies of the 4D application.
    - The primary disadvantage here is that the data would need to be synchronized between each copy so it adds extra complexity.
  - Use a single 4D Server application and handle the Web requests using 4D Client Web Server instances. This brings two advantages:
    - There is no need to synchronize the data as 4D automatically manages the transport of the data to/from client and server.
    - The DB4D requests from each client can be handled pre-emptively on the server.
- Partitioning should be considered if the data can be logically partitioned into separate databases, thus the load against each database is reduced.

4D Web Cache

The 4D Web Cache is used to cache static Web content with the following constraints:

- Only static pages, GIF pictures, JPEG pictures <100 KB, and style sheets are cached.
- When the cache is full and additional space is required, 4D "unloads" the oldest pages first, among the least demanded ones.

Use the 4D Web cache when appropriate:

- It is disabled by default so if it has been ignored in your Web applications, be sure to try it.
- Do not use it during development as clearing the cache between development changes can be tedious.
- Set the size based on the size of the static content, but bearing in mind the resources on the machine. Set it large enough to hold all of the static content but not so large that it overwhelms resources on the machine.
- Remember that the cache usage can be observed using the WEB CACHE STATISTICS command or the /4DSTATS URL.

Web Processes

One of the most important things to keep in mind about Web processes is, since they are really just 4D processes, they are cooperative. This has some important ramifications:

- If 4D Server is the Web server, there is no opportunity for the Web processes to scale with the number of CPUs/cores.
  - Consider client Web serving if this is a concern.

- As 4D processes, Web processes need to use the same techniques as other 4D processes in order to "play nice".
  - Use the IDLE command in order to yield control to other processes.
  - Use DELAY PROCESS when appropriate.
  - Avoid synchronous calls. Any 4D process is capable of locking down the entire application via synchronous calls. All plug-in calls, for example, are synchronous (though the plug-in can yield control to 4D). LAUNCH EXTERNAL PROCESS is synchronous as well, by default.

Here are some notes about Web Process settings:

- Inactive Process Timeout
  - Adjust this while keeping server resources in mind (refer especially to the memory section later). The goal is to strike a balance between process availability (creating processes takes time) and memory footprint (persistent processes consume memory).
- Maximum Concurrent Web Processes
  - This setting can be useful to prevent DOS attacks from overloading the server.
    - To be clear: you can't prevent DOS attacks with 4D; that is not the point here. The point is to make sure a DOS attack does not overwhelm 4D itself. The website may be down, but 4D can still be responsive to other users (clients, SQL access).
  - Set this to a "sane" value.
    - Not so low that typical usage results in HTTP 503 responses.
    - Not so high that it would be detrimental to the server.
    - Default is 100. Anticipate more than 100 concurrent requests? Set accordingly.

Text Conversion

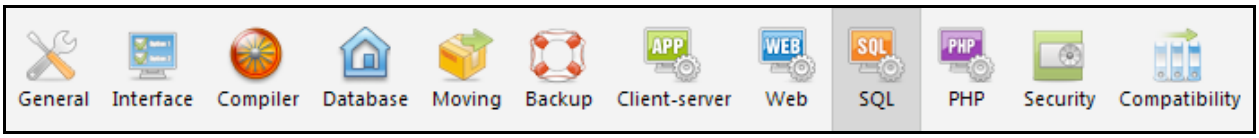
Set the "Standard Set" to UTF-8. This offers several advantages:

- It is important to use a Unicode encoding moving forward, especially if multi-language support is offered.
- Of the various Unicode encodings, UTF-8 is efficient for the Web because it is effectively a 1-byte character encoding for the most common characters; it is space efficient resulting in smaller data transfers.
- UTF-8 is the de facto standard on the Web and therefore enjoys the most support and compatibility.

Keep-Alive Connections

These settings are for controlling the TCP connections to the server, not Web request processes. As such the best practice will depend on IT infrastructure and is not necessarily interesting from a 4D perspective. Said another way, keep-alive connections are a TCP/IP concept, not a 4D concept or a Web concept. Look to references about TCP keep-alive connections for further information on this topic, not 4D documentation.

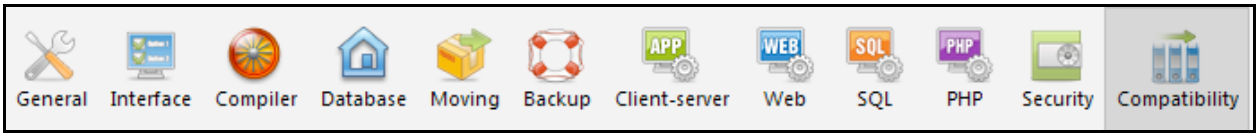
SQL



- SQL Server Publishing
  - There is a setting labeled "Launch SQL Server at Startup". This is slightly misleading. The 4D SQL server is **always** running. The port shown in this dialog will **always** be in use for 4D Server; the SQL server connections are a fundamental part to 4D Remote connections. What this setting is toggling is the ability to make **external** connections to the 4D database. To

connect via the 4D ODBC driver or "direct" 4D-to-4D SQL connections, this setting must be enabled. If these connections are not necessary, this setting should be disabled of course, but it will **not** turn the SQL server "off".
    - For security reasons, if this setting is enabled, the On SQL Authentication database method should be implemented as per the documentation.

Compatibility



Note: *The Compatibility tab will only appear if the database was converted from a version prior to 4D v11 SQL.*

There are some very important settings to be aware of for converted databases.

- There are several settings related to executing formulas on the server instead of the client. From a performance and optimization standpoint, these settings are critical. These settings affect the way in which 4D handles commands like QUERY BY FORMULA.
    - In compatibility mode, the formula is evaluated on the client. This means that **each** record is sent to the client for evaluation while the query is executed. This is extremely inefficient.
    - If the setting is enabled, the formula evaluates on the server side. Only the results are sent to the client.
    - The reason this setting is not enabled by default is that changing the execution context of the formula can have drastic consequences. What if the formula is a 4D method that relies on process variable data on the client?
- QUERY BY FORMULA was well known to be a slow and inefficient command in previous versions of 4D, especially when acting across table relations. The compatibility setting to allow QUERY BY FORMULA to use SQL joins can drastically improve the performance of cross-table queries.
- **Important Note:** the "contextual mode" option for 4D's Web server is completely removed in 4D v13. It is very important to stop using this feature.
- The "Unicode mode" setting is addressed in the next section as it is a large and important topic.

Best Practices

- Concerning timeouts, refer to Mastering 4D Timeouts & Connectivity Settings
- Set the Compilation Path to "All Variables are Typed".
- Only set "Also compile for 64-bit processors" if deploying 4D Server v12 64-bit.
- Only use "Adaptive Cache" when there is no control over the cache setting.
- Only use "Keep cache in physical memory" for troubleshooting.
- Load test Web applications.
- Use proxying, rewriting, load-balancing, and partitioning as needed.
- Use the 4D Web cache…
    - Except during development.
- Tune Web process settings to match the expected load.
- Use UTF-8 for Web content.
- Set keep-alive connections based on network topography.
- SQL Server Publishing does not disable or enable the SQL server, only external connections.
- Always implement On SQL Authentication if external connections are allowed.
- Adjust compatibility settings to execute BY FORMULA commands on the server for improved performance.
- Adjust compatibility settings to allow QUERY BY FORMULA to perform joins for improved performance.

- Stop using contextual mode.

UNDERSTANDING UNICODE MODE

4D v11 SQL introduced support for the Unicode standard for character encoding. Furthermore since 4D v11 SQL Release 2 4D uses the ICU collation algorithm for sorts, queries and data comparisons. This section addresses some important points about using Unicode in 4D.

Note: this content was adapted in part from Technical Note 11-22, *ICU in 4D: Impact on Queries, Sorts and String Comparisons*. This Technical Note is included with the pre-class materials.

What is the ICU?

The International Components for Unicode (ICU) is an open source project that provides C/C++ and Java libraries to support Unicode, various types of exchanges, and software globalization. The ICU gives applications the same behavior and results on all platforms and between C/C++ and Java software.

The ICU also lets you handle Unicode text, character properties, and character set conversions as well as Unicode analysis of regular expressions.

The ICU manages character, word, and line boundaries as well as case-sensitive comparisons, collation, and searches. The ICU also allows transformations concerning normalization, switching from upper to lowercase (and vice versa), as well as script transliterations. (Transliteration refers to the practice of systematically converting text from one script system to another).

In addition, the ICU supports complex text layout (e.g. Arabic, Hebrew, Thai, etc.), as well as formatting and parsing of dates, times, numbers, currencies and messages based on rules.

What is Unicode?

Unicode is a set of unified character codes that can manage practically any standard writing system in the world. Unicode is a coding system, just like Morse or its forerunner ASCII; it applies a code (i.e. a set of symbols, here in the form U+XXXX) to a set of data (in this case characters). Unifying the encoding allows different countries (each using its own coding or its own coding version, according to the characters required) to exchange data.

Whereas in US-ASCII numeric values are normally included between 1 and 127, in the case of Unicode the upper limit exceeds 65,000.

There are several ways to code Unicode numeric values; here are some examples:

- UTF-16 uses 16-bit integers.
- UTF-32 uses 32-bit integers.
- UTF-8 uses 8-bit integers. UTF-8 offers a kind of lossless compression (see below).

In UTF-16 a character takes up two bytes of space vs. one byte in ASCII (American Standard Code for Information Interchange). The first 256 characters in UTF-16 correspond pretty closely to the table defined by ASCII, whereas the coding uses a system of zones (in this case, a succession of ranges, planes and blocks), in all extending to over one million potential characters.

Unicode exists in several forms to suit different purposes. Currently, the most widely used is UTF-8, which is a variable-width encoding. A character is represented by a group of up to 4 bytes depending on the desired Unicode character. This adaptability means that it can be used "as is" for systems used to ASCII, while those that need more complex coding can call the Unicode character at will.

How does it work in 4D?

In versions prior to 4D v11 SQL, 4D contained “character sets” for each supported language. A character set is a table that maps the characters (a-z, 0-9, ç, è, à, and so on) to their numeric values (1,2,3, and so on), but limited to a 1-byte range. Thus each character set had overlapping values with all other character sets.

On startup 4D selected the character set corresponding to the system language. Since each character set had overlapping values, it made it impossible for 4D to manage several languages simultaneously.

Starting with 4D version 11 all text data is stored as UTF-16. Similarly 4D uses UTF-16 strings for language objects, with the exception of defaulting to UTF-8 for Web-related needs. UTF-8 has better compactness and readability for standard characters (a-z, 0-9).

However, converted databases are not in "Unicode mode" by default. There is a check box in the Database Settings to ensure compatibility (new 4D databases do not have this option).

The behavior of 4D in "Compatibility mode" is the same as in previous versions of 4D. That is, all text data is treated as if 4D is still using the MacRoman ASCII character set. However be aware that in compatibility mode all text data is being converted on-the-fly from UTF-16 to MacRoman ASCII. This can have an important impact on performance.

Note that Unicode values from 1 to 127 correspond to the same characters as for ASCII. This is practical since these are the most commonly used characters (A-Z, 0-9, etc.)

Since 4D v11 SQL Release 2 (11.2) 4D uses the ICU collation algorithm when the database runs in Unicode mode. This is used for sorts, queries, and data comparisons. This algorithm compares 2 character strings in an intelligent manner. This is transparent to the user and developer. However there are a number of rules inherent in the ICU (by virtue of Unicode) that can drastically alter the behavior of 4D commands. The handling of characters defined as "ignorable" is one example:

```
C_TEXT($textToFind_t;$textToSearch_t)
C_LONGINT($pos_l)

$textToFind_t:=Char(1)

$textToSearch_t:="a"+Char(1)+"b"

$pos_l:=Position($textToFind_t;$textToSearch_t)
```

When running in Unicode mode the value of $pos_l is 1. Char(1) is defined as an ignorable character in Unicode. It must not be taken into account when completing string comparisons. Because Char(1) was ignored, the search pattern in $textToFind_t becomes the empty string, which is thus found right at position 1.

Certain 4D commands (Length, Replace string, Position, Delete string, Substring, Change string) have thus been modified to support the optional * parameter to force searches to be based on character codes instead of using the ICU rules.

Best Practices

There are two important facts to understand about Unicode mode in 4D:

- All databases should switch to Unicode mode moving forward.
- It is important to be educated about what Unicode support means.

Unicode is a relatively complex topic and is also largely development related so, in fact, it is not covered in-depth here. There is a Technical Note included with the Summit materials that covers the development ramifications of switching to Unicode mode. This should be considered required reading for any 4D developer and an understanding of Unicode can be useful for administrators as well for troubleshooting and diagnosis of text-related issues.

Here are some other tips that may be useful:

- Use UTF-8 for Web data. It is generally more compact (especially for English), has better compatibility, and is the de facto standard on the Web.

- XLIFF files that are accessed by 4D must be UTF-8 encoded. If they are encoded as ANSI or MacOSRoman they will not be correctly processed by 4D. Take care when editing these files to make sure the editor does not alter the encoding.
- Be sure to review and understand how the Unicode defines keyword boundaries if keyword searches are used:
  - http://www.unicode.org/unicode/reports/tr29/#Word_Boundaries
  - As with the rest of the ICU support in 4D v11 SQL, this was added in 11.2. Keyword indexes from prior releases will need to be rebuilt.

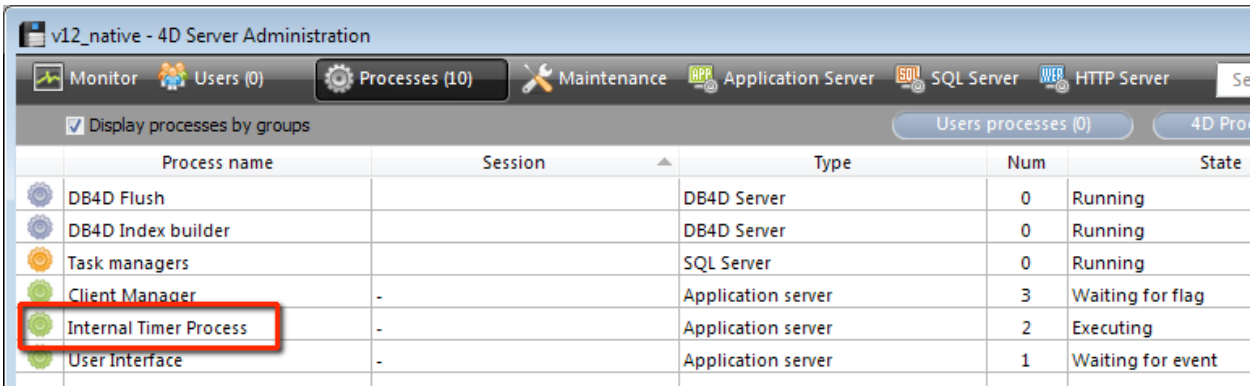
BACKUP AND RESTORE

This chapter offers some insight and tips for maintaining optimal database integrity.

4D Backup Scheduling

This section is not about how to use the backup scheduler – the scheduler is intuitive – but, rather, aims to offer some insight about how to troubleshoot any incidence of a periodic backup not completing.

Periodic backups are managed by a 4D kernel process called the "Internal Timer Process". The Internal Timer Process periodically checks the backup settings to know if it is time to launch a backup:

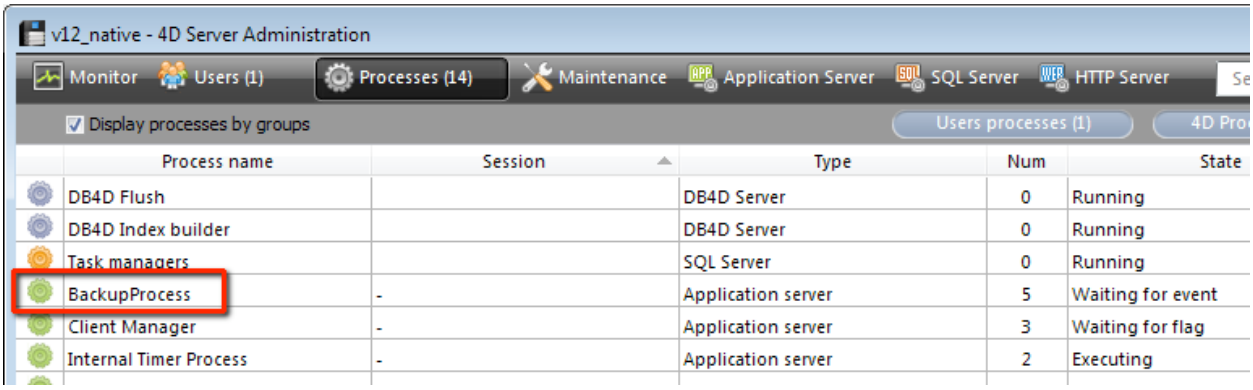


Note that the Internal Timer Process has other tasks. For example it collects information about processes displayed in the 4D Server Administration window. If this process is stuck somehow, no periodic backup can occur. One easy way to tell if the Internal Timer Process is stuck is if the Administration dialog is not refreshing.

Normally the Internal Timer Process's State value will always be "Executing" on 4D Server and either "Executing" or "Waiting for Event" in 4D Local Mode. If it is stuck, you can get its state from the Runtime Explorer dialog because the Runtime Explorer collects process state from the "User interface" process instead.

Because the "Internal Timer Process" has many other tasks besides running periodic backups, it may be something totally unrelated to backup that blocks the process. Other duties for this process include Monitor data collection, client-server progress indicator updates, resource folder change notifications for clients, checking for structure modifications, etc. If any of these tasks encounter and unexpected failure that causes the Internal Timer Process to become stuck, no periodic backups can occur.

Assuming the Internal Timer Process is running correctly, once it detects that a backup should execute, a new process named "BackupProcess" is launched:



It is therefore important to monitor the status of "BackupProcess" if a problem occurs as well:

- Does it exist?
- Is it running?
- What is the State?

Monitoring the status of the Internal Timer Process and BackupProcess can be automated, of course, using the PROCESS PROPERTIES command. It can be useful to have a stored procedure, for example, to monitor the state of these processes if periodic backup problems are suspected.

The "On Backup Startup" Database method is called before backup and the "On Backup Shutdown" Database method is called after backup. By logging calls to these methods, one can tell if the application is stuck in the middle of a backup operation or outside the backup operation. This is important information to know, to be able to tell where the backup process is failing.

The "On Backup Shutdown" Database method receives a status code in $1. It is important to log this status to know if a particular backup failed for some reason. Additionally, based on the backup configuration settings of course, 4D should be attempting to backup again even if the backup failed. Logging all of this information can be very important.

It is also good practice to add some code to the "On Backup Shutdown" Database method to notify the database administrator by email, for example, whenever $1 indicates the backup failed.

4D Restore

When 4D creates a database backup, it assigns a numeric suffix to it. If the data log file is enabled, a backup is also created for that file. The suffix for the log file backup is always one less than the database backup. At first this may seem confusing and, even worse, when it's time to restore a failed database this may add confusion.

There are two ways to address this issue:

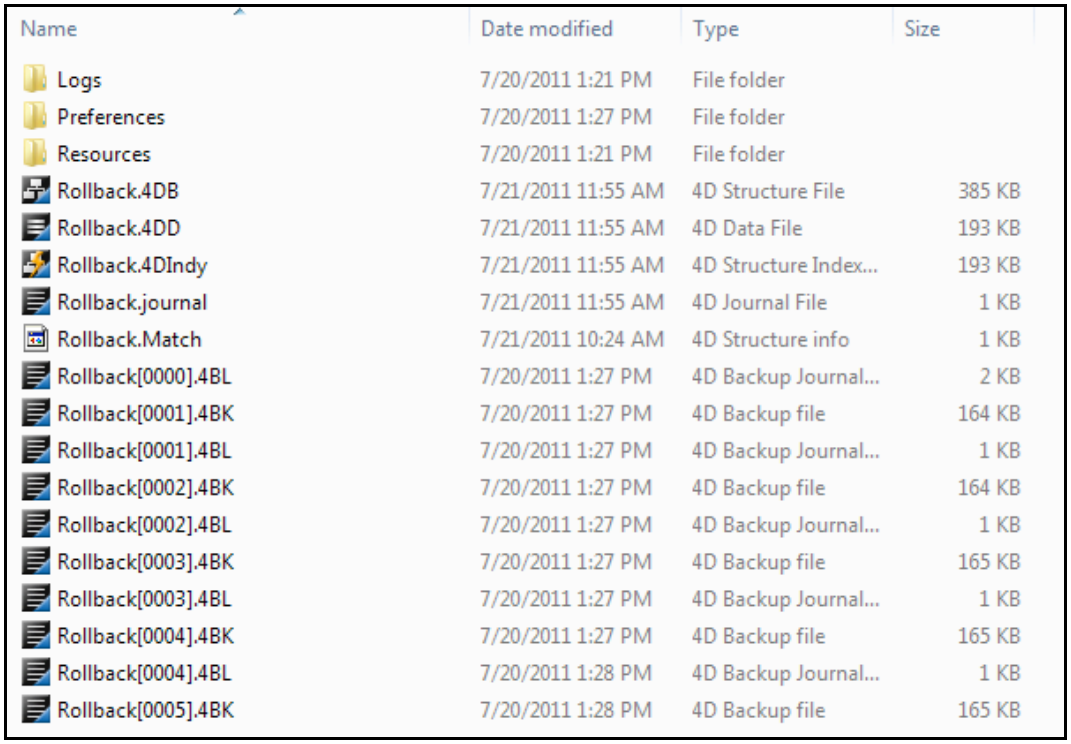
- Ignore it and follow a simple rule.
- Understand the design (but you can still use the simple rule).

How to Restore the Simple Way

Regardless of whether or not the backup file numbering is understood, there is one simple rule to follow when restoring a database:

Always choose files that have matching numbers.

For example, given this file structure:



Name	Date modified	Type	Size
Logs	7/20/2011 1:21 PM	File folder	
Preferences	7/20/2011 1:27 PM	File folder	
Resources	7/20/2011 1:21 PM	File folder	
Rollback.4DB	7/21/2011 11:55 AM	4D Structure File	385 KB
Rollback.4DD	7/21/2011 11:55 AM	4D Data File	193 KB
Rollback.4DIndy	7/21/2011 11:55 AM	4D Structure Index...	193 KB
Rollback.journal	7/21/2011 11:55 AM	4D Journal File	1 KB
Rollback.Match	7/21/2011 10:24 AM	4D Structure info	1 KB
Rollback[0000].4BL	7/20/2011 1:27 PM	4D Backup Journal...	2 KB
Rollback[0001].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0001].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0002].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0002].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0003].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0003].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0004].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0004].4BL	7/20/2011 1:28 PM	4D Backup Journal...	1 KB
Rollback[0005].4BK	7/20/2011 1:28 PM	4D Backup file	165 KB

There are 5 total backups. If backup #3 is to be restored, select the following files:

- Rollback[0003].4BK
- Rollback[0003].4BL

Because the files match, 4D restore will automatically restore the files to the same folder and the database will be ready to use.

This rule requires no further explanation and so can be useful for administrator training.

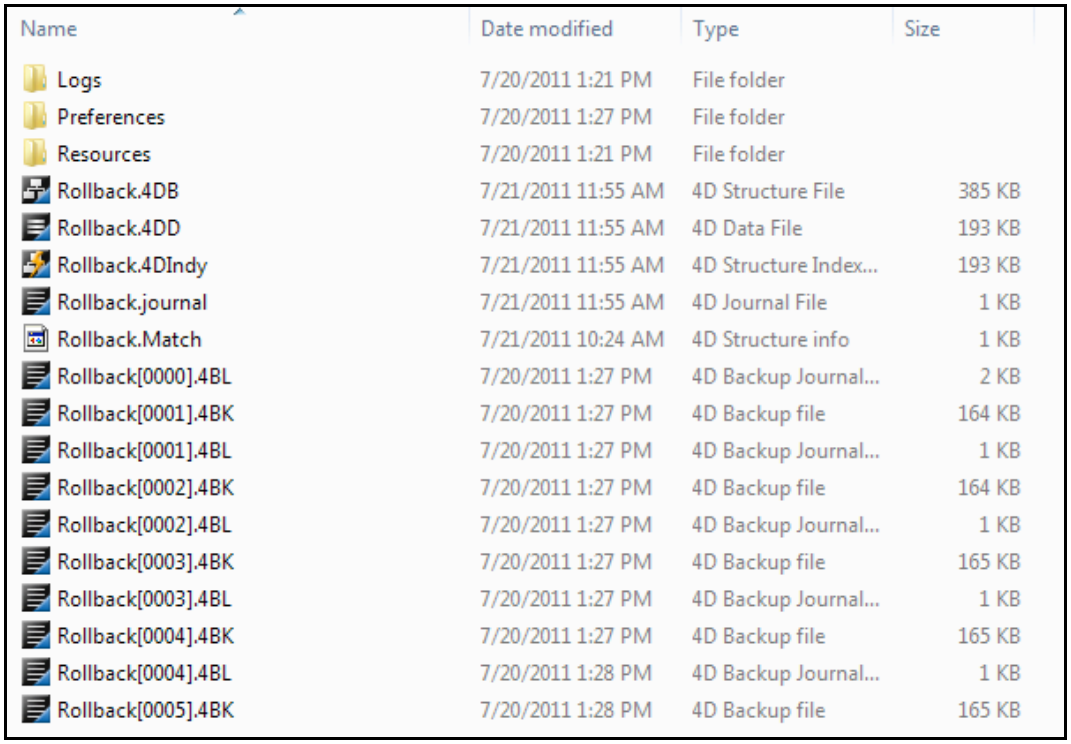
Understanding Backup File Numbering

As mentioned previously, 4D's backup process always assigns a number to the backed up log file that is less than the current database backup. This is completely deliberate. The database being backed up **already contains** all of the information in its current log file. If this backup is restored, there is no reason to restore the then current log file. The database already contains those changes.

There are two scenarios for a restore:

- This is the most recent backup. When the database is launched, select the current log file of the broken database (assuming it is also not damaged). If the current log file cannot be selected, there is no choice but to create a new log file.
- This is not the most recent backup. Use the simple rule mentioned above; select the matching 4BL file based on the number.
    - If there are subsequent log files to be integrated, select those in numeric order until reaching the current log file.

Here is slightly more complex example. Given this file structure:



Name	Date modified	Type	Size
Logs	7/20/2011 1:21 PM	File folder	
Preferences	7/20/2011 1:27 PM	File folder	
Resources	7/20/2011 1:21 PM	File folder	
Rollback.4DB	7/21/2011 11:55 AM	4D Structure File	385 KB
Rollback.4DD	7/21/2011 11:55 AM	4D Data File	193 KB
Rollback.4DIndy	7/21/2011 11:55 AM	4D Structure Index...	193 KB
Rollback.journal	7/21/2011 11:55 AM	4D Journal File	1 KB
Rollback.Match	7/21/2011 10:24 AM	4D Structure info	1 KB
Rollback[0000].4BL	7/20/2011 1:27 PM	4D Backup Journal...	2 KB
Rollback[0001].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0001].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0002].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0002].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0003].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0003].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0004].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0004].4BL	7/20/2011 1:28 PM	4D Backup Journal...	1 KB
Rollback[0005].4BK	7/20/2011 1:28 PM	4D Backup file	165 KB

Assume the live database is damaged and backup #1 needs to be restored and rolled forward. Perform the following steps:

- Launch 4D.
- Open a database; this is necessary to be able to access the full MSC Restore interface. Creating a new/blank database is completely sufficient.
- Open the MSC.
- Select the Restore tab.
- Click the Browse button and select “Rollback[0001].4BK”.
- Check the “Integrate one or more log file(s) after restore” check box.
- Click the Restore button.
  - 4D restores Rollback[0001].4BK to a folder named “Rollback[0001]”.
- 4D will prompt you to integrate log files until you click cancel.
  - Integrate Rollback[0001].4BL; 4D places the log file in a folder named “Rollback[0001].integrate”.
  - Integrate Rollback[0002].4BL; 4D places the log file in a folder named “Rollback[0002].integrate”.
  - Integrate Rollback[0003].4BL; 4D places the log file in a folder named “Rollback[0003].integrate”.
  - Integrate Rollback[0004].4BL; 4D places the log file in a folder named “Rollback[0004].integrate”.
  - Click Cancel to stop integrating log files.
- Move or copy the database in the “Rollback[0001]” folder into the live database folder, replacing the damaged database.
- Launch the database in 4D.
- If prompted, select the current live log file (“Rollback.journal”); 4D integrates the remaining changes and the database is now ready to go.

One sometimes overlooked benefit to 4D’s Restore feature is that it will automatically place the matching files into the same folder after the restore. If only a single restore is needed, a simpler procedure can be used. For example, to restore only backup #3 from above:

- Launch 4D.
- From the File menu, select Restore.
- Browser for “Rollback[0003].4BK” and restore it.
- Repeat these steps for “Rollback[0003].4BL”.
- Both the database and log file will be extracted to the same folder, by default. The database is now ready to use.
- When the database is launched in 4D, the changes in log file 0003 will be integrated; thus database 0003 is “caught up” to database 0004.

Keep in mind that the usability of a restored backup is directly related to what was backed up. For example if the database uses plug-ins, and those plug-ins are not included in the backup, then they will need to be installed separately after the restore. Alternatively copy the restored files into the live database folder. Be sure to make a copy of the live database before doing this just in case any mistakes are made in the restore process.

Log Management

The 4D Log Management feature is a critical part of any plan to maintain database integrity for non-read-only databases. The principal of the log file feature is very simple:

4D uses a memory cache to speed up runtime operations so that it does not need to constantly or repeatedly access the disk for database content. When database objects are changed (record CRUD, index updates, structure changes, etc.) the changes are made in the cache, not on the disk. Of course these changes need to be periodically flushed to the disk to commit them permanently. If the database fails in some way (e.g. a crash) and the cache had pending changes that were not flushed, those changes are lost.

Note: *the extension for the log file was changed from “.4dl” to “.journal” in 4D v11 SQL.*

The log file is quite different from the cache. Every data operation (record CRUD) is logged in the log file immediately on disk. The log file is managed in a different way from the other database files so this immediate disk access is efficient. Thus if the database crashes without flushing all changes from the cache, those changes can be integrated from the log file the next time the database is launched.

The log file further offers the ability to restore from a backup and integrate all changes that have occurred to the data since that backup. As in the example in the previous section, log files can be repeatedly integrated into a restored backup until that backup is “caught up” to the live system.

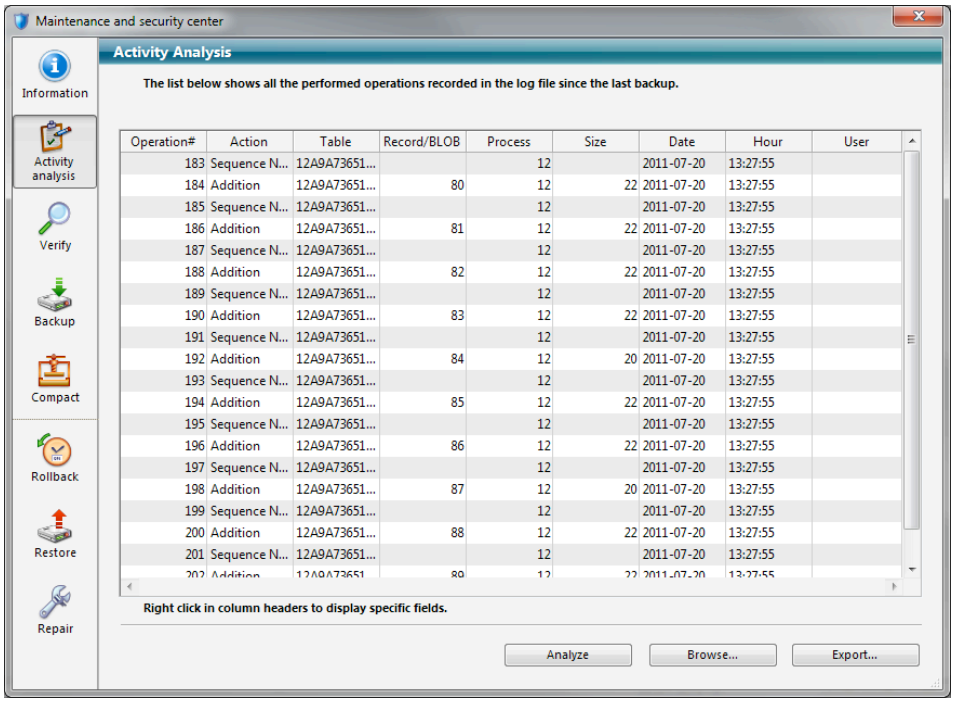
Finally this powerful feature also makes 4D’s Logical Mirroring feature possible.

Recovery with Log Files

The previous example of restoring backup #1 of the Rollback database already illustrated the power of the 4D log file. There are some other powerful abilities linked to the log file feature as well.

Activity Analysis

The MSC contains a page called “Activity Analysis”. This section of the MSC allows the user to view the operations contained in a selected log file (it defaults to the current log file, but other files may be selected). I.e. this UI allows the user to view the metadata contained in the log file:



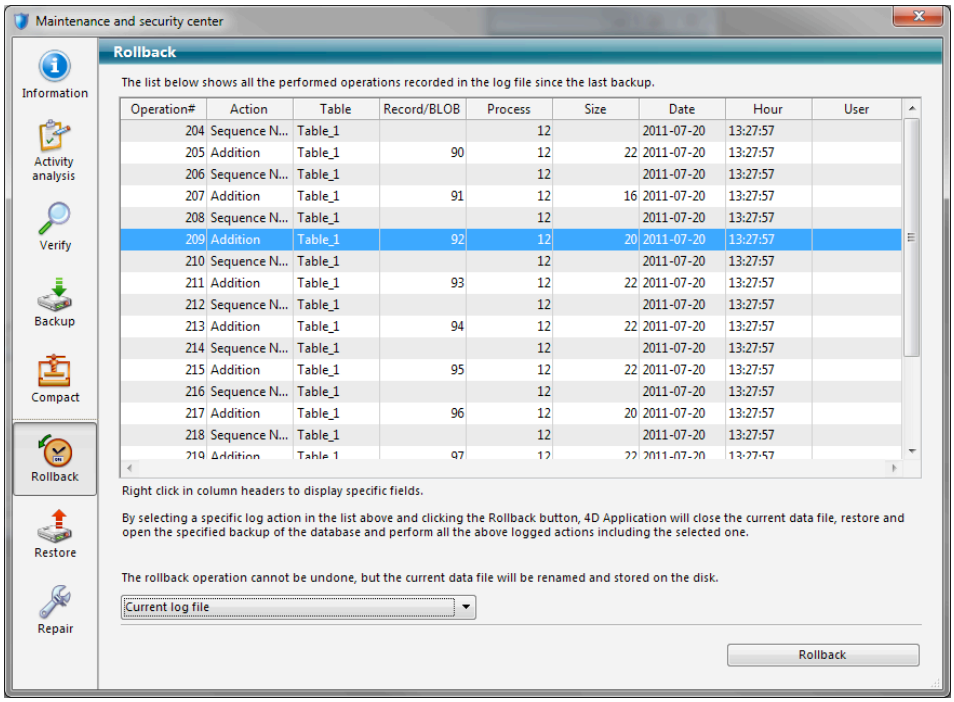
The following fields can be added or removed from the view:

Perhaps more importantly, the contents of this UI can be exported to a text file for analysis using other tools as well (e.g. for reporting). This will be explored further part 3 of the pre-class.

Rollback

Another powerful feature linked to the log file is the ability to "roll back" changes to the database.

Imagine that a user, or perhaps some automated task, has made some undesirable changes to the database. The Rollback feature allows the administrator to select the last valid operation in the log file and ignore all the subsequent invalid operations. This is done by accessing the Rollback tab in the MSC, selecting the last valid operation, and clicking the "Rollback" button:



In the above example all operations after the creation of record 92 in Table_1 will be ignored.

Note that a rollback operation does not actually "roll back" the changes. Instead what 4D does is restore the most recent backup and then integrate the operations in the log file up to the point selected in the Rollback tab, ignoring the remaining items.

Very Important: the most recent backup is **required** in order to use the rollback feature! This is one reason that enabling the log file feature requires a database backup.

Rolling back to changes in a previous backup (i.e. not the most recent backup) can make this process somewhat complicated; this situation is explained in the following section.

Rollback to Backup

In order for the Rollback feature to work, the most recent backup must be locatable by the MSC. The mechanism for locating the most recent backup involves checking the Backup project file, called "backup.xml" located in the "Preferences" folder. This can be a problem if it is necessary to roll back to changes that are *not* in the current log file.

Imagine a scenario where an invalid change was made, but was not noticed until more than one backup was completed. I.e. in order to roll this change back, it is necessary to restore a backup that is not the most recent backup. In a live database this is not possible because the Backup project file will not refer to any backup other than the most recent one. In this case it is necessary to recreate the state of the database at the time the change was made. Here is an example, using the Rollback.4dbase database shown previously:

Name	Date modified	Type	Size
Logs	7/20/2011 1:21 PM	File folder	
Preferences	7/20/2011 1:27 PM	File folder	
Resources	7/20/2011 1:21 PM	File folder	
Rollback.4DB	7/21/2011 11:55 AM	4D Structure File	385 KB
Rollback.4DD	7/21/2011 11:55 AM	4D Data File	193 KB
Rollback.4DIndy	7/21/2011 11:55 AM	4D Structure Index...	193 KB
Rollback.journal	7/21/2011 11:55 AM	4D Journal File	1 KB
Rollback.Match	7/21/2011 10:24 AM	4D Structure info	1 KB
Rollback[0000].4BL	7/20/2011 1:27 PM	4D Backup Journal...	2 KB
Rollback[0001].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0001].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0002].4BK	7/20/2011 1:27 PM	4D Backup file	164 KB
Rollback[0002].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0003].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0003].4BL	7/20/2011 1:27 PM	4D Backup Journal...	1 KB
Rollback[0004].4BK	7/20/2011 1:27 PM	4D Backup file	165 KB
Rollback[0004].4BL	7/20/2011 1:28 PM	4D Backup Journal...	1 KB
Rollback[0005].4BK	7/20/2011 1:28 PM	4D Backup file	165 KB

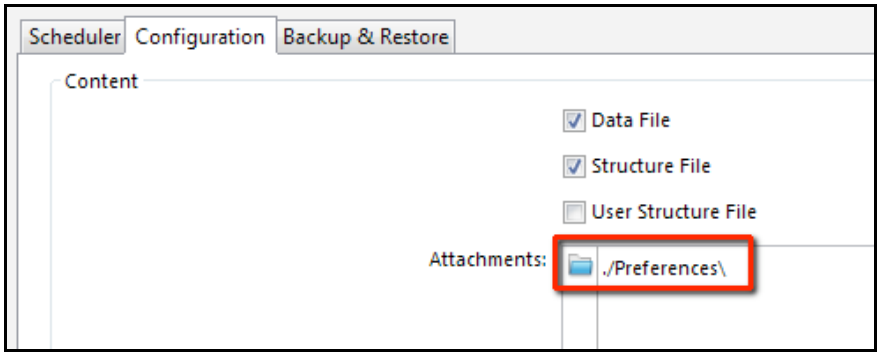
Note: *this example attempts to minimize the risk to the live database by carrying out the operations in a separate folder. Because of this the Backup project file must be edited to fix the paths pointing to the most recent backup.*

This database has 5 backups ([0001] through [0005]). The database is located at:

```
C:\Share\4D\Projects\Misc\Rollback.4dbase
```

A problem is identified in backup 0004. Here is how to fix it:

Important Note: the Backup configuration includes the backup.xml file (actually the whole Preferences folder). This makes the restore and rollback much easier:



- Create a new folder for the whole process (e.g. C:\Share\4D\Projects\Misc\Test).
- Copy backup 0004 and log file 0004 to this folder.
  - Restore backup 0004 to C:\Share\4D\Projects\Misc\Test\Rollback[0004]
  - Restore log file 0004 to C:\Share\4D\Projects\Misc\Test\Rollback[0004]
- At this point database version 0004 is in place; this database contains the transaction I want to roll back to. But the most recent backup file **relative to backup 0004** is still needed.
  - Copy backup file 0003 to C:\Share\4D\Projects\Misc\Test\Rollback[0004]\Rollback[0003].4BK

- Now the most recent backup file is in place. But this is not sufficient. 4D needs to know where to find this file. Because the Preferences folder was included in the backup, the correct backup.xml file has already been restored. However since I've restored backup 0004 to a different folder than the live database, the paths to backup 0003 need to be updated.
  - Open the restored backup.xml file at C:\Share\4D\Projects\Misc\Test\Rollback[0004]\Preferences\Backup\ and replace all instances of "C:\Share\4D\Projects\Misc\Rollback.4dbase" with "C:\Share\4D\Projects\Misc\Test\Rollback[0004]"; i.e. the paths must be updated because I have restored to the non-default location.
- Launch C:\Share\4D\Projects\Misc\Test\Rollback[0004]\Rollback.4DB with 4D.
- Open the MSC.
- Open the Rollback tab.
- Select the offending operation.

Result: the Rollback button is enabled, rollback can now be performed.

Summary: in order for the Rollback feature to work, the previous backup must be present and the backup.xml file must have the correct values (including backup number and correct paths). When rolling back to a version earlier than the most recent backup, it may be necessary to edit the backup.xml file to make sure 4D knows where the most recent backup is.

Tip: including the backup.xml file in the database backup means the other values in the backup.xml file (like the most recent backup number) will be correct when the database is restored and therefore not need to be edited. In this example after restoring the backup.xml file only the paths needed to be edited.

It is also possible to create a proper backup.xml file from scratch to make the rollback possible. After all it is only an XML file. But this adds unnecessary complexity to the task; adding the backup.xml file to the backup configuration is much easier.

Of course it is possible to restore the files to the default location of the live database. In that case the backup.xml file need not be edited but there is some risk to the live database if something goes wrong. Be sure to make a copy just in case.

LogTools

4D log files are, in fact, a kind of database. When a record is modified, for example, a copy of the entire record is stored in the log file. For situations of extreme database corruption, it is even possible to extract this data from the log file in order to import it into a new database.

There is a highly recommended third-party tool that can accomplish this called LogTools from Business Brothers (http://bbsp.com/). Log Tools will be covered in part 3 of the pre-class.

Mirroring

4D v11 SQL introduced a feature called "logical mirroring" that is based around two new commands:

- New log file – this command closes the current log file and creates a new one.
- INTEGRATE LOG FILE – this command is used to integrate a log file.

The mechanism is quite simple: the live database calls New log file to create a new log file. The previous log file is sent to the mirror database, and the mirror database integrates it. These log

files are referred to as being “segmented” or as “log segments”. In this way the mirror database has logical parity with the live database and is thus a perfect copy of the data.

Management of the log segments is a task left to the developer so the logical mirroring implementation can be tailored to match any infrastructure. If you are not familiar with this feature, please refer to the 4D documentation.

The purpose of this section is not so much to explain the mirroring feature but, rather, to point out why mirroring can be considered a best practice for any database and what some of the advantages are to using this feature.

Note: *4D’s logical mirroring offers unlimited redundancy. There is no reason why more than one mirror cannot be implemented. Simply ship the log segments to all mirrors in order to maintain additional copies.*

In addition to maintaining a live backup of the database, mirroring offers several other advantages that will be covered in the next sections.

Hotswap

The most obvious benefit to 4D’s logical mirroring is the ability to “hotswap” the application. If the primary server goes down the mirror server may be put in its place (or traffic can be redirected to the mirror) in order to minimize downtime while the problem is solved.

Offline Maintenance

Mirroring provides the opportunity to isolate the production database from maintenance tasks.

- Backup
  - In fact with the logical mirroring feature it is not possible to use 4D Backup on the live database without breaking the mirror. Instead backup operations are done on the mirror. This eliminates any drop in production database responsiveness due to the backup. Though 4D Backup allows read-only and transaction-based data access to continue, removing the backup task from the live server can considerably increase performance as the duration of a backup can be quite significant (on the order hours for larger databases).
  - Keep in mind that while the mirror is backing up it is not integrating log segments so it will not be fully up-to-date until the backup is complete and the mirror catches up integrating the logs.
  - Log file integration is typically much faster than database usage so it is generally not a concern that the mirror server might “fall behind” the live server. Still, for databases with **extreme** load this is something to consider.
- Verify/Repair
  - Similar to 4D Backup, MSC tasks can be executed on the mirror. A Repair operation, for example, cannot be completed while the database is running. A repair on the production server will require the database to be offline for the duration of the repair.
  - The Verify operation (there several forms of Verify; more on this in the next chapter) can be performed on a live production server but it will impact database performance.
  - Moving these tasks to the mirror ensures there is no performance impact on the live server at all.
  - Because the databases are logically identical, the risk of missing a problem on the live server is greatly reduced. In other words if a problem is detected on the mirror server, the problem probably also exists on the live server.

Furthermore since maintenance tasks do not impact the live server, they can be executed more often.

Offline Compact

Note: *this is technically part of the "offline maintenance" opportunities described in the previous section but, as this technique includes a bit more explanation, it has been moved to a separate section.*

Database compacting can be a time-consuming process. The larger the database, and the more fragmented it is, the longer it takes to compact. It's even possible the compact duration could be too long to fit the maintenance window of a production system.

"Offline Compacting" is a process by which a copy of the database is compacted while the production database is still live and running. When the maintenance is complete the only downtime is the time it takes to copy the compacted data file into production and complete the logical mirroring procedure.

4D's logical mirroring feature works well for this. For example, given two systems: Production and Mirror

- Shut down the Mirror database.
- Compact the Mirror database.
- Launch the Mirror database and integrate all log segments that were created on Production while the compact was underway.
- When the Mirror is "caught up", shut down Production.
- Integrate the current log file from Production into the Mirror.
  - At this point the compacted Mirror database is fully up to date.
- Copy the compacted Mirror database to Production and then complete setup of a logical mirror as before.

In this way the Production system is only down for the duration of a file copy and mirror setup, instead of the full compact.

Note that this may not be beneficial in all cases. The logical mirror setup process invokes a 4D Backup. If the backup operation takes longer than a compact, there's no reason to use this procedure as the Production server will be down for the duration of the backup. Just do the compact on Production instead.

Best Practices

- Implement the On Backup Startup and On Backup Shutdown database methods. Make note (e.g. via logging) of each call to each method so that when a problem occurs with the backup there is a trail to help determine when the problem is happening.
- When restoring a database and log file backup, make sure the backup numbers match.
- Use 4D's Log Management feature. There are many advantages to this feature beyond simple crash recovery.
- Log file metadata can be exported via the MSC for external analysis and reporting.
- Integrate the index files in the backup to avoid rebuilding the index after a restore.
- Integrate the backup.xml file into the backup configuration; it makes the rollback feature easier to use.
- Backup to a separate drive, but not an external drive. If you need to move the backup, to a network location for example, use the OS to move it. Do not force 4D to access the external storage. Launch a script from On Backup Shutdown, for example.
- Use 4D Backup; it is the best tool for backing up 4D databases. Furthermore it offers access Log Management feature.
  - Do not use Retrospect with 4D as we have known cases of corruption.

- Do not use Time Machine to backup 4D, it cannot backup open files.

- On the other hand use 4D Backup to back up 4D databases, not as a general file backup system. 4D Backup is not the appropriate tool to backup up hundreds of thousands of files, for example.

MAINTENANCE

This chapter is about best practices for maintenance of 4D databases. In particular the following sections explore the 4D Server Administration dialog and Maintenance and Security Center (MSC).

Standard Mode versus Maintenance Mode

It is important to understand that the MSC actually has two modes of operation:

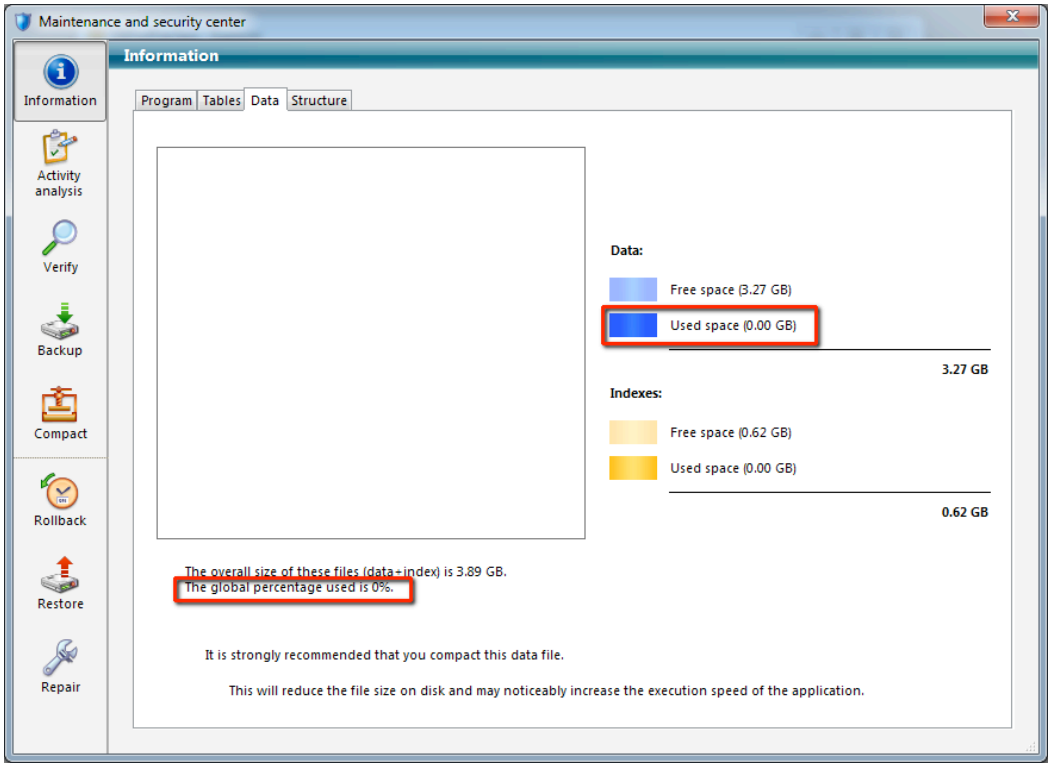
- Standard Mode
- Maintenance Mode

Standard Mode

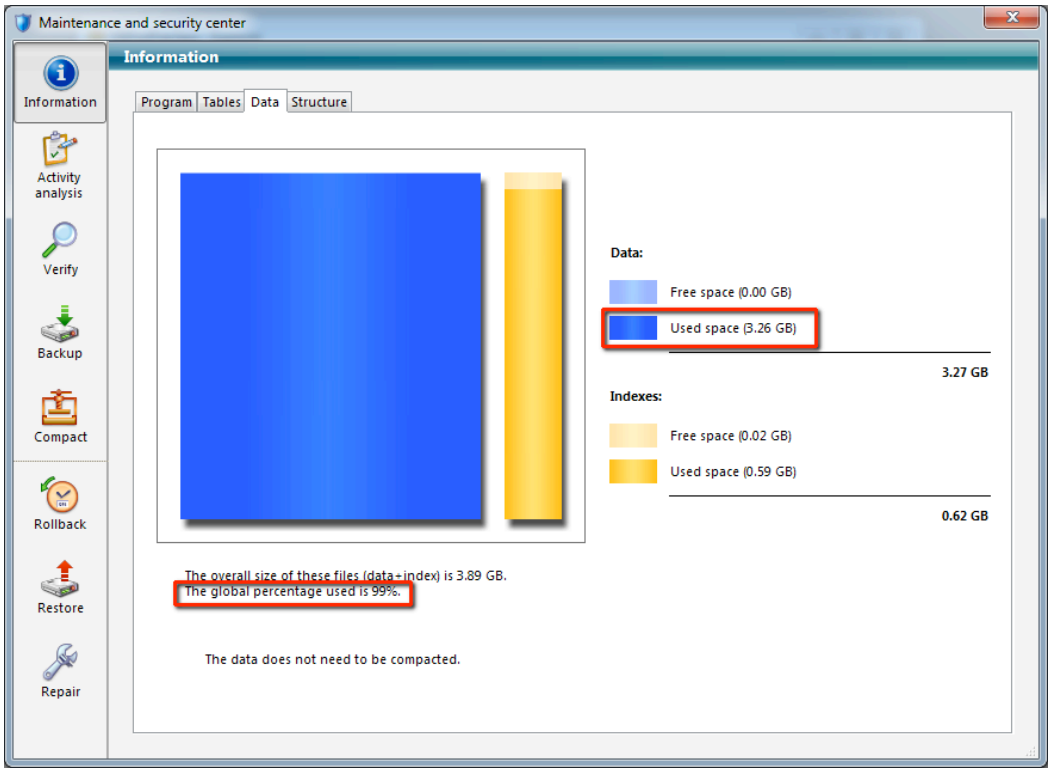
The MSC runs in standard mode if it is opened while 4D has the database open. In this case the database is live and running. Furthermore the MSC can access statistics like the number of records in the database and the global percentage of used space in the data and structure files.

Note: *See the "Access Rights" section below for more details about access rights in the MSC.*

Important Note: Depending on the size of the database, the display of content on the Information tab may be delayed, as shown below:



This is because it is necessary for the MSC to parse the data file and structure file in order to generate the metrics for this tab. Be sure to wait until the files are parsed before reacting to the messages displayed (i.e. the message "It is strongly recommended that you compact this data file" in the above screen shot is invalid until the data file is actually parsed). Once the parsing is complete, the UI will update:



Keep in mind that if another tab is selected before the parsing is complete, for example the Compact tab, the MSC icons will update as the files are parsed. During parsing the icons may represent the "compact needed" state:

Again, be sure to wait until parsing is completed before heeding the icons.

The tasks executed in standard mode (especially Verify and Backup) can be run in "real time"; that is these tasks can be performed while users are logged into the system. There are some constraints:

- No new connections are allowed for the duration of the task.
- Existing connections work in read-only mode. If a write is attempted, the process will be paused until the maintenance completes.

Maintenance Mode

In maintenance mode the database is not opened by 4D. No startup code runs, no database cache is created, no clients can connect, etc. In this mode only maintenance tasks can be performed; the Information tab will not display information about the Data or Structure files for example.

Note: *See the "Access Rights" section for more details about access rights in the MSC.*

Access Rights

Certain MSC functions are not available depending on the type of 4D application being used, the MSC mode, and the current user. Here are the guidelines:

- Functions which affect the application structure (verify, repair and compact) can only be accessed from the 4D Local Mode and 4D Server applications. In 4D Remote Mode and 4D Desktop the corresponding buttons and tabs are hidden.
- Information concerning the contents of data and structure files is only available when the database is open by 4D; as described previously, this is when the MSC is opened in standard mode.
- Data compacting, rollback, restore and repair functions can only be used with data files that are not open by 4D; as described previously, this is when the MSC is opened in maintenance mode. If one of these tasks is requested, the MSC shows a warning, closes the database, and restarts in maintenance mode.
- If passwords have been activated, data compacting, rollback, restore and repair functions can only be accessed by the Administrator and Designer users.

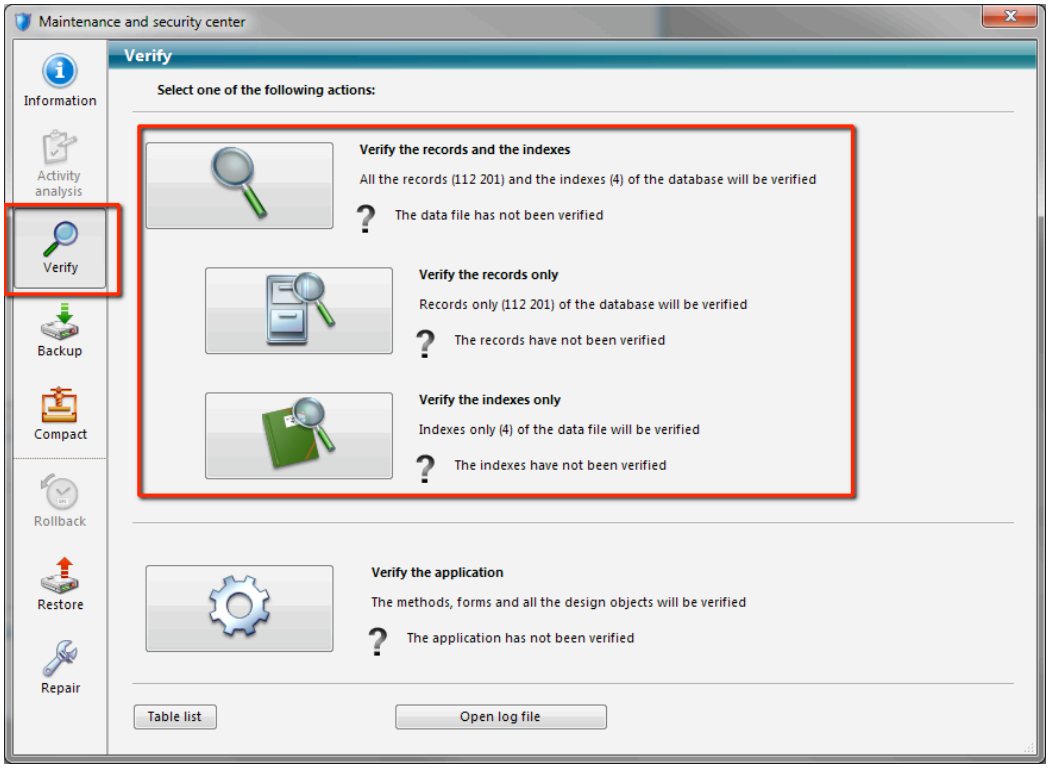
Verify

Database verification involves checking the structure and, to some extent, the content of 4D database files (this includes the structure file, data file, and index files). Verifies are an important task for any 4D database and should be done on a regular basis in order to detect problems as soon as possible.

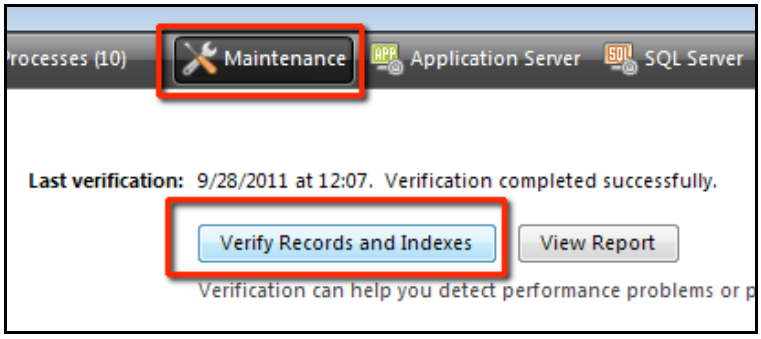
There are several different options for database verification, and different ways to access those options. This section explains each option. Note that structure verification, being largely a development task, is not covered here.

A data verify can be accessed in one of three ways:

- From the Verify tab of the MSC:



- From the Maintenance tab of the 4D Server Administration dialog:



- From the language using the VERIFY CURRENT DATA FILE command

There are actually 4 different kinds of Verify:

- Verify Records
- Verify Indexes
- Verify Records and Indexes
- Verify All

These correspond to the parameter options for the VERIFY DATA FILE and VERIFY CURRENT DATA FILE commands. Some of them also correspond to the Verifies offered by the UI as shown above. In particular, here is how each GUI task maps to the language:

Task	4D Language
Administration Verify	**VERIFY CURRENT DATA FILE**(Verify Records+Verify Indexes;0;"")
MSC Verify All	**VERIFY CURRENT DATA FILE**(Verify All;0;"")
MSC Verify Records	**VERIFY CURRENT DATA FILE**(Verify Records;0;"")
MSC Verify Indexes	**VERIFY CURRENT DATA FILE**(Verify Indexes;0;"")

The different options are designed to allow the developer to selectively verify parts of the database in order to make the verify operation take less time. As mentioned previously, a verify will prevent new connections to the database and place existing connections is read-only mode. The trade-off is the quantity of database parts being verified versus the full availability of the database. In general, a Verify Indexes or Verify Records on its own will be faster than a full verify.

However note that the MSC button to "Verify the records and indexes" is not the same as the "Verify Records and Indexes" button in the Administration dialog. The MSC option performs a "Verify All"; this operation calls data engine function that handles the full verification. The Verify Records and Verify Indexes options are managed at a higher level (in the C++ code) and only call into the database engine for specific parts of each verify operation. There are two points to be made here:

- The Administration verify is the only verify that can be executed from a client by default.
- The MSC Verify is generally faster because, since it executes in the data engine, it is highly optimized.

Thus the recommended best practice for database verification is to use the MSC when possible (or use the language! See the next section).

4D Language Support

The 4D language supports many MSC tasks via the following commands:

- VERIFY DATA FILE
- VERIFY CURRENT DATA FILE

- Compact data file

In fact, as hinted at in the previous section, the MSC is calling the same entry points as these 4D commands. What does all this mean? Two things:

- Automation
- Granular Control

Note: *keep in mind any maintenance command acting on the current database will need to be executed on the server, not the client. This can be accomplished via the EXECUTE ON SERVER command or "Execute on Server" method attribute.*

Automation

The exposure of MSC maintenance tasks via 4D commands makes it possible to fully-automate database maintenance. In particular this works very well for the Verify task (which does not require the MSC to be running in maintenance mode).

One of the most important features offered by these commands is a callback system by which the developer can detect the state of the maintenance task and take action. Here is an example:

```
VERIFY CURRENT DATA FILE(Verify All;0;"callback_Verify")
```

With this code the "callback_Verify" method will be called at regular intervals throughout the verification process. The callback method receives 4 parameters with the following values:

Event	$1	$2	$3	$4
Message	1	0	Progress message	Percent Done
Verification finished	2	Object type	OK message	Table/index number
Error	3	Object type	Text of error	Table/index number
End of execution	4	0	DONE	0
Warning	5	Object type	Text of warning	Table/index number

Note: *there is a $5 parameter but it is currently unused.*

With this system the verify task on the current data file can be fully automated by the developer and, furthermore, custom actions can be taken depending on the issue encountered (i.e. at minimum sending an email to the database administrator).

Granular Control

As opposed to the operations available in the MSC, the 4D language maintenance commands allow maintenance tasks to be filtered to a subset of the database objects. Here is an example that checks only three of the indexes in the database:

```
ARRAY LONGINT($tableNums_al;0)
ARRAY LONGINT($indexNums_al;2;0)

$indexNums_al{1}{0}:=4  // table number
APPEND TO ARRAY($indexNums_al{1};1)  // field number
$indexNums_al{2}{0}:=5  // table number
APPEND TO ARRAY($indexNums_al{2};2)  // field number
APPEND TO ARRAY($indexNums_al{2};3)  // field number

VERIFY CURRENT DATA FILE(Verify Indexes;0;"cb_Verify";$tableNums_al;$indexNums_al)
```

In this example only the index on field 1 of table 4, field 2 of table 5, and field 3 of table 5 will be checked. Similarly Verify Records can be restricted to specific tables by modifying the

$tableNums_al array. This gives complete granular control of database maintenance to the developer.

As mentioned previously, maintenance operations in standard mode put the database in read-only mode, so they need to be as fast as possible. The ability to restrict the maintenance task to a subset of the database makes it possible to both break the maintenance task up (e.g. verify 1 table every 30 minutes) or remove entire portions of the operation (e.g. do not verify tables in which the data never changes). In this way the maintenance can be less obtrusive to the user.

Best Practices

- Understand the difference between standard mode and maintenance mode:
  - Standard mode tasks can be executed while the database is in use.
  - Maintenance mode tasks can only be executed if the database is not open.
- Understand the MSC restrictions:
  - Only the Administration window Verify can be executed on a client.
  - Only Administrator and Designer can perform MSC tasks in a database with the Password Access System activated.
- Use "Verify All" (aka "Verify the records and indexes" in the MSC) for the quickest full Verify.
- Use the maintenance 4D commands instead of the MSC for:
  - Automation.
  - Granular control.

OPERATING SYSTEM

Currently the most important Operating System (OS) consideration for 4D, for both Mac OS X and Windows, is to use a 64-bit OS. This is critically important for maximizing the amount of memory available to 4D (even with 32-bit 4D). In addition to increased total memory access, the way in which memory is partitioned and used is more efficient with a 64-bit OS. Memory will be discussed in detail in the "Hardware" chapter.

Of course a 64-bit Windows OS is required to run the 64-bit version of 4D Server v12 (or any 64-bit software for that matter).

Keep in mind that moving from a 32-bit OS to a 64-bit OS is only a software upgrade. It is quite difficult to find a modern CPU that does not already support 64-bit OS's. In fact for Mac OS X Apple has been using a 64-bit kernel for several years. For Windows upgrading to a 64-bit OS is only a matter of installing the new software.

Note: *early Intel-based Macs (Mini, MacBook, iMac) were based on 32-bit Intel Core (Core Solo, Core Duo) processors. These machines do not support 64-bit OS's. The subsequently used "Core 2" (Core 2 Solo, Core 2 Duo, etc.) and "i5/i7" CPU's are 64-bit processors and therefore support 64-bit OS's.*

Windows

Here are some specific best practices for Windows machines running 4D Server.

Windows Server 2003

When running 4D on Windows Server 2003 there are some recommended "Performance settings" to use:

- Windows Processor Scheduling: set to "Programs" instead of "background processes".
- Windows Memory: set to "Programs" instead of "System Cache".
- Windows Page File: set to "System Managed Size" instead of "Custom Size".

The location for these settings may vary based on the OS but the key settings to look for are named above. They can typically be found following these steps:

- Click Start, click Run, and then type "sysdm.cpl" in the Run box.

- In the System Properties dialog box, click the Advanced tab, and then click Settings under Performance.

These recommendations apply to all 4D applications (single-user and server) whether running under a user account or as a service.

In Windows Server 2003 Microsoft enabled a security feature that reduces the size of the queue for concurrent TCP/IP connections to the server. This is exposed through the registry via the SynAttackProtect key. This feature helps prevent denial of service attacks; however, under heavy load conditions in 4D, the TCP/IP protocol may incorrectly identify valid TCP/IP connections as a denial of service attack. Be sure to review the information in this Tech Tip to deal with this issue:

- http://kb.4d.com/search/assetid=76127

For a Server 2003 machine running 4D Server, be sure to set the Network Properties to optimize for network applications, as described here:

- http://technet.microsoft.com/en-us/library/cc728171(WS.10).aspx

Windows Server 2008

For best 4D performance, set the machine to the "Application Server" role:

- http://technet.microsoft.com/en-us/library/cc754024(WS.10).aspx

This can be managed using the "Add Roles" Wizard in Windows Server 2008:

- http://technet.microsoft.com/en-us/library/cc732263.aspx

Microsoft has identified a problem with the system file cache feature in Windows Server 2008. The issue is that the system will aggressively cache files into memory, thus starving applications of RAM, leading the thrashing.

The issue affects any large application and especially database applications because they typically require fast random access. For this reason it is extremely important to use the R2 release of Windows Server 2008. More information about the issue can be found here:

- http://support.microsoft.com/kb/976618

Mac OS X

Be sure to disable any unnecessary services in order to maximize the resources available to 4D. This can be especially important when using Mac OS X Server.

Best Practices

- Use a 64-bit OS, unless that hardware cannot support it.
- Use Windows 7 or Windows Server 2008 for TRIM support (more in this in the Hardware chapter).
- Avoid Windows XP 64-bit; it was not a good implementation.
- Windows Server 2003:
  - Set Windows Processor Scheduling: set to "Programs" instead of "background processes".
  - Set Windows Memory: set to "Programs" instead of "System Cache".
  - Set Windows Page File: set to "System Managed Size" instead of "Custom Size".
  - Check the status of SynAttackProtect, adjust accordingly.
  - Set Network Properties to optimize for network applications.
- Windows Server 2008:
  - Use the "Application Server" role.

- Use Windows Server 2008 R2.
- Mac OS X:
  - Disable unnecessary services.

HARDWARE

This chapter covers some specific best practices when it comes to hardware for running 4D.

Memory

The topic of memory for 4D covers two primary areas:

- Application memory
- Database cache

Of course these two items are related because the database cache occupies a portion of application memory. This section thus addresses two issues: how to maximize the total application memory for 4D, and how to maximize the size of the cache. Finally the non-ideal situation will be addressed.

Note: *This section uses the word "process" in an OS context, not a 4D context. A "process" on both Mac OS X and Windows is typically a single application.*

Note: *This section uses IEC binary units for memory values when intentionally expressing a binary number. For example a Gigabyte (GB) is a decimal value. It is 10^9 or 1,000,000,000 bytes. A Gibibyte (GiB) on the other hand is a binary value. It is 2^30, or 1,073,741,824 bytes. It is important to use binary ranges because memory addresses are defined as binary numbers (2^32 for 32-bit processors, 2^64 for 64-bit processors).*

Understanding Application Memory

Before looking at how to maximize memory for 4D it is important to establish what is meant by "application memory". Modern OS's use a combination of storage technologies to achieve a memory "address space" that is larger than the physical RAM in the machine. The RAM is differentiated from the total memory via the terms "physical memory" and "virtual memory", respectively.

Total application memory is directly related to the OS. The way in which the OS manages the memory determines how much memory is assigned to the application. In fact OS constraints can cause some confusing scenarios when it comes to trying to figure out just how much memory 4D can use. In this section these issues are explored.

Physical Memory

The term physical memory typically refers to RAM. RAM is fast memory, but relatively expensive, so the quantity of RAM is generally far less than the size of the total address space.

The most basic differentiation between how much physical memory a given OS can use is if the OS is 32-bit or 64-bit. The idea is that a single memory address is either 32-bits or 64-bits in total and refers to 1 byte of memory. Thus, in theory:

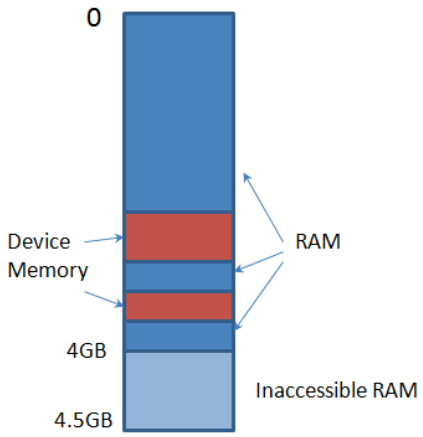
- 32-bits can address 2^32 bytes of storage:
  - 4,294,967,296 addresses
  - 4 GiB (4 gibibytes)
- 64-bits can address 2^64 bytes of storage
  - 18,446,744,073,709,551,616 addresses
  - 16 EiB (16 exbibytes)

So a 32-bit OS can only address 4 GiB of memory while a 64-bit OS can address 4,294,967,296 times more.

In practice the amount of physical memory the OS claims to support is quite a bit lower. Hardware limitations and SKU limitations are the most common restriction.

Hardware limitations are most prevalent for 32-bit OS's:

- The total RAM plus the total other physical memory in a given machine makes up the "physical address space". "Other physical memory" is memory for things like video cards, network adapters, etc.
- Using a technique called "memory mapped I/O" (MMIO), the OS tracks hardware addresses in the physical address space. Furthermore, for compatibility reasons, 32-bit Windows maps hardware addresses below the 4 GiB limit. For example if a 32-bit machine has a video card with 1 GiB of RAM, and 4 GiB main memory, the total addressable main memory will 3 GiB. Other hardware devices generally contain RAM and will therefore further reduce the total RAM that can really be used. Effectively each hardware address range "pushes" some of the RAM beyond the 4 GB limit:



- Intel developed a technology called Physical Address Extension (PAE) that allows the OS to use 36-bits for memory addresses instead of 32.
  - However, even with PAE, all 32-bit Windows client SKUs support a maximum of 4GB of physical memory. This turns out to be a practical limitation. In attempting to add support for more than 4GB of physical memory to 32-bit Windows, Microsoft found that many of the systems they tested would crash, hang, or become unbootable because of poorly implemented device drivers. The problematic client driver ecosystem led to the decision for client SKUs to ignore physical memory that resides above 4GB, even though they can theoretically address it.

64-bit OS's use MMIO as well but, since 64-bit addresses (and applications) can easily access more than 4 GiB of memory, the hardware addresses cause no practical limitation.

Limiting memory for the purpose of SKU differentiation is common as well. For example, Microsoft differentiates different Windows SKUs by artificially limiting the amount of supported physical memory. Here is the table for Windows Server 2008 R2 (which is only available in 64-bit editions):

Version	Limit
Windows Server 2008 R2 Datacenter	2 TB

Windows Server 2008 R2 Enterprise	2 TB
Windows Server 2008 R2 Foundation	8 GB
Windows Server 2008 R2 Standard	32 GB
Windows HPC Server 2008 R2	128 GB
Windows Web Server 2008 R2	32 GB

The complete list can be found here:

http://msdn.microsoft.com/en-us/library/aa366778(v=vs.85).aspx

In addition to SKU differentiation note that the 2 TB maximum physical memory limit is a practical limit. Microsoft will only certify on hardware that it can actually test and, at the time Windows Server 2008 was certified, 2 TB of physical memory was the largest amount available.

Apple's Mac computers generally list an artificial maximum value for supported RAM as well, though it is per-machine rather than per-OS.

Virtual Memory

The term "virtual memory" has unfortunately been used incorrectly as a synonym for "paged" memory. Paged memory is not virtual memory. More to the point, paged memory storage is **part** of virtual memory.

Note: *Memory "paging" is a process by which a page of physical memory is copied to disk storage so that the physical space can be used for something else. Paging is what allows the modern OS's to access far more memory than can be physically installed on the machine. On the other hand paging comes at a performance cost.*

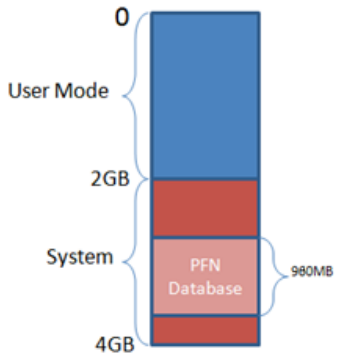
Virtual memory is more of a concept than a particular piece of hardware. Virtual memory is the total address space that the OS could **possibly** access. Portions of this address space may have no physical representation at all. For example Mac OS X, being that is uses 64-bit memory addresses, could conceivably address 16 EiB of memory. However there is no shipping Macintosh computer with this much memory installed (whether a combination of RAM, disk, or anything else). Virtual memory is, at least in part, literally virtual.

All applications on Mac OS X and Windows access a portion of this virtual address space. This portion is referred to as the "user-mode address space" on Windows and "process address space" on Mac OS X. When memory is requested (even if it is not immediately used; think of the 4D cache) the addresses allocated come from the application's portion of the virtual address space.

As with physical memory, 32-bit applications can theoretically access a total of 4 GiB of virtual memory while 64-bit applications have access to 16 EiB. However the true amount of virtual memory for an application can very as follows:

- 32-bit OS's can present a much lower value for per-process virtual memory for a few reasons:
  - System reservation: Windows, for example, reserves half of the address space for the system, giving the other half to processes. This means there is a limit of 2 GiB of addressable memory for processes in 32-bit Windows, not 4.
  - The technology exists for a 32-bit OS to access far more memory than the amount of RAM by simply storing a mapping to those locations. But this mapping must be stored in the virtual memory for the process. For example on Windows the data to describe this addressable memory is called the PFN

database. The PFN database occupies a portion of the system memory a process can access:



  - The /3GB parameter can be used to force Windows to reduce the system reservation by half. Thus a 32-bit process can access 3 GiB of memory instead of 2.
- For 64-bit OS's there is no issue; each 32-bit application can access the full 4 GiB of virtual address space because the OS has plenty of headroom to handle things like the system portion of memory and the PFN database. However for practical reasons, as with physical memory, processes in 64-bit Windows are limited to 8 TB of virtual memory.

So "application memory" is really virtual memory. Since it includes physical memory in this context it is not important to speak about the maximum amount physical memory that 4D can access. Increased physical memory increases **performance** (by reducing paging), not maximum storage. However the amount of physical memory can affect the amount of total virtual memory 4D can access because of the previously mentioned OS limitations.

Measuring Application Memory

4D accesses virtual memory, not physical memory. Thus the question is not really "How much memory can 4D use?" but, rather, "How much virtual memory can 4D use?"

To answer this question it is necessary to measure virtual memory. There are many different memory metrics built in to Mac OS X and Windows. These are the most useful ones:

Windows

Task Manager and/or Resource Monitor and/or Perfmon provide the following metrics:

- Working Set - This is the physical memory (RAM) in use by a process. Because of virtual memory this is obviously not an accurate reflection of the total memory in use.
- Commit Size = total amount of pageable virtual address space for which no backing store is assigned other than the pagefile.
  - For 4D it is **close** to the maximum memory.

There are other metrics available in Windows but unfortunately none of them actually represent the total virtual memory used by 4D.

Mac OS X

Activity Monitor (aka top) provides the following metrics:

- Real Memory – a "rough count" of the number of physical pages that the current process has. Tthis is not important in this context because, again, measuring

physical memory does not tell anything about the total virtual memory 4D is using; 4D uses virtual memory, the OS chooses whether or not to use physical memory.

- Virtual Memory - this is exactly the amount of virtual memory used by 4D, though it is rounded when displayed in the GUI.

In fact neither OS really has the best tools for measuring the total memory used by 4D. On Windows none of the available metrics actually give the used virtual memory for a process. This information is available to a process via Windows API calls but it is not exposed in the reporting tools available. The situation is better on Mac OS X as the Virtual Memory metric is accurate. In fact there is a better way to measure memory used by a 4D application that works on both platforms…

GET CACHE STATISTICS

The results of the GET CACHE STATISTICS (GCS) command include a value called "Used Virtual Memory".

For example, given the following constraints:

- 64-bit OS
- 32-bit 4D application
- Allocate memory in a loop until 4D runs out

The maximum Used Virtual Memory value returned by GCS will be very near 4 GiB or 4,294,967,296 bytes (GCS returns bytes). If the value returned by GCS is approaching this number, **the application is very likely about to run out of memory**.

Note that the amount of RAM installed in the machine in this example **does not matter**! 4D will be able to allocate 4 GiB of virtual address space in all cases. The RAM installed will only determine how much active physical memory 4D can use, but with paging all of the virtual memory is still accessible (if far more slowly). **Increased RAM increases performance, not maximum addressable memory.**

Thus the best way to measure the memory used by 4D is to use GET CACHE STATISTICS.

How to Achieve Maximum Memory for 4D?

Unlike measuring the memory used by 4D, the issue of maximizing the memory available to 4D is directly related to the physical RAM in the machine. With some of the limitations mentioned previously, as well as other challenges like the way in which the OS allocates memory for drivers or the way the OS rounds allocation units, it is important to make sure the machine has enough RAM to ensure best performance.

It is true that 32-bit 4D can access 4 GiB of virtual memory on a machine with only 1 GiB of RAM. But the performance may be undesirable because the OS will be forced to page much of the memory out (excessive paging is often referred to as “thrashing”). On the other hand, because of virtual memory implementations, it is impossible for a process to use **only** RAM. There will always be some page file use.

In all cases the 32-bit versions of 4D can access only 4 GiB of virtual memory, though it may be reduced by OS limitations as described previously. 4D Server v12 64-bit has no practical limitation on memory access (meaning no existing hardware is approaching the limit for 64-bit applications) but as explained previously, Windows imposes an artificial limit for certification. Here is a break down for the total memory that 4D can access assuming sufficient RAM:

- 32-bit OS - 2 GiB

    - 3 GiB with PAE enabled
- 64-bit OS, 32-bit 4D - 4 GiB
- 64-bit OS, 64-bit 4D Server - 8 TB (Windows only)

Follow these guidelines for maximizing memory for 4D:

- Use a 64-bit OS.
- If the 32-bit 4D application will need to access the entire 4 GiB address space, install at least 6 GB of RAM in the machine.
- For 64-bit 4D Server v12 the issue is more complex. 32-bit 4D has a hard limit of 4 GiB so it is easy to say how much extra RAM to have. Since 64-bit 4D Server has no practical limit, some analysis of the application is necessary. The cache size and amount of memory used by the application's 4D processes will determine the memory footprint. E.g. there is no reason to install 6 GB of RAM if the 4D application has a 100 MB cache and only uses another 100 MB at runtime. One goal can be to have enough RAM to fit the largest footprint possible in memory. E.g. if the database has a 16 GB cache and, at runtime with the maximum user count, uses an additional 3 GB of memory, shoot for installed RAM in the neighborhood of 36 GB.
- Do not forget about the OS system cache! The memory that 4D does not use is not just for other applications or OS processes. The system cache is an important part of both Mac OS X and Windows.
- See the next section for information about how to calculate the memory footprint of 4D.

Related Resources

These articles are relevant to understanding memory access.

- Windows:
    - http://blogs.technet.com/b/markrussinovich/archive/2008/07/21/3092070.aspx
    - http://support.microsoft.com/kb/888732
    - http://support.microsoft.com/?kbid=291988
    - http://msdn.microsoft.com/en-us/library/aa366778(v=vs.85).aspx
- Mac OS X:
    - http://developer.apple.com/library/mac/#documentation/Performance/Conceptual/ManagingMemory/Articles/AboutMemory.html
    - http://support.apple.com/kb/TA27734?viewlocale=en_US

How to Calculate the Memory Footprint of 4D?

Rather than measuring the runtime memory footprint of a 4D application it is also possible to estimate it analytically. There are 3 main metrics that can be estimated:

- Database cache
- Process stack sizes
- Process and interprocess variables

The database cache size is straightforward. Just use the value specified in the Database Settings for the estimation.

Process stack sizes will be documented below.

The symbol table generated by the compiler can be used to get an idea of the memory cost of variables in the database. Simply generate the symbol table and add up the size of each variable (4 bytes for longints and reals, 2 bytes for integers, text is 2-bytes per character, etc.

The following is an example of how to perform this kind of calculation.

Note: *this technique is useful to get an "order of magnitude" estimation for the memory footprint of the database, not to exactly calculate the memory that will be used. This is useful for determining hardware requirements and also to determine if stack sizes need to be reduced globally, for example.*

Baseline Usage

This is the usage for a 4D Server application without any client processes.

System Libraries	1,000 MB
Cache	1,200 MB
IP Variables	10 MB
4D Server Threads	10 MB
Total	2,220 MB

- 1,000 MB for system libraries (approximate)
- 1,200 MB for the cache (set in the Database Settings)
- 10 MB for IP variables is more like an upper bound, allowing for a few large BLOBs, for example.
- ~10 MB for kernel preemptive/cooperative threads (cache manager, index builder, interface, server listener, etc.)

Per-Process Usage

Here we look at the stack sizes for client processes, along with an example cost for process variables (which, again, can be calculated using the symbol file).

Note: *SDP refers to the ability of the SET DATABASE PARAMETER command to reduce the preemptive process stack size with selector 53.*

Process	Default	SDP 512K	SDP 256K
Cooperative twin	.25 MB	.25 MB	.25 MB
Preemptive twin	1 MB	.5 MB	.25MB
Total	1.25 MB	.75 MB	.5 MB

With SQL Triplet	1 MB	.5 MB	.25 MB
Total	2.25 MB	1.25 MB	.75 MB

Process Variables at 500 KB	.5 MB	.5 MB	.5 MB
Total with twins	1.75 MB	1.25 MB	1 MB
Total with triplets	2.75 MB	1.75 MB	1.25 MB

Process Variables at 1.5 MB	1.5 MB	1.5 MB	1.5 MB
Total with twins	2.75 MB	2.25 MB	2 MB
Total with triplets	3.75 MB	2.75 MB	2.25 MB

The SQL triplet processes are listed separately because they are created as needed to handle SQL calls. If no SQL calls are made from the clients, this potion can be removed.

The two proposed process variable footprints were selected as follows:

- 500 KB of process variables per process is reasonable for processes that use only scalar variables.
- 1.5 MB of process variables per process reflects processes that use Text, Blobs, Pictures, arrays, etc.

Example Calculation

Assume the following:

- 32-bit 4D server
- 64-bit OS
- 6 GB RAM
- No 4D SQL commands
- Client processing is executed in the Application process (i.e. only 1 process per-client)
- The goal is to reach 500 simultaneous users

Remember that the maximum virtual memory for 32-bit 4D Server is 4 GiB. Using the previous estimations, here is the size for this scenario:

500 KB Variables	Default	SDP 512K	SDP 256K
Baseline	2,220 MB	2,220 MB	2,220 MB
500 Processes	875 MB	625 MB	500 MB
Total	3,095 MB	2,845 MB	2,720 MB

1.5 MB Variables	Default	SDP 512K	SDP 256K
Baseline	2,220 MB	2,220 MB	2,220 MB
500 Processes	1,375 MB	1,125 MB	1,000MB
Total	3,595 MB	3,345 MB	3,220 MB

Warning: this calculation is certainly simplified. It does not take into account everything that can be executed on the server under normal operation (sets, arrays, BLOBs, pictures, transactions, etc.). But as an estimate it is possible to see that the Default stack configuration with 1.5 MB of variables per-process is already dangerously close to the 4 GiB virtual memory limit. By contrast the SDP 256K configuration with 500 KB per-process variables has plenty of extra head room (over 1 GB). For the latter configuration the database cache size could be increased for better performance.

Database Cache

In many ways the cache is the core of all database engine I/O in 4D. Access to the data file, structure file, index files, replication headers, etc. achieved via the cache. The cache is a critical and integral part of 4D.

With the introduction of 4D v11 SQL the cache was dramatically revamped. Support for 64-bit memory addressing and improved memory management in 4D allowed the cache to be much larger. The cache was redesigned to be much more efficient, especially in light of the increase in size. Finally new tools were added to aid the developer in tracking the usage of, and troubleshooting the cache.

With 4D v12 the cache continues its evolution of increased performance and better management.

This section is about some of the best practices to use when working with 4D's database cache.

Maximum Cache

By its nature the cache is a way to improve performance by placing data that would normally be accessed on the disk into RAM. In theory, the bigger the cache is, the better the performance for 4D. In practice this is not always the case, but the principal is valid.

When looking at the maximizing the size of the cache for 4D there are two main points to take into consideration:

- How big can the cache be?
- What machine configuration is required to support it?

The maximum size for the database cache is as follows:

- 32-bit: 2,384 MB
- 64-bit: effectively unlimited (8 TB on Windows, 16 EiB on Mac OS X)

In previous releases of 4D v11 SQL it should be noted that the maximum cache was 4,095 MB. In retrospect, and through the version 11 lifecycle, this was determined to be too large. Since 32-bit applications can only access 4 GiB of virtual memory **total**, a 4095 MB cache leaves nearly nothing for the rest of 4D. The maximum cache size for 32-bit 4D was decreased to 2,384 MB in order to try and help keep developers from shooting themselves in the foot, so to speak, by setting the cache too high. To be clear: if there is insufficient memory for the rest of 4D, 4D will crash. On the other hand if the cache is perhaps "too small", 4D will perform slower. The potential for slightly reduced performance is a better trade-off than a crashing database.

For 64-bit, there is no effective maximum cache size because there are no systems with enough memory to approach the maximum of 16 EiB of memory for 64-bit addressing. It is perfectly valid to set the cache size as large as the data file in 4D Server v12 64-bit, assuming the necessary hardware is in place.

Achieving the maximum cache size is unfortunately not just a matter of entering a large number in the Database Settings. The biggest challenge is that the cache **must be contiguous**. That is, when 4D starts up and asks the OS for the cache memory it is requesting a contiguous chunk of memory. The limitation is that 4D is never running in a vacuum; every application process requires system libraries, drivers, etc. These items all occupy portions of the process's memory space. Thus when 4D asks for the cache memory it may not always get the desired amount from the OS. This is particularly a limitation for 32-bit OS's (as explained in the previous section). Thus the recommended configuration to support maximum cache size is as follows:

- 32-bit 4D:
  - Use a 64-bit OS in all cases.
  - Install at least 6 GB of RAM.
    - A previous recommended of 8 GB RAM minimum was given at previous Summits; this reflected the time when the maximum cache size for 4D was 4095 MB. Now that the maximum cache size is 2,384 MB it is generally possible to achieve this size with only 6 GB RAM.
- 64-bit 4D Server v12:
  - Obviously a 64-bit OS is required.
  - It is possible to allocate a cache of any practical size (10's of GB's for sure), even on a machine with little RAM.
  - However it is important to ensure there is at least as much RAM in the machine as there is requested cache size, plus some extra for the OS. This is important for a couple of reasons:
    - System cache: both Windows and Mac OS X maintain their own RAM cache of disk content. This is referred to as the "system cache". It is important not to "fight" the system by monopolizing all of the memory with 4D. Some thought should be given to allowing the system some head room for its own file caching.

- Paging: given that both 4D and the system can potentially consume large amounts of virtual memory, if they both consume enough memory that paging occurs, the 4D cache is heavily marginalized.
    - Say 4D wants to flush the cache to disk.
    - 4D attempts to read a page from the cache, but the page has been paged to disk. It needs to be swapped in so, effectively, 4D is reading from disk. This completely negates the purpose of the database cache.
    - Once the cache page is back in RAM, 4D needs to flush the change (write to disk).
    - Furthermore the OS pays a penalty since, in order to page 4D's requested memory into RAM, something else must be paged out.

Flush Interval

The default value for the "Flush Data Buffers" Database Setting in 4D v12 is 20 seconds. In previous versions of 4D it was 15 minutes.

4D v12 features an improved cache manager that is more efficient when there are fewer changes in the cache to flush. Therefore flushing more often tends to be more efficient than in previous versions. The goal is to make the overall size of the flush smaller. This is particularly effective when using SSDs.

Note: *this change does not automatically apply to converted databases; the previous value will be used. However unless you are experiencing performance issues with the cache there is not necessarily a reason to change it.*

Adaptive Cache

Note: this is the same content from the Database Settings section, but repeated here for quick reference.

A common misconception is that adaptive cache means the size of the cache can change at runtime; i.e. the cache is resized dynamically. This is false.

Another common assumption is that adaptive cache takes runtime resources into consideration when calculating the cache size. I.e. it is presumed that the cache size might be based on the free memory in the machine or how many programs are running. This is false. Unless the hardware on the machine is modified (i.e. adding or removing RAM) the cache size **will not** change.

The purpose of adaptive cache calculation is very simple: this setting is useful for "shrink-wrap" products where the developer has no control over the system on which the database might be installed. In this situation it is prudent to try to set a cache size that will not dominate the machine memory. Since the developer has no direct control over the hardware used, the adaptive cache setting allows for a cache of varying sizes depending on the hardware.

If the database is deployed on a single machine, or on several machines where the developer has direct control over the cache size (whether by setting it directly, or via training and documentation), the adaptive cache feature offers **no advantage** and, furthermore, makes configuring the cache size unnecessarily complex. The best practice in this situation is to turn the adaptive cache off and simply set the value directly.

Non-Optimal Cache

So far this section has only addressed achieving maximum cache size. This is presumptive in that it assumes a best-case scenario of optimal hardware configuration. What if the hardware is not optimal? How should the cache be set? It also assumes that

the maximum cache size is **needed**. This is rarely the case; generally only the largest/highest load systems need large cache sizes.

It is important to consider that the cache is only part of the total memory footprint of 4D and, furthermore, that 4D is not the only software that requires memory from the OS. On the other hand it is important to not set the cache to small! Some real-world scenarios that have been observed:

- 2.3 GB cache on a machine with 3 GB RAM
  - This pretty much guarantees the OS will be paging a lot, thus reducing performance.
- 100 MB cache for 25 users on a machine with 16 GB RAM
  - 100 MB was the default in previous versions. It has since been increased to 400 MB. Be careful to check converted databases to make sure the cache is not too small.

Do not set the cache so large that it would monopolize the memory in the machine. On the other hand do not set it so small as to hobble the database.

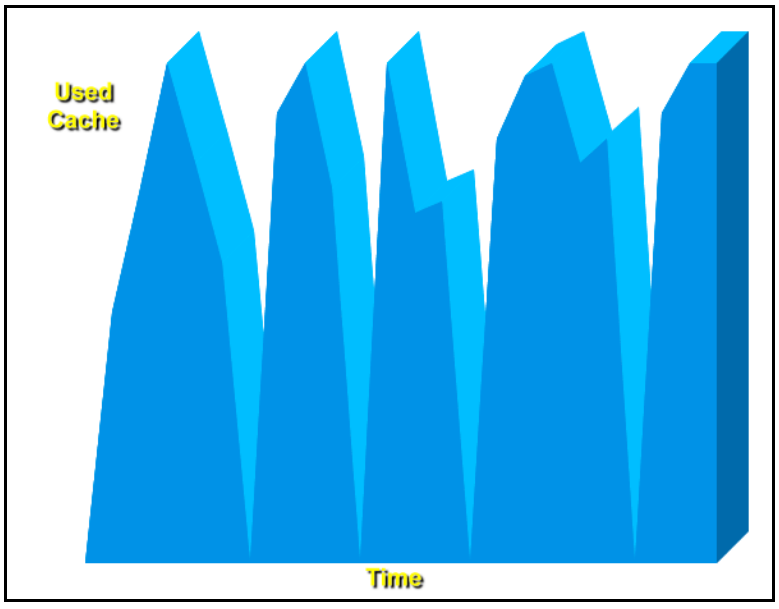
By far the best way to determine optimal cache size is through analysis of the cache usage over time. Part 3 of the pre-class covers cache analysis in detail but here is one example:

The foundation if cache analysis is the GET CACHE STATISTICS command:

```
GET CACHE STATISTICS( 1 ; arrNames ; arrValues ; arrCount)
```

This command returns the used cache size and, when called in a loop, can be used to graph the cache usage over time.

Here is a simple example of a cache that is too small:



A normal cache purge should be 25% of the total cache. In this example the entire cache is being purged over and over. This is a classic case of a cache that is too small.

Solid-State Drives

You will probably hear the recommendation to use Solid-State Drives (SSDs) many times this week, as you have at past Summits. This section is meant to help you understand why we are making such a strong recommendation.

4D is a disk-intensive application. It is also generally well-understood that magnetic hard disk drives (HDDs) are not terribly fast when compared to flash memory or RAM. The recent push of SSDs into mainstream computing has really brought to light just how slow HDDs really are. However the affordability of mainstream SSDs brings the advantage of pointing out the solution along with the problem. SSDs make it obvious that HDDs are a huge bottleneck. Simply replace the HDD with an SSD and all is well, right?

This section examines the answer to this question from the 4D perspective.

Why Storage Speed Matters

To fully appreciate the impact SSDs can have on 4D it is important to understand just how critical storage is to 4D. Like any database, 4D is a disk-intensive application. Of course data lives on the disk (in the data file) but also consider these important disk related features:

- Data and indexes live on the disk (in the data and index files). It is especially critical that indexes be loaded as fast as possible in order to provide maximum query speeds.
- 4D uses the disk in conjunction with the cache, so storage speed becomes important for cache speed as well.
- Indexes are built using the disk (you may notice files being created in the "temporary files" folder while indexes are building).
- Fast disk access is important for the data log file. Naturally the data log file should be considered critical for any 4D application as it provides quick recovery in the event of a failure (by integrating the most recent changes at startup) and makes 4D Backup more robust (by providing the ability to integrate all changes since the last backup).
- Cache flushing speed is directly determined by disk speed.

These are just some of the important pieces of 4D that rely on the disk and, therefore, are subject to the performance of the disk.

Of course fast data access may be important but this consideration tends to be tied to the design of the database for a couple of reasons:

- First, 4D already optimizes data access in many areas in order to only load the data that is necessary.
- Second 4D uses a data cache in RAM to mitigate some of the performance loss associated with going to the disk. If the design of the database is such that the most often accessed data manages to stay in the cache, then disk access speed becomes less of a factor.

Yet perhaps most important to 4D is random access speed for both reads and writes. Being a database 4D tends to want access to (and make changes to) relatively small pieces of disk (think records though in reality it is "blocks"). Typically 4D is not being used to move large, multi-gigabyte files so sequential disk speed tends to be less important. Fast random access is key.

So I need a Fast Disk…So What?

This may seem relatively obvious. The faster the disk, the faster 4D can be. Why make such a point of talking about it?

What is new here is it was not generally understood just how much of a bottleneck the disk could be for 4D. This was due, to some extent, to a lack of access to anything faster than a really fast HDD. The realm of RAM disks and SSDs was out of reach for the typical 4D application simply because the costs were astronomical (thousands of dollars

for a single drive, let alone the IT infrastructure that typically comes along with this level of cost).

In the past several years the prices for SSDs have dipped into the realm of mainstream computing. The opportunity has arrived to take advantage of what is perhaps the single most important piece of hardware for 4D, and improve the single biggest bottleneck. The performance improvement that can be had using an SSD is shocking, to say the least.

Note: k*eep in mind that, as with all other hardware considerations, database performance is still heavily dependent on database design. The disk may not be a bottleneck in your 4D application. The difference now is that SSDs are cheap enough to try them and see what kind of improvements they bring.*

Where do I sign up?

Not so fast. There are still definitely some pitfalls to picking the right SSD. The market for this technology is young and still evolving. In this section an attempt is made to educate the 4D Developer about SSDs in general, and then point out specifics that matter to 4D. Finally some benchmarks will be presented.

Understanding SSDs

This section discusses some of the technology behind SSDs in brief. There is a ton of information about SSDs available online, if further reading is desired.

SSD Innards

SSDs are made up of some of the same components as an HDD:

- A controller
- Some cache RAM
- Storage media

The primary difference between an SSD and HDD is the storage media. Hard drives use spinning, metal platters coated in a magnetic material and a mechanical arm that is positioned to read the data. SSDs use flash memory.

SSD Advantages

What really sets an SSD apart, in terms of performance, is:

- No moving parts (the obvious one)
- File fragmentation has no effect on performance
- The controller (this is a bit trickier and explained in a separate section)

Because an SSD has no moving parts (no spinning platters, no read head) they bring a lot of advantages to the table:

- They use less power
- They run cooler
- They are faster, especially with regards to seek time and random access

The effect of file fragmentation on an HDD is directly attributable to the moving parts: the read head has to move from location to location in order to get the different pieces of the file. SSDs have no read head, so they are completely unaffected by file fragmentation.

SSD Disadvantages

There are a few disadvantages to SSDs:

- Cost
- Size
- Lifespan *
- They get slower with use

SSDs are still vastly more expensive than HDDs. Count on spending up to $1000 for a decent SSD to run 4D, and that is only for 100-500 GB of storage. Though flash technology has evolved greatly in an attempt to increase storage space, the cost for an SSD with comparable storage to today's HDDs (2+ TB) is in the thousands of dollars.

On the issue of lifespan, flash memory tends to have a much lower lifespan than the magnetic disks used in HDDs. To put a finer point on it: a single flash "cell" (1 or 2 bits, explained later) supports a finite number of write operations. **Flash cells will fail, period.** On the other hand, because HDDs contain moving parts, though the media itself may not fail the other parts certainly will.

Note: *I put an * on this point because, although flash memory has a finite, known lifespan, the SSD manufacturers have already addressed this issue. I will go into this in more detail but the overriding understanding I want the reader to have about SSD lifespan is that it is really a non-issue.*

The issue of performance degradation will be addressed in the next section.

SSDs in Detail

In the preceding sections it was mentioned that a few things would be explained in greater detail and this section will address those topics. However several of those topics (Fragmentation, The Controller, etc.) all tie into one main issue: the fact that SSDs get slower as they are used. This will be addressed first.

Understanding SSD Performance Degradation

All SSDs perform worse with use ("use" in this case meaning write operations). This is caused the fact that flash memory cannot be directly overwritten like an HDD; it must be erased before it can be rewritten. Note that this is an architectural limitation: it is not really a "fixable" problem without using some other storage medium besides flash memory.

Note: *Before I scare you off, SSD manufacturers are already dealing with this problem and providing solutions that effectively solve the issue.*

When you delete a file in any modern OS, the response at the drive level is nothing. The drive does not do anything at all. The OS updates the File Allocation Table (FAT) to indicate that the space is free but the data is still actually on the drive. Of course as new files are saved, the old data will eventually be overwritten.

Note: *This is why a lot of data recovery software works. It simply restores the data that's already sitting there on the drive.*

Just like an HDD, SSDs are broken into "blocks" of fixed size. Each block contains multiple "pages" of memory. SSDs can read and write pages at-a-time, but only erase blocks at-a-time. The crux of the issue is that a block is bigger than a page. I.e. blocks contain multiple pages.

So the SSD does not know which files (pages) have been deleted (problem 1) and when it goes write a file, it can only erase by blocks (problem 2).

Over time each block will contain both pages that are in use and pages that represent deleted files (for reasons explained later, it is intentional that every block is partially occupied). Thus, over time, every write operation actually becomes several write operations.

Write Amplification

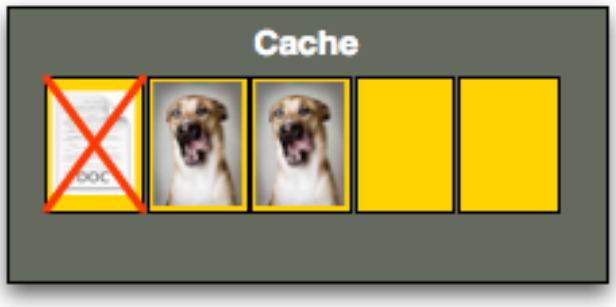
Imagine that we want to save a file to an SSD. In this example our SSD has a block size of 5 pages. In this scenario a DOC file occupying 1 page has already been written to the SSD and later deleted. A picture of a dog, occupying 2 pages, is also already on the disk. We want to write a picture of the beach that needs 3 pages of storage, as below:

1

Presume also that all other blocks contain occupied pages (otherwise we could just save to an empty block).

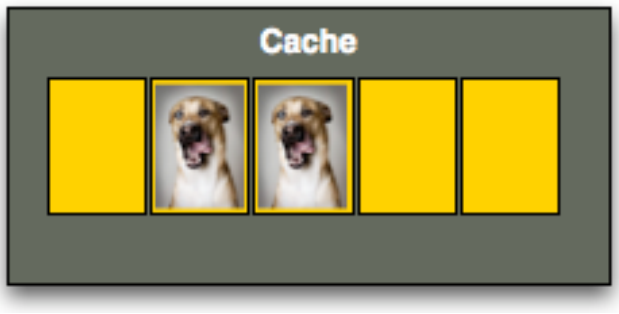
The SSD knows this block is occupied, but as explained before, it has no idea that the DOC file has been deleted. It can only erase entire blocks and that obviously would not be safe (the existing data would be lost). So how can it save the new file? First it must read the block into a cache/buffer.

2



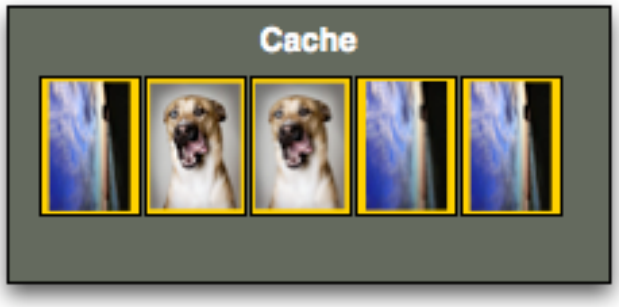
Then it needs to identify the free space and clear any deleted files (but in the cache, not on the disk yet).

3



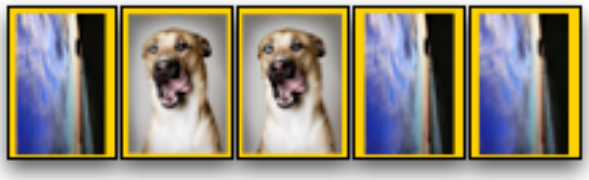
Then it writes the new file to the cache.

4



And finally erases the original bock and writes the new block back to disk.

5



Thus, in order to write only 3 pages of data, the drive actually wrote 5 pages total. Once every block contains occupied pages (and this will happen relatively quickly, see the next section) **every write operation becomes read/modify/erase/write!**

This problem is called “write amplification”. It is a ratio given by this formula:

$$\frac{\text{USER DATA WRITTEN}}{\text{ACTUAL DATA WRITTEN}} = \text{WRITE AMPLIFICATION}$$

Note that this is a “bounded” issue. That is to say the performance will degrade to a certain point (once all blocks are occupied) and not get any worse.

Note: *Typical write performance degradation in the SSDs available in 2011 is 25-35%.*

SSD Memory: Flash Cells

As mentioned before, SSD flash cells have a limited lifespan when it comes write operations. Mainstream SSDs typically use flash in one of two forms:

- Multi-Level Cell (MLC)
- Single-Level Cell (SLC)

The difference here is the number of bits that each cell represents. SLC flash uses one cell per bit. MLC flash uses one cell for 2 bits. Why does this matter? MLC flash is cheaper to produce than SLC because it offers twice the density. On the other hand SLC flash lasts longer. Because MLC flash contains 2 bits per cell, each cell will typically be written to twice as much as SLC.

There is a subtle point being made here, or perhaps not so subtle if you do any IT managing; all this talk about lifespan may have you worried about using SSDs at all…

Not to worry, see the next section!

The Controller

It was mentioned earlier that the SSD controller plays an important part in terms of SSD performance.

In fact the SSD controller is the single feature that differentiates the various SSD retailers. Most of them buy their flash memory from the same sources so the speed and lifespan of the memory is not a differentiating factor. The controller is really what sets things apart.

The controller features fall into two general categories:

- Lifespan features
- Performance features

Lifespan Features

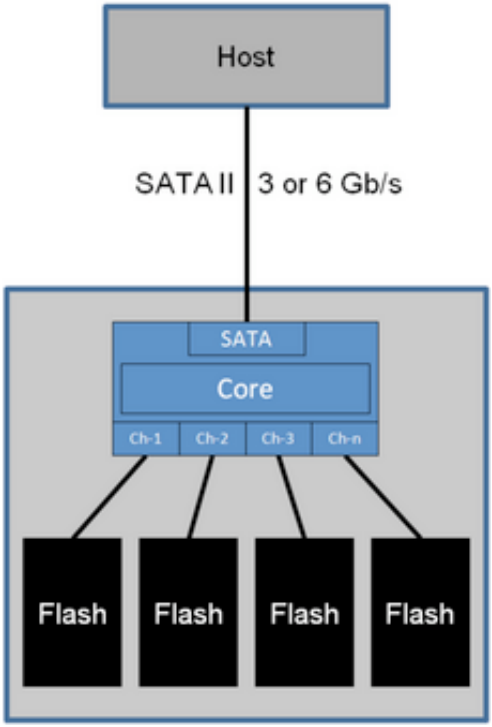
We know that flash cells have a limited write/erase lifespan. The controller must, therefore, do all it can to avoid a drive failure. All SSD controllers implement a simple scheme for this called “wear-leveling”.

Say you want to save a 100MB file. An HDD will start at one end of the disk and write the file, sequentially, wherever it has free space. A fancier scheme might attempt to make the file contiguous (certainly defragmenters do this) so that reads are faster.

An SSD controller will spread the file far and wide over all possible blocks in the disk. It does this for two reasons:

- Performance
- Integrity

From a performance standpoint, intentionally fragmenting the file during a write screams out for a very obvious feature: parallelism. A good quality SSD will have multiple layers of flash memory accessible via multiple channels:



Thus the SSD controller has the ability to write to these memory channels at the same time. So continuing the previous example, the 100 MB file can be split into chunks, with multiple chunks being written in parallel. Generally, the more expensive the SSD, the more flash channels it has. This offers a huge performance boost over HDDs.

Note: *generally the* ***larger*** *the SSD the more channels as well, so performance with SSDs often scales with size.*

From an integrity standpoint, if the file were to be saved sequentially or, even worse, contiguously it would mean that every time that file is modified the same flash cells are being erased and written to. By spreading file data all over the drive the SDD controller is attempting to limit the number of writes to each cell. In other words the SSD controller **intentionally** fragments files. This is "wear-leveling".

Now, thinking about the performance degradation problem already discussed, wear-leveling would seem counter to everything we know about SSD performance; intentionally spreading files all over the SSD means that, eventually, every flash block will contain occupied pages. This is absolutely correct! Wear-leveling contributes directly to write amplification, or rather causes the write amplification problem to appear sooner than it would if the controller wrote sequentially. It is simply a balance that must be struck between performance and lifespan.

There is another important feature to mention in this context: all SSDs (especially MLC SSDs) are "over-provisioned". What this means is the SSD actually contains more flash memory than its stated capacity. The controller use this extra space for further wear-leveling, to further increase the lifespan of the drive and delay the effects of write-amplification.

Performance Features

Here are the important drive controller performance features:

- Parallelism
- Garbage collection

- TRIM
- Queue-depth optimization

In the "early days" (2008 ☺) of mainstream SSDs this was a big, valid concern. Nearly every shipping SSD had the write amplification problem and the only way to "fix" the performance degradation was to reformat the drive (not a quick format mind you, because that leaves the occupied pages in place). In fact an entire generation of SSDs using a controller from JMicron was sold with no solution to this problem. If you bought a JMicron-based drive, you lost 25-35% of the drive's initial performance and there was nothing you could do about it but format the drive.

Some SSD manufacturers released tools to be used in order to "clean" the SSD. These tools essentially notified the SSD of any deleted pages, but needed to be run manually.

Other SSD manufactures implemented proprietary "garbage collection" schemes to deal with this problem. With SSD garbage collection the controller scans through the SSD and identifies occupied pages. It then moves clumps of occupied pages so that they fill a single block, thus freeing up other blocks. As you can imagine, the drive cannot be in use for this kind of operation so garbage collection can only occur when the drive is idle.

A big boost came with a new ATA command called TRIM. If an operating system supports TRIM, what it means is the OS keeps track of files deleted from the SSD and notifies the drive controller. The drive controller then knows which pages are no longer occupied and can be smarter about both saving files and also how it handles wear-leveling. But there are some catches with TRIM:

- 4D OS Support: As of the writing of this session only Windows 7 and Windows Server 2008 support TRIM natively. For 4D this means you need to run Windows to get the benefits of native TRIM support.
  - Mac OS X Lion adds TRIM support, but only for SSDs included with the machine.
- Drive Support: Most SSDs (many are now in their second generation) support TRIM but it is definitely important to check for TRIM support when considering an SSD. As SSDs move forward, expect TRIM support to be integrated into every major OS and drive but this is an important feature to be aware of.
- RAID Support: no current RAID controller supports TRIM.

The topic if queue-depth optimization is an important one when comparing different SSDs. The term queue-depth simply means concurrent accesses to the drive. The more applications or threads making requests to the drive, the higher the queue-depth. The word "queue" is key; there really is no such thing as parallel access to a SATA drive. So the requests are queued.

As the queue-depth increases, SSD performance will tend to diverge rather quickly. SSDs with controllers optimized for high queue-depth (e.g. Intel) will continue to perform impressively with a large number of simultaneous requests while other drives fail miserably.

Note: *it is important to stress that queue-depth performance matters most when comparing different SSDs, i.e. all mainstream SSDs, even with poor queue-depth optimization, will still be faster than an HDD.*

SSD Maintenance

Fragmentation

Why does fragmentation not affect SSDs?

Thinking back to the example about saving a picture to the SSD, where each block already contains some occupied pages, and with an introduction to wear-leveling, you might be thinking that SSDs intentionally fragment themselves. This is absolutely correct! Fragmentation is what keeps the flash cells alive!

The reason fragmentation does not affect SSD performance is quite simple: there are no moving parts. There is no "seek time" in the traditional sense of an HDD. There is no mechanical read head searching back and forth over the disk to locate data. Certainly it takes time for an SSD to access data but it is an order of magnitude faster than HDDs. And there is no rotational latency whatsoever. It is just as fast for an SSD to load contiguous data as fragmented data.

Since file fragmentation does not matter, it is important to **not** run disk defragmenting programs on SSDs. In fact, because of wear-leveling, it is actually **impossible** to defragment an SSD. Defragmenting and SSD serves to only prematurely wear the drive by forcing it to write data that doesn't need to be written otherwise.

Compacting

It has been mentioned many times in the past, and in this document, to not defragment SSDs. Somewhat contradictorily, database compacts can still be important with SSDs. The issue is the way in which 4D optimizes access to the file system. If the content has spatial locality it is still advantageous.

When 4D reads from the data file, it does so in blocks (as opposed to say reading a single record). The idea is that, for example, if 4D needs one record it might need the other records that are "nearby" as well. Think about this in the context of a sequential search to make it clearer.

If the data file is compacted, sequential records are likely to be located in the same block. If the data file is fragmented, sequential records might be located in different blocks.

What's the difference? **The quantity of block operations**. A compacted data file potentially reduces the number of operations 4D has to do when accessing the data file. Here is an example based on real-world experience:

- Large database – over 50GB, 300 clients, 100,000 new records per day, not regularly compacted
- MSC Verify: 2 hours
- MSC Compact: 5 hours
- **MSC Verify of compacted data file: 30 minutes**
- **MSC Compact again: 2 hours**

This example is extreme, but it illustrates that regular database compacting is still recommended, even with SSDs. The interval of such compacting is of course dependent on the database usage. A read-only Web database, for example, may never need to be compacted because the data file is never (or rarely) being modified.

Why the caution, why not just always compact? Any database maintenance must be balanced with the best practice of avoiding unnecessary writes to SSD drives. A compact is, essentially, the creation of a full copy of the database. There's no reason to do this unless the compact would actually be beneficial.

Note that 4D v12 introduces a new command, GET TABLE FRAGMENTATION. This command can be used to measure fragmentation in the data file and thus make an evaluation of whether or not compacting might be necessary.

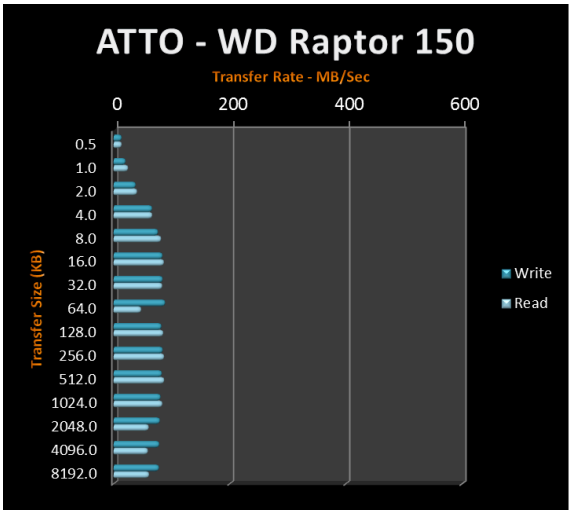
Buying the Right SSD

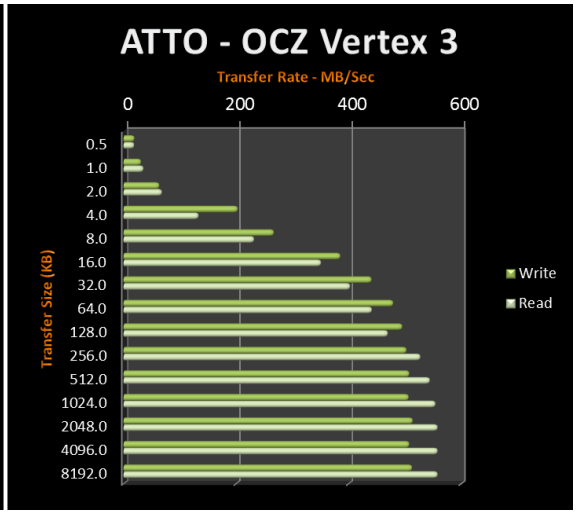
The most important features for any SSD are:

- TRIM support.
  - Or at least the ability to "clean" the drive in a proprietary fashion. You want the ability to restore peak performance without wiping the whole drive.
- Mean Time Before Failure (MTBF).
  - Literally the time it will take the drive to fail, usually expressed in hours.
  - A sub-feature to consider here also is the warranty.
- SATA 6GB/s connectivity.
  - I will explain this last point…

The second generation of mainstream SSD drives introduced support for the SATA 3 standard (first generation drives were SATA 2). The maximum bandwidth with SATA 2 is 3 Gb/s; SATA 3 is 6 Gb/s. For HDDs this is irrelevant. Most HDDs simply cannot saturate even a SATA 1 connection (1.5 Gb/s). Second generation SSDs on the other hand have already hit the cap for SATA 3. Here is a benchmark example using the ATTO benchmark:

This benchmark writes and reads data to/from the drive in incrementally increasing sizes (512K up to 8 MB), extrapolating the bandwidth for each size. The queue-depth in this test was 4.





The hard drive in this case was a Western Digital Velociraptor; this is a 10,000 RPM high-performance enthusiast hard drive. The SSD was an OCZ Vertex 3, which is a second generation mainstream SSD supporting the SATA 3 standard.

As can be seen, the HDD was unable to break even a 100 MB/s transfer rate while the SSD peaks at nearly 600 MB/s.

Note: *Keep in mind that in order to enjoy SATA 3 performance the motherboard drive controller must also support SATA 3.*

The two most important SSD specs for 4D are:

- Random read and write speed.
  - The drive manufacturers general publish this metric.
- High Queue-Depth performance.
  - The best way to check this is to research benchmarks for the drive.

Random access is the bread and butter of pretty much any SSD so it is not necessarily the defining benchmark/feature. In point of fact Intel's first-generation SSDs claimed very humble random

write speeds. However at the time only Intel provided consumer-level drives with what should be considered acceptable controller performance. Specifically, Intel destroyed the competition when it came to queue-depth optimizations. Even though Intel's read/write speeds might not have measured up to other drives on paper, in testing Intel always outperformed the competition, in some cases exponentially, when the queue-depth increased. When reviewing SSD benchmarks, be sure to check and see if the author is including queue-depth as a factor.

This is not meant to be an outright endorsement of Intel drives. Second-generation SSDs based on the Sandforce controllers, for example, perform very well at higher queue-depths and are even beating Intel's second-generation drives.

In the end the goal of this section is not to convince you to buy a particular SSD but, rather, be educated about making the decision of which one to buy.

Using Your SSD

Here are some important tips about using an SSD, both in general and in the context of 4D.

- Do not defragment the SSD, ever. It provides no performance benefit and excessively wears the drive.
- Consider disabling operating system indexing. SSD performance is sufficient that file indexes may not be needed and indexing the drive again wears it unnecessarily.
- Keep backups on a different drive. This should be done anyway, for security's sake but it avoids unnecessary writing to the SSD.

When to expect good results

It's important and interesting to monitor disk I/O when considering using an SSD. For example you can use Resource Monitor on Windows to watch disk activity. If the throughput is ~50 MB/s or less the SSD probably won't be faster than an HDD for sequential operations. As you approach 100 MB/s the SSD will start to pull away. It all gets back to where the bottleneck really is, which is often a function of database design.

On the other hand, for random operations, measuring throughput is not the way to go. Here it is better to benchmark your database to check the performance difference.

Cache flushing is an important, critical difference. Objects in the cache are usually not contiguous on disk. SSDs do not care. At 4D, Inc. we have seen cache flushes go from 15-30 seconds to being imperceptible. Also keep in mind cache flushing can affect user actions so the faster it happens, the happier the users will be.

When considering an SSD it is important to be realistic. SSDs are not a magic bullet. Although SSDs are a lot faster than magnetic disks, they are still nowhere near CPU/Cache/RAM speed. The disk is still going to be a bottleneck when compared to other resources (network, RAM, CPU). SSDs will perform much better for sure but definitely will not eliminate disk access as a bottleneck, so all of the things you may be doing already to avoid disk access are still valid.

Optimal Operating System Settings

There's really nothing specific for 4D but lots of resources online about tweaking the OS for using SSDs.

In general:

- Install the OS to SSD, not just the database!
- Disable OS indexing.
- Disable OS caching.
- Disable OS defragmentation.

Here are some specific resources:

- Mac OS X SSD Tweaks: http://blogs.nullvision.com/?p=357
- Windows SSD Tweaker: http://elpamsoft.com/Downloads.aspx?Name=SSD%20Tweaker

Benchmarks

Keep in mind that any benchmark is a "synthetic" test. The numbers here may not be representative of every real-world case. But they are compelling, nonetheless.

Test Configuration

Here are some important details about the tests:

- All tests were performed with Windows 7 (for TRIM support).
- The operating system was **not** installed on any of the SSDs. This was done to give as fair a comparison as possible between all three drives. In fact, if you put the OS on the SSD as well you will see even faster results.
- Tests were performed with a "cold" cache, unless otherwise noted. Once the cache is filled, the data is in RAM so the disk is a non-issue.
- All code was executed on the server, in order to remove network communication as a contributing factor.

Machine specs

This is the test machine configuration:

Processor	Intel Core 2 Quad Q6600, 2.40 GHz
RAM	8192 MB
OS	Windows 7 Home Premium (64-bit)
4D Version	4D Server v11 SQL

Drives tested

The following drives were used in the testing for this session (the testing abbreviations are used later in the results section):

Drive	Testing Abbreviation
Western Digital Raptor 150 GB	HDD
Intel X25-M 80 GB	Intel
OCZ Colossus 120 GB	OCZ

Here are some details about the drives:

Western Digital Raptor

The Raptor is a 10,000 RPM "enthusiast" hard drive, featuring a large cache and excellent HDD performance including fast seek times. This fast drive was used in an attempt to create a "best case" mainstream scenario for the HDD testing. Keep in mind that many HDDs will be quite a bit slower than the Raptor.

Intel X25-M

As already mentioned, the Intel SSD drives represented the best example of first-generation mainstream-level SSDs. The choice to test this drive was a "no-brainer".

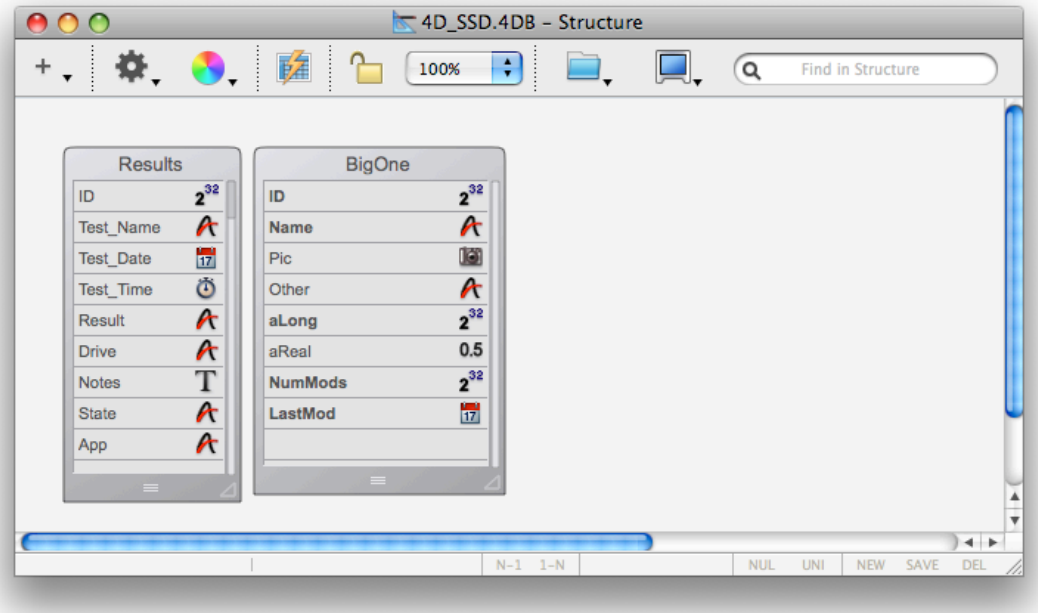
OCZ Colossus

This OCZ drive was chosen for a few interesting reasons:

- The Colossus does not support TRIM. It instead has a unique internal RAID configuration, pairing two individual SSDs. The proprietary RAID controller features a garbage collection technique. Perhaps the primary difference is that the OCZ garbage collector is not prompted by the OS as with TRIM. It runs on its own in the background.
  - Make note of the fact that the OCZ garbage collector needs the drive to be idle in order to do its job.
- The Colossus comes in a 3.5" form factor. This can be a compelling feature when installing an SSD into most any system because typically an adapter is needed to make them fit correctly (most SSDs are 2.5" or 1.8"). The Colossus is a drop-in replacement for any standard 3.5" SATA HDD.
  - This situation is more compelling on modern Mac Pros and many rack mount servers because the SATA connections are hard mounted, not flexible cables. The drive (or adapter) must conform to the 3.5" form factor to align correctly.
- The Colossus fairs well against the Intel drives, especially in sequential access. Where it falters though is with increased queue depth. So it was chosen for the purposes of comparison.

Test Database

The test database consists of a single table with 6 million records and another table to store test results. Here is the structure:



The table [BigOne] contains the 6,000,000 records. The data in this table is randomized with the exception of the picture field. All records contain a copy of the same 5K picture. This results in a 32 GB data file and a 413 MB data index file.

Note that in the structure some fields are indexed in order to present a more realistic scenario. That is to say during the tests indexes were not dropped nor added. They were left as-is for 4D to update and maintain.

Note: *There is no database included with this session simply because the database is "not interesting" and the data file is quite large (~32 GB). However the code for each benchmark is included below.*

Test 1: Modify 500,000 Records

This test is designed to simulate random access for the purposes of changing data. The test simply performs a loop and within this loop a random record is queried for modification. In this test 500,000 out of 6,000,000 records were modified. Keep in mind that indexed fields were being modified as well so 4D has to maintain the indexes.

Test Code

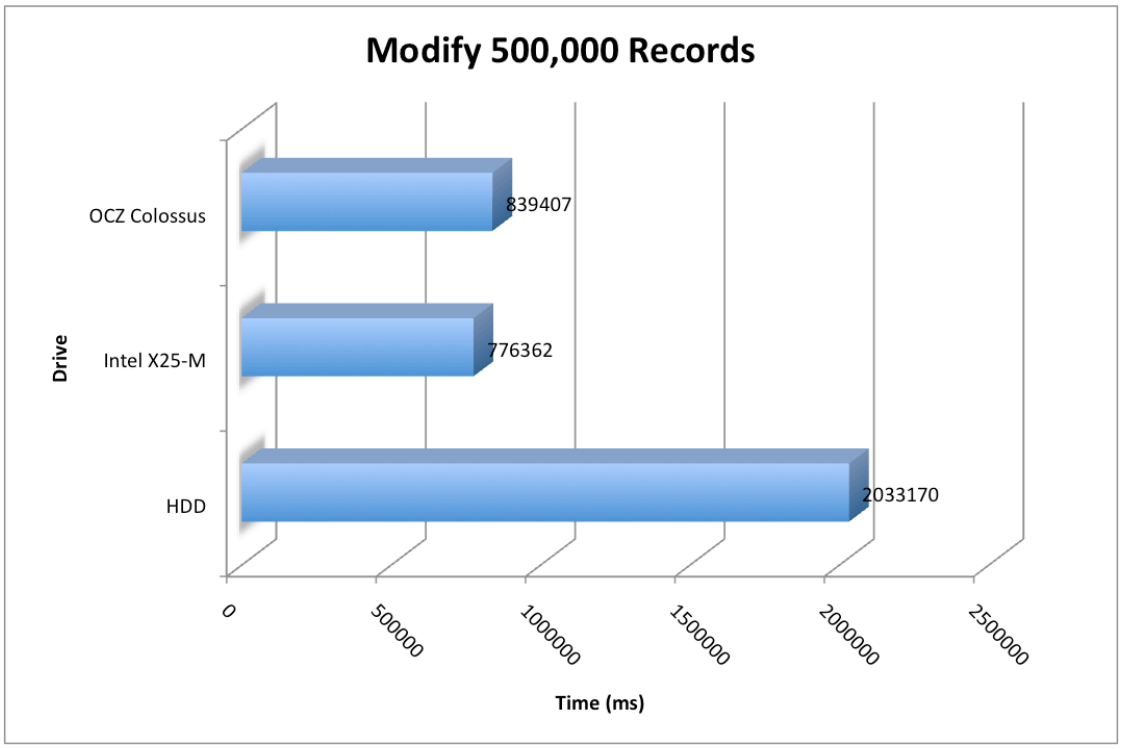
Here is the code:

```
C_LONGINT($numberOfMods_l)
C_LONGINT($rangeOfRecords_l)
C_LONGINT($i;$id)

  ` The number of records to modify.
$numberOfMods_l:=500000
  ` The range of record ID's to use.
$rangeOfRecords_l:=6000000

For ($i;1;$numberOfMods_l)
        $id:=(Random%($rangeOfRecords_l))+1
        QUERY([BigOne];[BigOne]ID=$id)
        [BigOne]Name:="name"+String(Random)+"x"
        [BigOne]aLong:=Random
        [BigOne]aReal:=Random
        [BigOne]NumMods:=[BigOne]NumMods+1
        [BigOne]LastMod:=Current date(*)
        SAVE RECORD([BigOne])
End for
```

Results

This first example shows a marked performance improvement using the SSDs although not as dramatic as one might expect.

(All times are in milliseconds).



Still, a 50-60% improvement is nothing to scoff at.

This is a processor intensive test due to the loop as well as code to create the data values so it is not necessarily disk-bound. But what was interesting about this test was, when testing the HDD, the processor activity tended to spike and recede while the disk was being accessed whereas during the SSD tests the processor (1 core since this is 4D Langauge) tended to stay maxed. I.e. a faster drive allows 4D to, in turn, use more of the CPU.

Here we can see the Intel SSD outperforming the OCZ by over a minute as well.

Test 2: Goto 100,000 Random Records

This test is designed to simulate random access as cleanly as possible. In contrast to the previous test, this test is not processor intensive. It is almost entirely bound by disk speed.

The test simply performs a loop and within this loop a random record is loaded using GOTO RECORD. In this test 100,000 out of 6,000,000 records are visited.

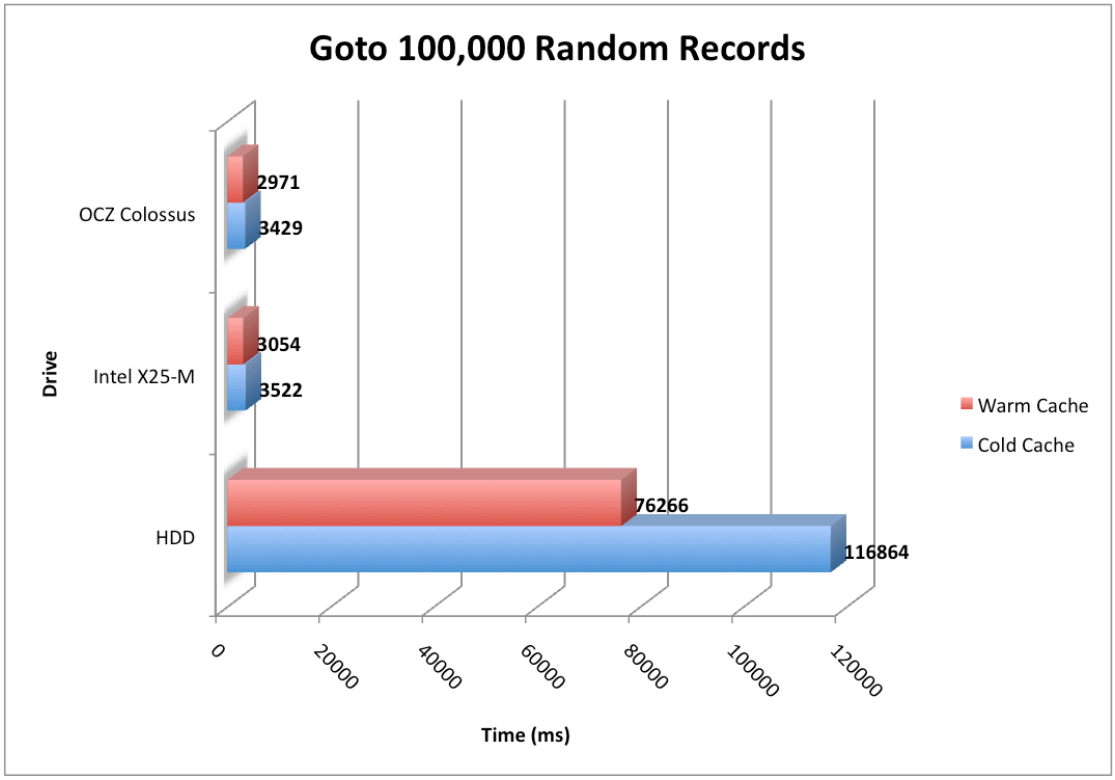
Test Code

Here is the code:

```
C_LONGINT($numberOfMods_l)
C_LONGINT($rangeOfRecords_l)
C_LONGINT($i;$id)

  ` The number of records to go to.
$numRecsToGoto_l:=100000
  ` The range of record ID's to use.
$rangeOfRecords_l:=6000000

For ($i;1;$numRecsToGoto_l)
      GOTO RECORD([BigOne];(Random*Random)%$rangeOfRecords_l)
End for
```

Results

Here we see the kind of shocking results that are possible with an SSD:



The HDD really is not on the same level at all. The SSDs simply destroy the HDD performance.

The cold cache results came from a fresh launch of the database to run this test. The test was then repeated just to compare a more real-world situation where the cache would be warmed. In any case the HDD performance is simply laughable.

Test 3: Sequential Sort on 6,000,000 Records

This test simply performs a sequential sort on the [BigOne] table.

Test Code

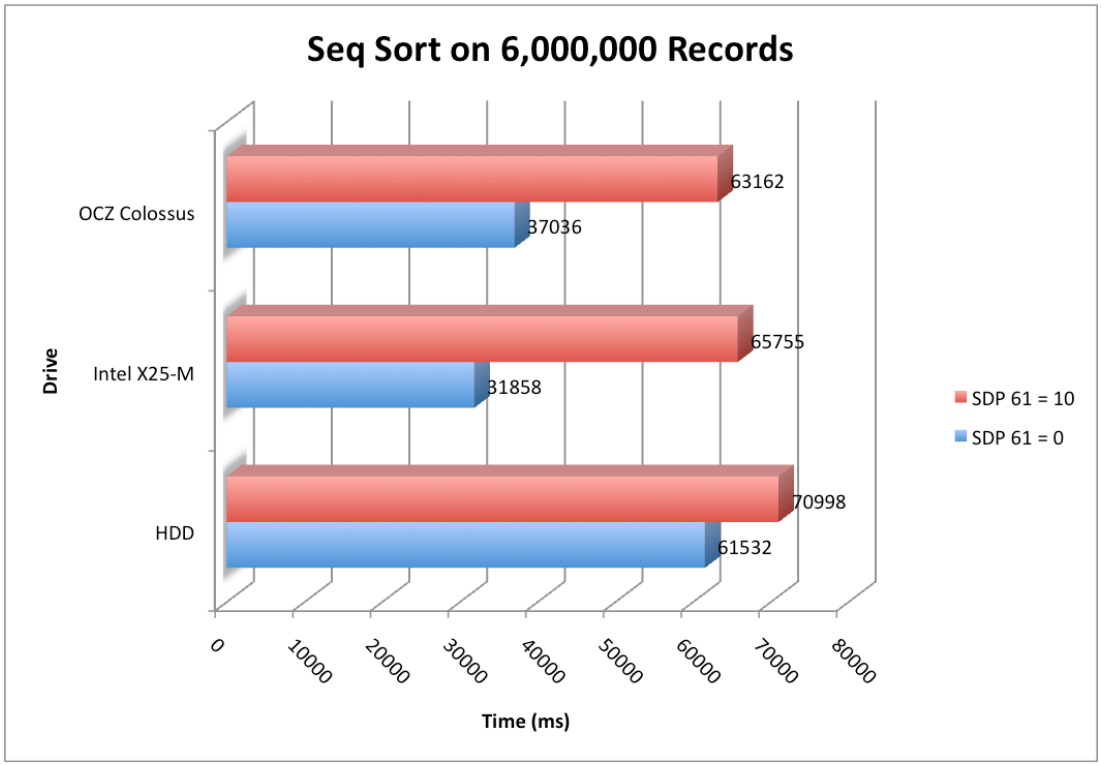
Here is the code:

```
ORDER BY([BigOne]aReal)
```

Note that aReal is not indexed.

Results

Here is a more sobering test in the sense that SSDs are not “magic”.



The test was run twice with the following difference:

As of 4D v11 SQL Release 5 (11.5) there is a new selector for SET DATABASE PARAMETER (SDP) that limits the amount of cache memory that 4D will use in order to perform a sequential sort (sequential sorts occur in the cache, by default). This is selector 61. The default is 0 which means "unlimited" – bounded by the cache size of course. In this test the default setting is compared to a value of 10, which means 4D will only use 10 MB of cache in order to perform the sort. When 4D hits this limit, it flushes the current buffer to a file on disk and keeps working.

With the default value you can see that the SSDs are still the clear winners. But when the memory that 4D uses for sorting is limited the results are not so dramatic. Why is this? The root cause is sequential disk I/O; in this test 4D is writing many 10MB files to disk with is largely a sequential operation. But aren't SSDs faster?

SATA 2 SSDs are not **inherently** faster for sequential disk operations. Some are, some are not. The Intel drives have relatively poor sequential write performance, for example (relative to other SSDs, but still fast). Second generation SSDs have greatly improved in this area through increased parallelism and SATA 3 support.

The second issue is perhaps more subtle: as mentioned before all SATA 2 drives on most current systems will top out at approximately 250-280MB/sec of sequential throughput. This is a limitation of SATA 2 connections. We need SATA 3 connections to break this barrier.

- There are both SSDs and HDDs capable of hitting this barrier currently, but SATA 3 SSDs have already blown past it.

Conclusion

There will probably never be an "official" endorsement of SSDs by 4D, Inc. (nor any other specific piece of hardware or OS). However in this situation the performance differential is so important we would be remiss if we did not inform our customers of this.

As with any new piece of hardware, your mileage may vary. Any new hardware configuration should be thoroughly tested. Consider SSDs as another tool in the arsenal of technology available to build the best, fastest possible applications for your customers.

Best Practices

- Memory
  - 32-bit 4D can use 4 GiB of virtual memory.
  - 64-bit 4D Server v12 can use 8 TB of virtual memory.
  - Measure application memory with the "Used Virtual Memory" metric from the GET CACHE STATISTICS command.
  - To maximize the amount of memory 4D can use:
    - Use a 64-bit OS
    - For 32-bit 4D install at least 6 GB of RAM.
    - For 64-bit 4D Server v12 balance the total RAM against the footprint of the server.
  - Use the same rules above to maximize the amount cache available to 4D.
    - Be sure to tune the cache size appropriately for both the hardware and database.
  - The cache flush interval can now be expressed in seconds. The cache manager in 4D is most efficient when there are fewer changes to flush so databases with a high rate of data changes may benefit from a decreased flush interval.
  - Do not use adaptive cache unless absolutely necessary.
- Solid-State Drives
  - Use them!
  - Important Features
    - TRIM Support
    - High random I/O performance
    - High queue-depth performance
    - SATA 3 Support
  - Usage
    - Do NOT defragment SSDs
    - Consider disabling drive indexing
    - Keep backups on a separate drive
    - Install the OS to an SSD
    - Consider disabling OS caching
    - There are SSD tools to automatically manage the OS settings.
  - Expectations
    - Disk I/O that is less than 50 MB/sec on a fast HDD may not benefit from SSD
    - However random I/O will always be faster
    - Cache flush durations should be greatly reduced
    - SSDs are fast, but not as fast as RAM; adjust expectations accordingly

4D AND DESKTOP VIRTUALIZATION

These are some recommendations regarding using 4D with desktop virtualization (i.e. "thin clients"). Unless otherwise noted, these items apply to both Microsoft Terminal Services and Cytrix.

RDPCLIP

There is a compatibility issue between 4D Server and clipboard sharing feature in Microsoft Remote Desktop. There are two ways to resolve this:

- Abort the "rdpclip.exe" process if 4D Server is unresponsive (this process will typically be consuming 99% CPU in this case).
- Permanently disable the "Clipboard mapping" feature in the Terminal Services Configuration.
  - It can also be disabled from the client in the Remote Desktop Connection configuration.

Printer Driver

If no other printer is installed, by default 4D uses the System printer driver. This driver may be the cause of problems if it has not been updated or if it was not installed with the same user that is running 4D. To avoid

these problems, it is recommended to install a “dummy” local printer (HP generic printer driver for example) and to set it as the default printer. This of course assumes no local printer is already installed.

Client Load

We recommend no more than 15 to 20 4D clients per thin client Server. These should be dedicated machines, not virtual instances.

Things to Avoid

- Design Mode
  - Opening the popup for font or font size in the object font property.
  - Opening the popup for variable type in the property list.
  - Drawing or moving objects on a form.
- Application Mode
  - Resizing rectangles in 4D Chart.
- We do not recommend the use of the SET TIMER command in TSE/Citrix environment.

Database Settings

Set all the CPU priorities to the middle setting. Make sure these values are not unexpectedly altered via the "SET DATABASE PARAMETER" command.

IN DEPTH PROBLEM ANALYSIS

A customer's complaint that "sometimes the server is very slow" or "sometimes the server does not work as expected" is the issue that's most difficult to handle, especially in large environments where you don't have a chance to ask the customer, "What did you do differently?"

By using logs and knowing how to read and analyze them, we can find and understand many of these issues. Even better, using the tools to watch the server behavior in normal times allows us to quickly identify possible issues and resolve them, avoiding potential issues, even before customers notice them.

Starting with Version 11 4D Server provides many log files, allowing detailed off-site analyzes helping to understand Server behavior and problems. This part of the class focus on four tools for this purpose.

4D CACHE LOG ANALYZER

While previous 4D versions uses the cache only to improve speed (avoiding hard disk access to speed up object access), 4D v11 SQL and newer also use the cache to store current selections, transactions, sets, internal objects needs for joins or sorting and much more. The cache manager was totally rewritten with the release of 4D v11 SQL.

The size of the cache – and the available space to store objects – has major impact for the speed of an application. Extremely low memory situation could lead to a kind of server hangs (=freezes) or even crashes.

As the scalability of 4D v11 SQL made it possible to use 4D in larger installations with 1000 processes and more handled by the server, the requirements for the cache manager to speed up freeing unused objects in low memory situations and at the same time to handle large cache size with 20 GB or more in 64 bit mode, the cache manager was drastically improved with 4D v11 SQL Release 8 and 4D v12.1. All statements and examples in this chapter are about this new cache manager.

In a perfect world the available memory on a Server would be endless (or practically, larger than needed) and we would not need to talk about Cache Log Analyses or how to understand what's going on in the cache. In the real world we have typically two situations:

- Existing server with too less memory. Understanding what's going on, optimize the settings and – if required – getting proves that a memory enhancement is a good investment
- Providing recommendations for a new server. This requires understanding the need of a specific application in an environment such as 20, 50 or 75 users. Creating profiles of existing customers, such as 5, 10, 40 users allows to interpolate the data to get a clear vision how the required memory increases when the amount of users is increased.

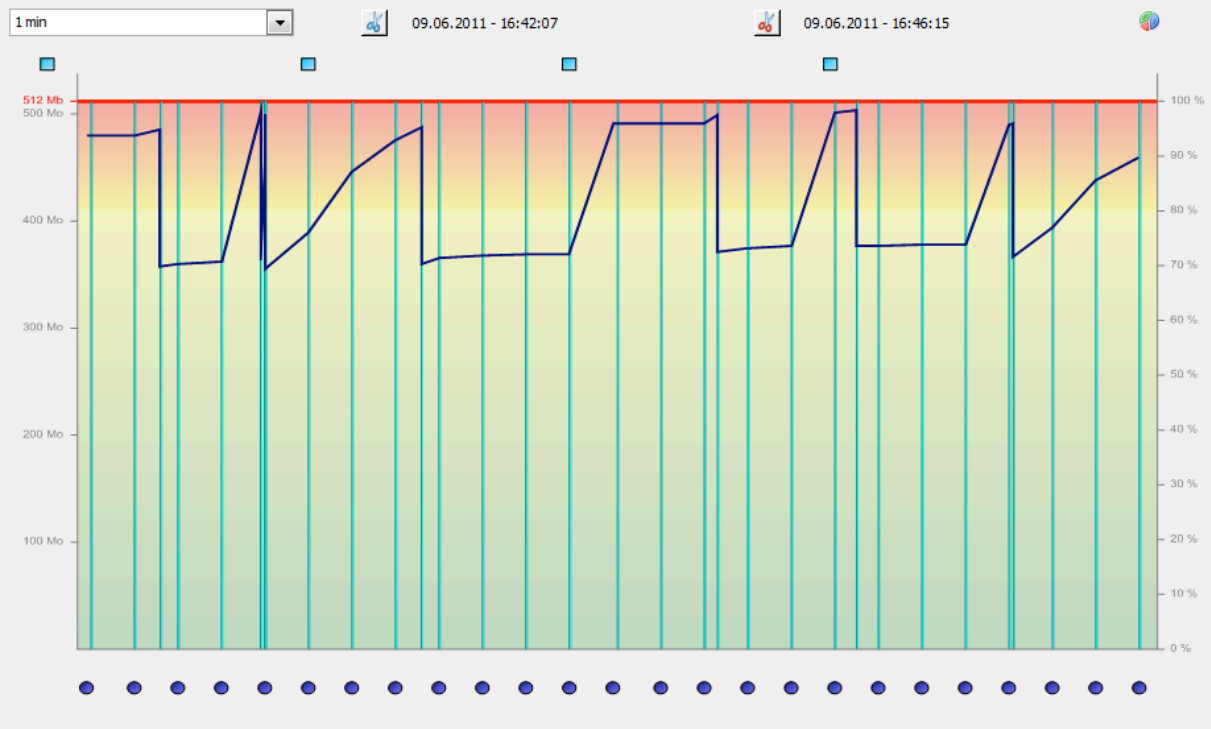
Of course there could be a 3rd situation, with a server with enough memory where all simply works well. But in that case you would not be in this class…

CACHE LOG RECORDER AND ANALYZE

4D provides a command named GET CACHE STATISTIC, which returns a huge amount of information's about the cache content. The command could be executed regularly, the result be logged – and analyzed. This session is about a component named "4D Cache Log Recorder", handling the logging part and an application "4D Cache Log Analyzer" to read these logs and display the data graphically.

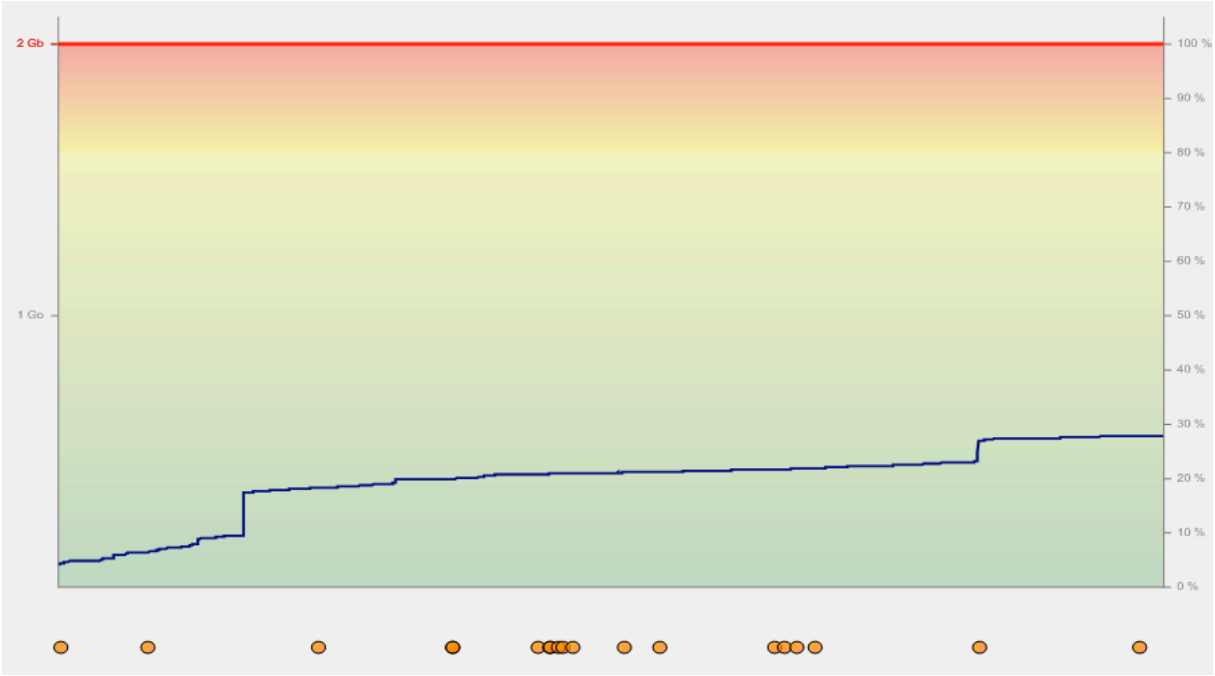
As an example here a productive server with v12.2 and 10 users/210 process on a computer with 512 MB cache, recorded for one working day:

Easy to see that it regularly requires more cache and purges about 100 MB, reload, purges and so on. Zooming closer to such a peak:



The blue dots below shows the regularly flush buffers process from 4D, here set to 20 seconds. The vertical lines in between shows additional flushes, either triggered by language or because data needed to be written before it could be purged. The chart shows that often it needed to purge and reload data several times per minute, which explained why the server was so slow.

Here the same customer, same installation, two days later on a 64 bit Server with 2 GB Cache assigned to 4D Server:



After same days of work the cache is usually at 50-60%, so the server could run with a less RAM – or saying it different, they can handle more users/processes with this computer without the need of memory increase.

4D CACHE LOG RECORDER

I. Introduction

The component named 4D Cache Log Recorder is a tool allowing to record information related to the used cache memory during the use of a 4D database. The recorded information are stored in files to be analyzed using 4D Cache Log Analyzer.

II. Installation

The component, as all other 4D component, is to be installed simply by copying the package (Mac OS X) or the directory (Windows) named 4DCacheLogRecorder.4dbase inside the Components folder next to the your database structure file (*.4DB or *.4DC). The minimum required version to run the component is 4D v11 SQL Release 5 (11.5). Beside the v11 component there is a build for v12, which also supports v13.

III. Opening the interface

Once the component is installed and the database is started, you can display the component interface by calling the method CLR_Interface. You will then get the following dialog :

IV. Setting the options

The above dialog provides the default settings. You can set your own options.

IV.1. Recording every

This field sets the interval (in minutes) between two information recordings. Increase this parameter if you want to record information on several days. The default value is 1 minute. These data are needed for “Statistics Details”. Importing logs for a full week created with 1 minute interval needs on a modern computer about 4-5 hours, which can be speed up by setting the interval to 5 or 10 minutes, but provides less details, so it could make sense to run the log with high details even importing logs need more time.

IV.2. Flush cache

This popup menu provides three options.

- None (by default) : lets the database record the cache following the usual way.
- Simple : records the cache based on the previous option « Recording every ».
- With purge : does the same operation as the « simple » option but, when the cache is flushed on disk, the cache content is entirely purged. Only use this option if it is recommended by the 4D Technical Services because this option can substantially slow down your database.

IV.3. Memory info

This checkbox must be enabled in all cases, unless the 4D Technical Support asks to disable it.

IV.4. Cache statistics

This checkbox must be enabled in all cases, unless the 4D Technical Services asks to enable it.

IV.5. C++ Objects

By enabling this option, the performances will decrease. Use this option only if the 4D Technical Services asks to use it.

IV.6. Other objects

By enabling this option, the performances will decrease. Use this option only if the 4D Technical Services asks to use it.

IV.7. Blocks

By enabling this option, the performances will decrease. Use this option only if the 4D Technical Services asks to use it.

IV.8. Internal watching

This option is used in 4D 11.6 (and next versions) to enable the cache watching mechanism and generate log files like CacheLog_timestamp_n.txt.
This option should be always used with current versions, as it allows creating the “Statistics Overview”, which is much faster to display.

IV.9. Memory watching

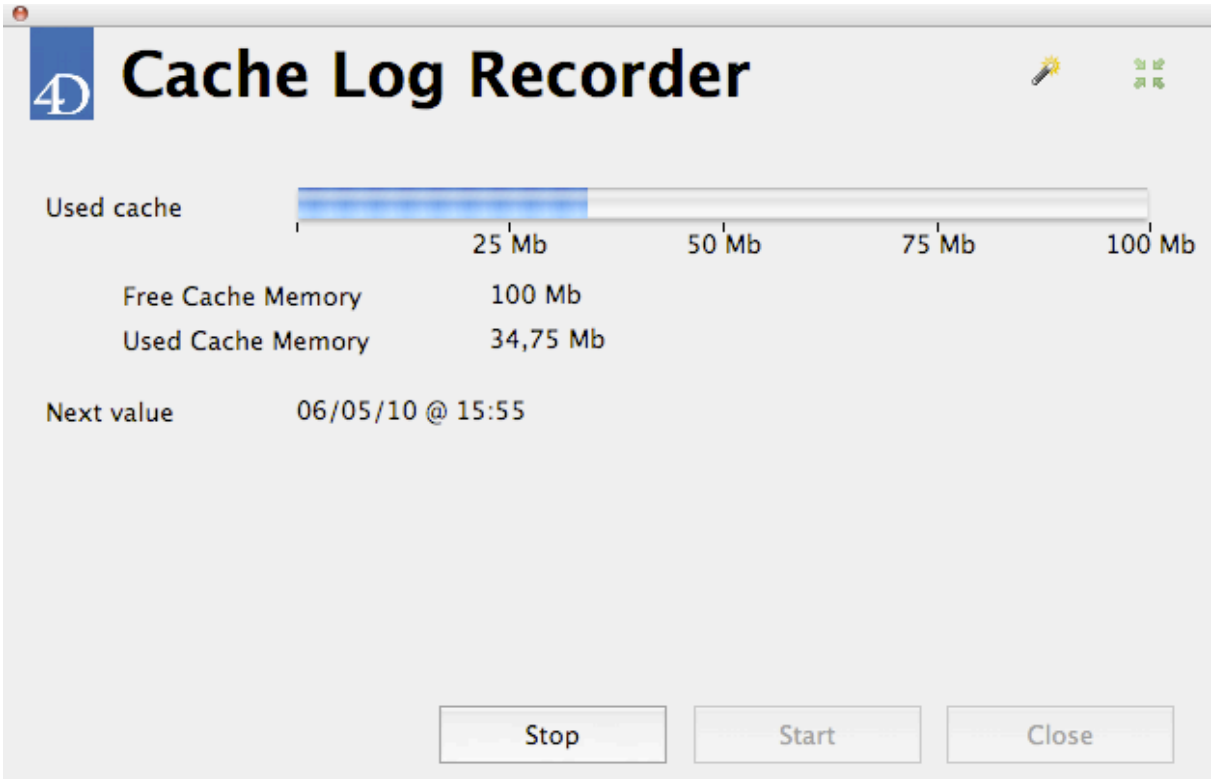
This option enables the memory watching mechanism every N steps for N steps, and generates log files like CacheMem_time. Use this option only if the 4D Technical Services asks to use it

V. Clearing the cache

The ***CLR_Launch_PeriodicalClearCache*** method allows you to purge the cache (all objects are deleted from the cache) periodically. When executing this method a dialog will be displayed to enter the interval time in minutes.
For example if you enter 1440 minutes, the cache will be purged every 24 hour.
To stop this process, execute the ***CLR_Stop_PeriodicalClearCache***.
Note that there can be a slowdown while the cache is increasing again after being purged.

VI. Starting the information recording

Once the options are set you can start the information recording by clicking the « Start » button. The dialog will be then:



VI.1. Used cache

The thermometer shows the used cache size compared to the available cache size. The numeric values related to the memory are returned in the two lines under the thermometer for « Free Cache Memory » and « Used Cache Memory ».

VI.2. Next value

This information displays the time of the next returned value.

VI.3. Stop

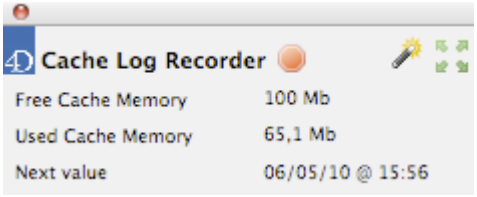
This button allows you to stop the recording.

VI.4. Magic wand icon

This icon allows you to manually clean the cache. The cache is then empty. Note that there can be a slowdown while the cache is increasing again after being purged.

VI.5. Resizing window icon

This icon allows you to minimize the dialog.



VII. Client / Server

In the case of the use in a client-server environment, the interface must be started with a 4D Remote. However during the information recording a process named ***CLR_logger*** is launched on the Server to execute the needed operations.

VIII. Faceless usage

To start the logging faceless use:

```
◊clr_logger:= Execute on server ("CLR_Logger";128*1024;"CLR_Logger";1;Record_Frequence;
Flush_cache;info_type1;info_type2;info_type3;info_type4;info_type5;b_Surveillance;b_Surveillance2
;Record_Every;Record_For;*)
```

with:
Record_Frequence =1 as default (minute)
Flush_cache = 1 (default, none), 2 = Flush Buffers every record frequency, 3 = Flush Buffers(*)
info_type1..5 = 0 or 1, enables checkboxes below Flush Cache
b_Surveillance, 0 or 1, activate SET DATABASE PARAMETER(65)
b_Surveillance2, 0 or 1, activates memory leak checking
Record_Every and Record_For: sets time in minute for memory leak checking as dialog, default Every=5 and For=1

Suggestion for usage:

```
$pr:= Execute on server ("CLR_Logger";128*1024;"CLR_Logger";1;1;1;1;1;0;0;0;1;0;5;1;*)
```

To stop the logging faceless execute on the server:

```
<>clr_logger_stop := true
CLR_Reactivate
```

Or on any client:

```
<>clr_logger_stop := true
VARIABLE TO VARIABLE(-1;<>clr_logger_stop;<>clr_logger_stop)
$pr:=Execute on Server("CLR_Reactivate";32*1024;"CLR_Reactivate")
```

IX.Debug logs

CLR_DebugLog_Interface

This component method displays a dialog where you can set the debug log selector and start/stop the debug logging on the Client side.

CLR_DebugLog_Interface_Server

This component method displays a dialog where you can set the debug log selector and start/stop the debug logging on the Server side.

LOG EVENT(4;”mylogmessage”)

This command allows to write comments directly into the Cache log, which often makes it easier to see which parts of your code is currently executed.

X.Sending the generated data

To send the generated data to the 4D Technical Services or transfer from your customer to your office, you have to find the folder named « Logs » next to your database structure (*.4DB or *.4DC). Inside this folder, there is a subfolder named CacheLog which contains the information we want.

Always compress the whole folder, never sent the text file directly, to make sure line endings are not modified from the email application. Inside the CacheLog folder, you will find already compressed files named Cache_Log_XXX.4DLog.

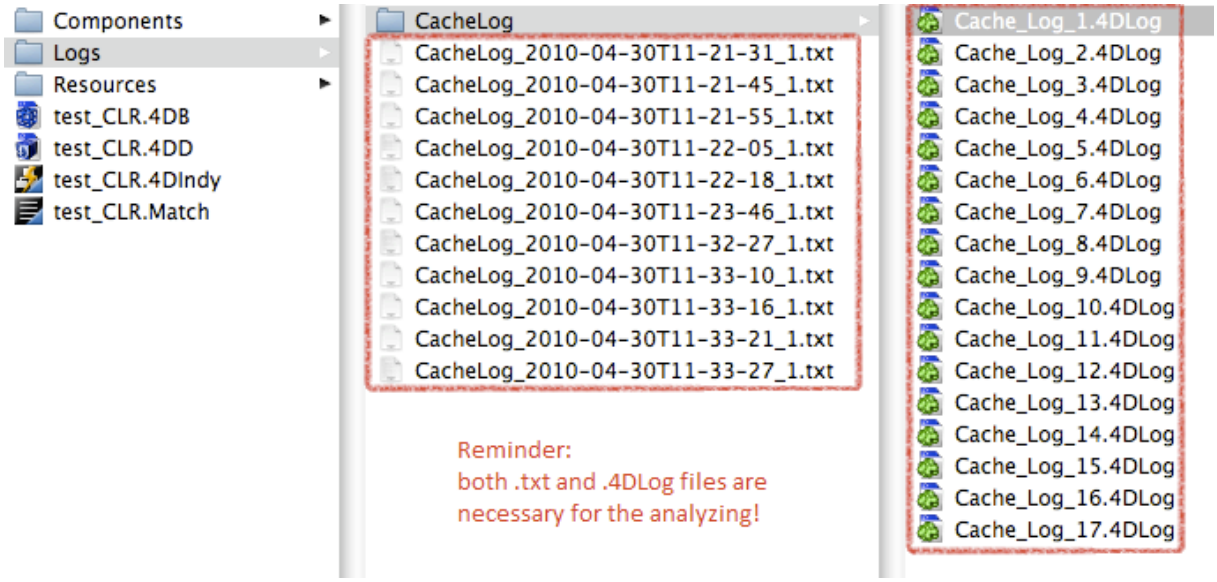
XI. Uninstalling the component

To uninstall the component from your database simply remove the component from the Components folder. Do not forget to delete the calls to the CLR_Interface component method in your code.

4D CACHE LOG ANALYZER

The application 4D Cache Log Analyzer, provided as v12 application, can read and analyze logs created with 4D Cache Log Recorder in v11, v12 or v13. It is recommended to create for each customer or even for each case a new datafile.

Use Menu File and select Import. Now select either the folder containing all files ending with .txt or the folder with the files ending with .4DLog:



4D creates for each recording job a .txt file (using a timestamp as name), plus extension _1.txt. As soon this file reaches 10 MB an additional file _2.txt is created, and so on.

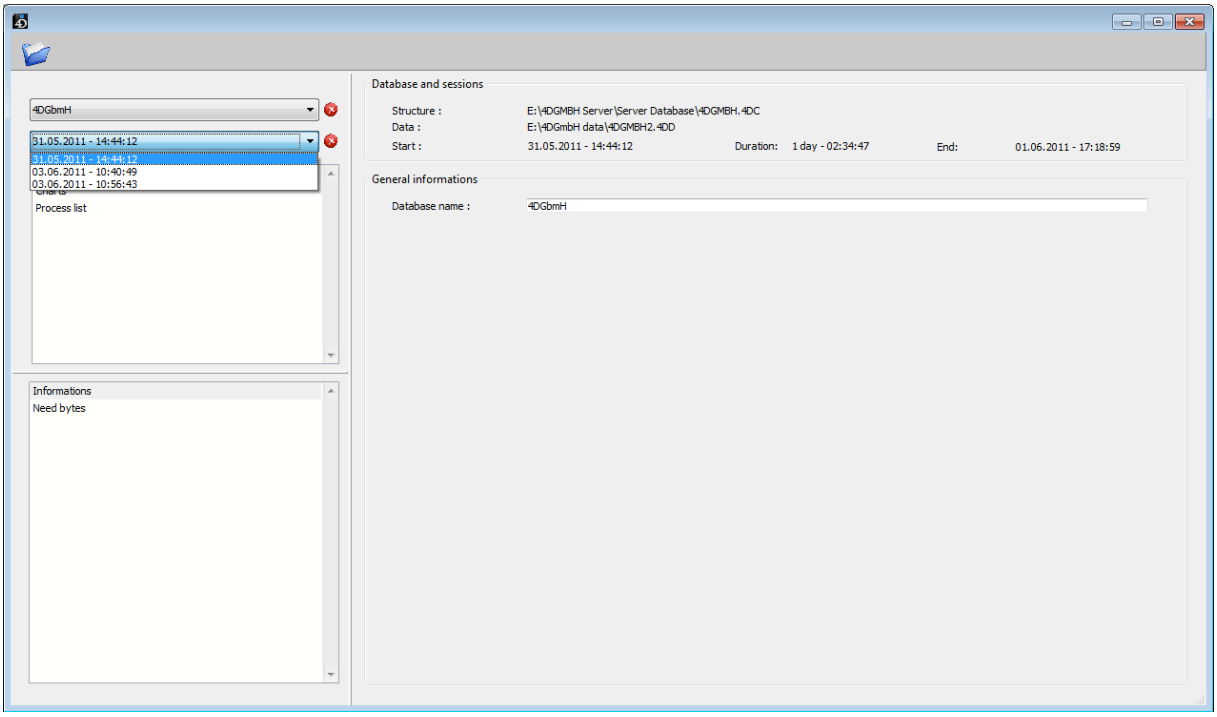
These files are relatively fast to import and a mandatory for the “Statistics Overview” Menu. Note that all created files, from _1.txt to _xx.txt are necessary, you must not skip some or use only the latest one.

Additionally for the recording component creates once per minute a file named “Cache_Log_xx.4DLog”. Running the recorder for several days will result in several thousand files, which could take some hours to import. While these files are not required for “Statistics Overview” they are mandatory for “Statistics Details”. As the import of these files could take a while, you could, if you need a fast overview, import only the .txt files and focus on the Overview dialog.

The import command first asks for a project name. This feature allows to group several imports, in case you want to collect different customers or servers in one datafile.

STATISTICS OVERVIEW

This dialog gives an overview about the cache situation.



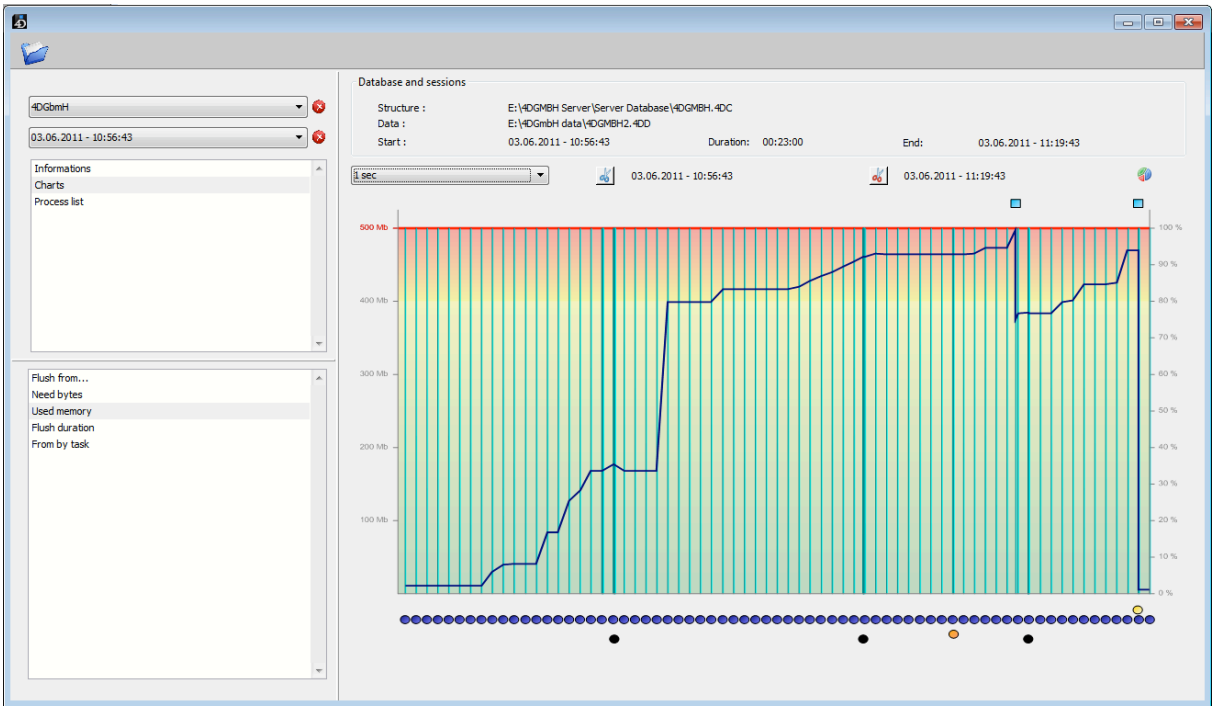
The popup menu upper left allows to select which log group you want to analyze. This is the name you entered when doing an import. Remember that you could create several data files, one per customer, to reduce the amount of records, which will speed up the analyzing.

The popup below shows the time frames, either contained in one log or for several imports with the same group name.

The listbox below defines the statistic group, with an additional listbox below to switch between different views.

USED MEMORY

Start by selecting “Charts” and “Used memory”, to get a first impression of the log:

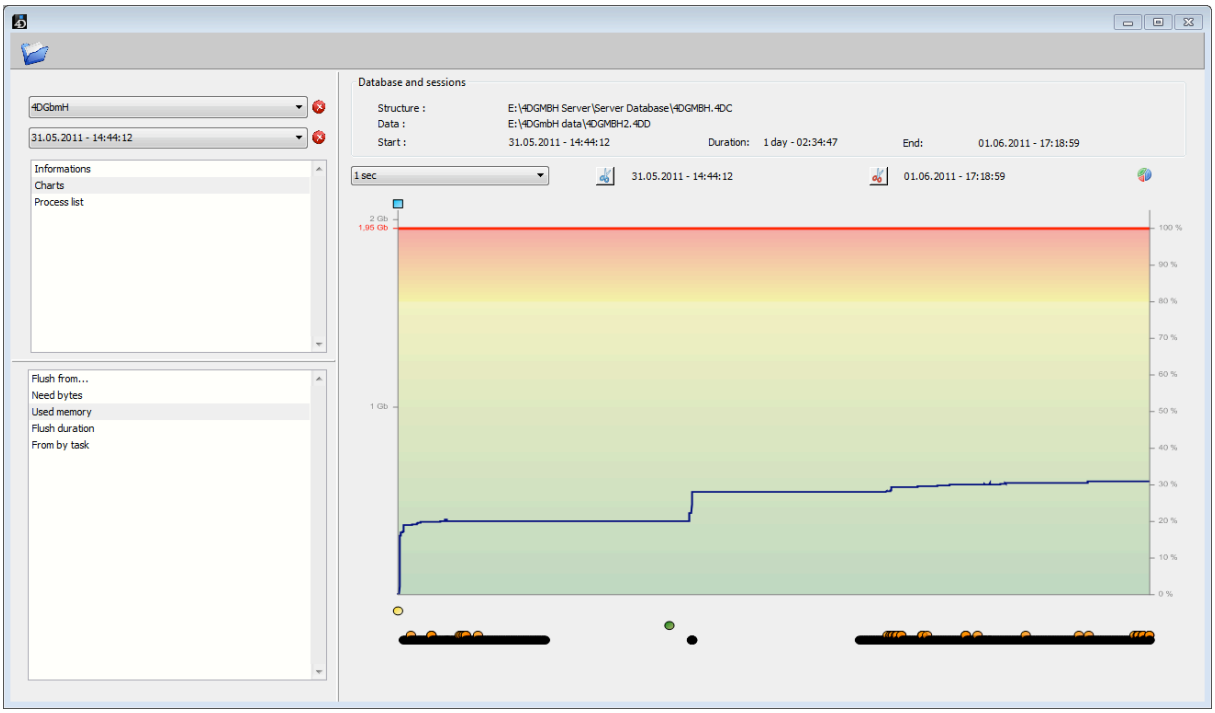


This example shows a short period, only 23 minutes, so it has a good scale. The informations on top display the start and end time as well as the duration.

The chart below displays the usage of the cache. The total in this example was 500 MB, which was once used (chart reached top), also notice the blue rectangle on the top. This rectangle says “need bytes”, the server needed more memory than available, so freed some Cache. The 2nd blue box was a faked memory request (issued by FLUSH BUFFERS(*)) to clear all cache.

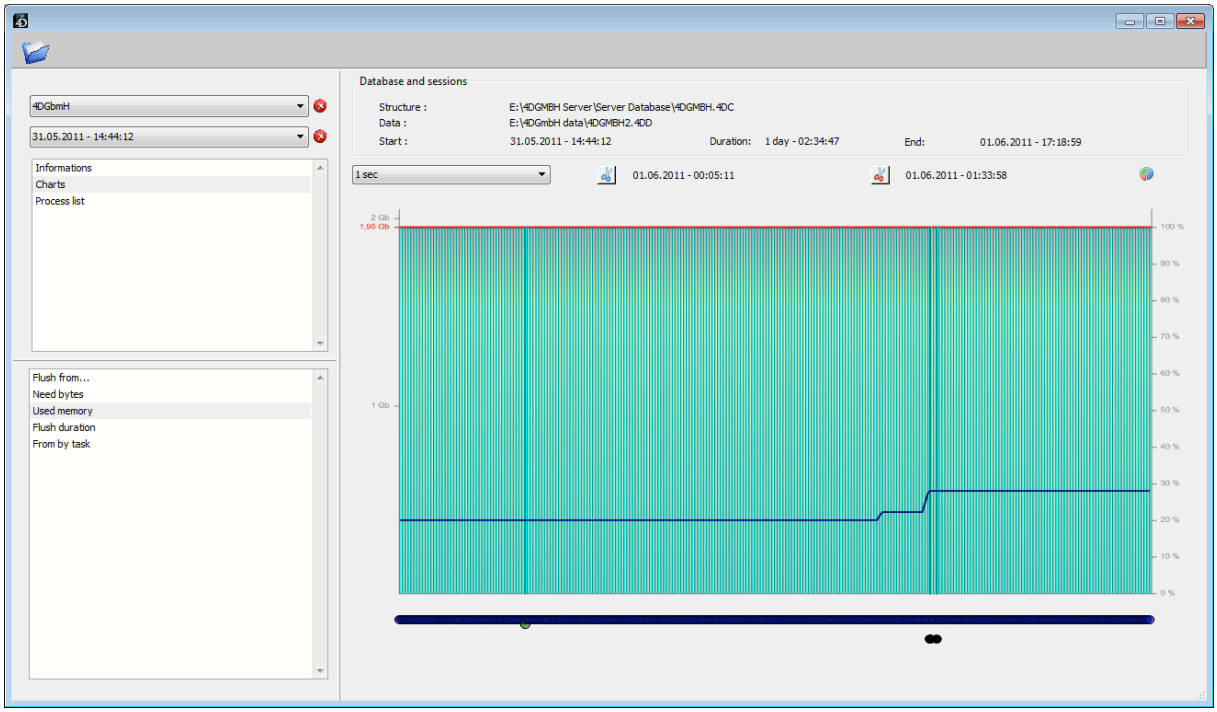
The vertical lines in the chart are displayed on larger scales like here and show when a FLUSH BUFFERS occurred, either per scheduler, by 4D Backup or by programing language. The blue bullets at the bottom showed a periodical flush from the scheduler (here every 20 seconds), the black bullets a FLUSH BUFFERS issued by a server procedure, the orange one from a client/remote process. The yellow one was FLUSH BUFFERS(*).

The vertical lines, as well as the bullets, are hidden if too many occurs in the given time frame, to avoid a big blue box and to increase speed. Following an example running more than a day. As you see the vertical lines are hidden, also the blue (20 seconds interval) circles.

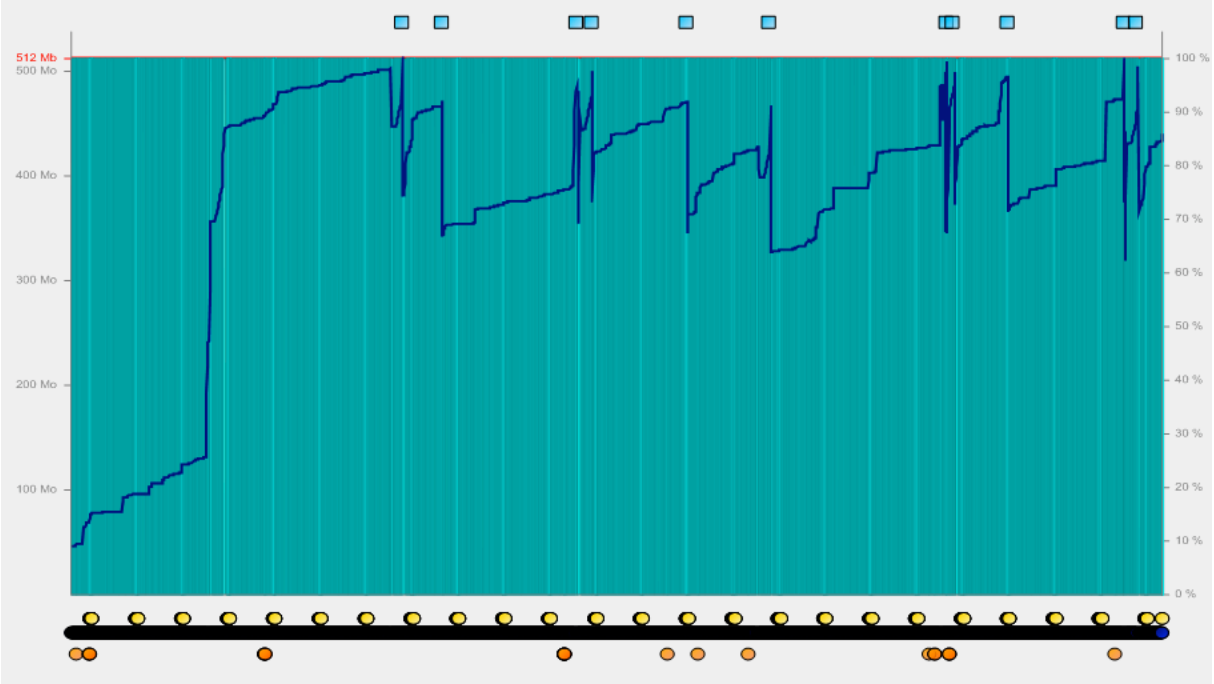


The black “rectangle” is a set of bullets, showing a server process calling FLUSH BUFFERS once per minute, except during the night.

The blue and red scissor allows to select a subset of the time frame, to have a larger scale. Click the blue one and then click somewhere in the chart to cut everything before. Do the same with the red to cut everything after:

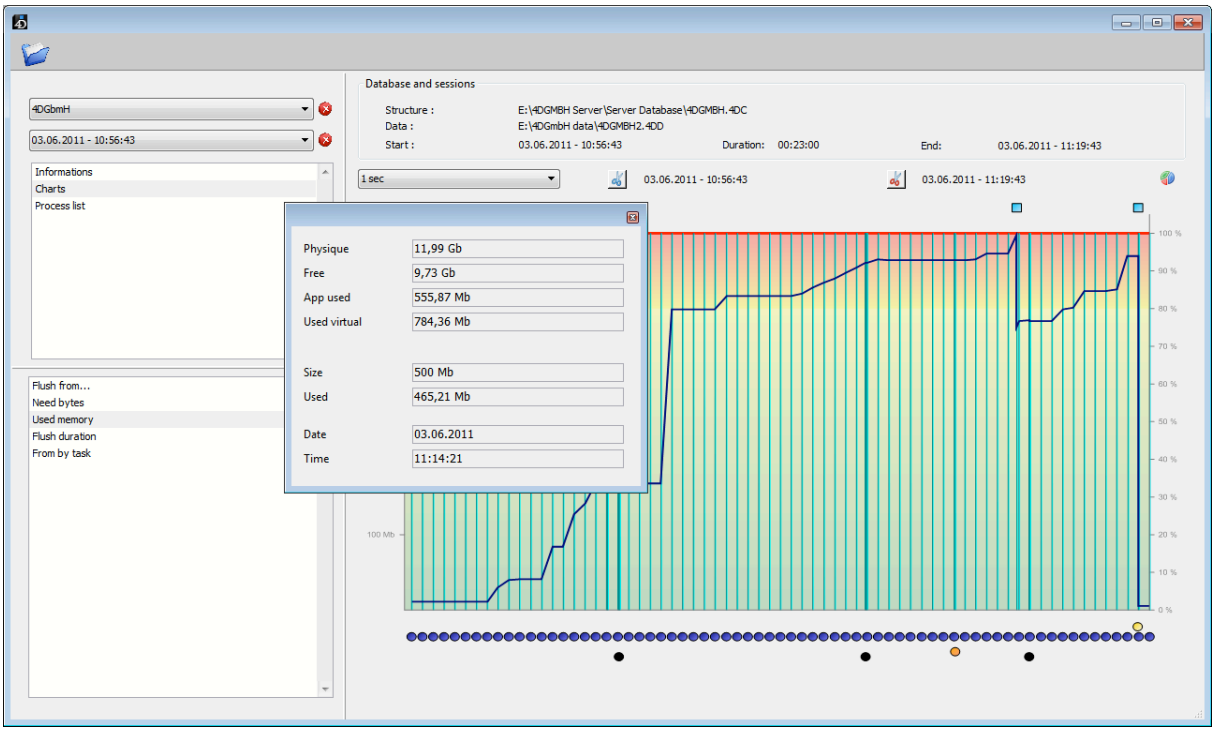


The first chart with 500 MB showed that the cache was – for this period – a little bit too less, the 2 GB is much more than needed. The following example shows a 512 MB cache which is much too less:



As these examples quickly showed this chart is a very important source of information's and usually the first to check with the 4D Log Analyzer. We will later discuss more examples.

Click the small pie icon on top to open a detail window. Now click any point in the chart to get details about this specific moment:

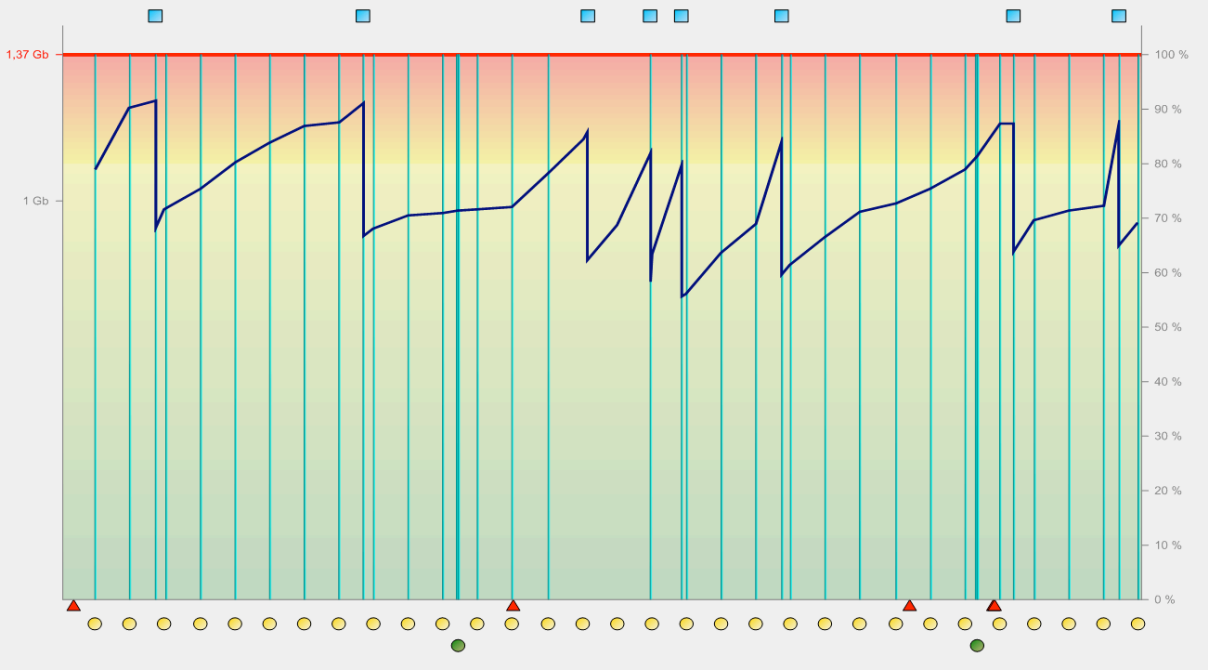


It shows that the server just used 465 MB of 500 MB Cache, while the computer itself still has 9.7 GB free.

LOG EVENT

Using the (undocumented) option 4 in LOG EVENT to comments into the Used memory chart:

```
LOG EVENT(4;"my comment")
```

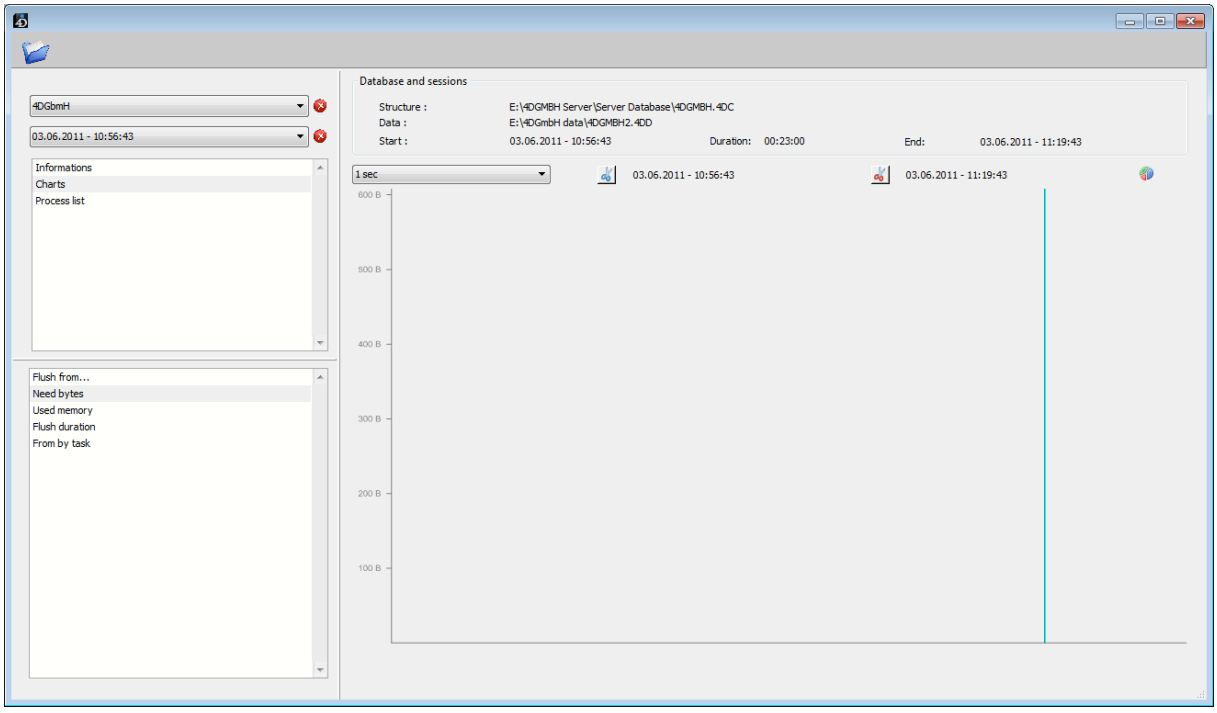


The chart shows red triangles below the chart. Move the mouse over a triangle to get the comment displayed. This option allows to associate events/actions with cache usage, like "started statistic method xyz" or "method xyz line 123".

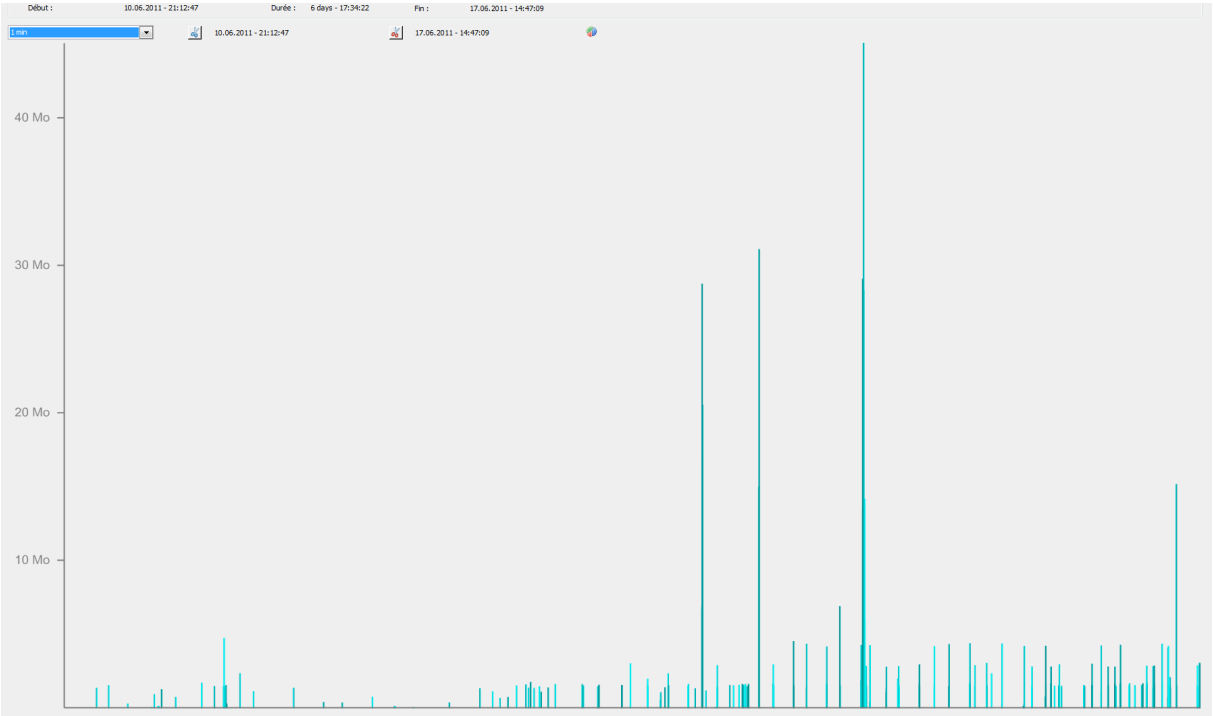
NEED BYTES

If the previous chart shows blue boxes on top of the chart, the server needed more memory as available:

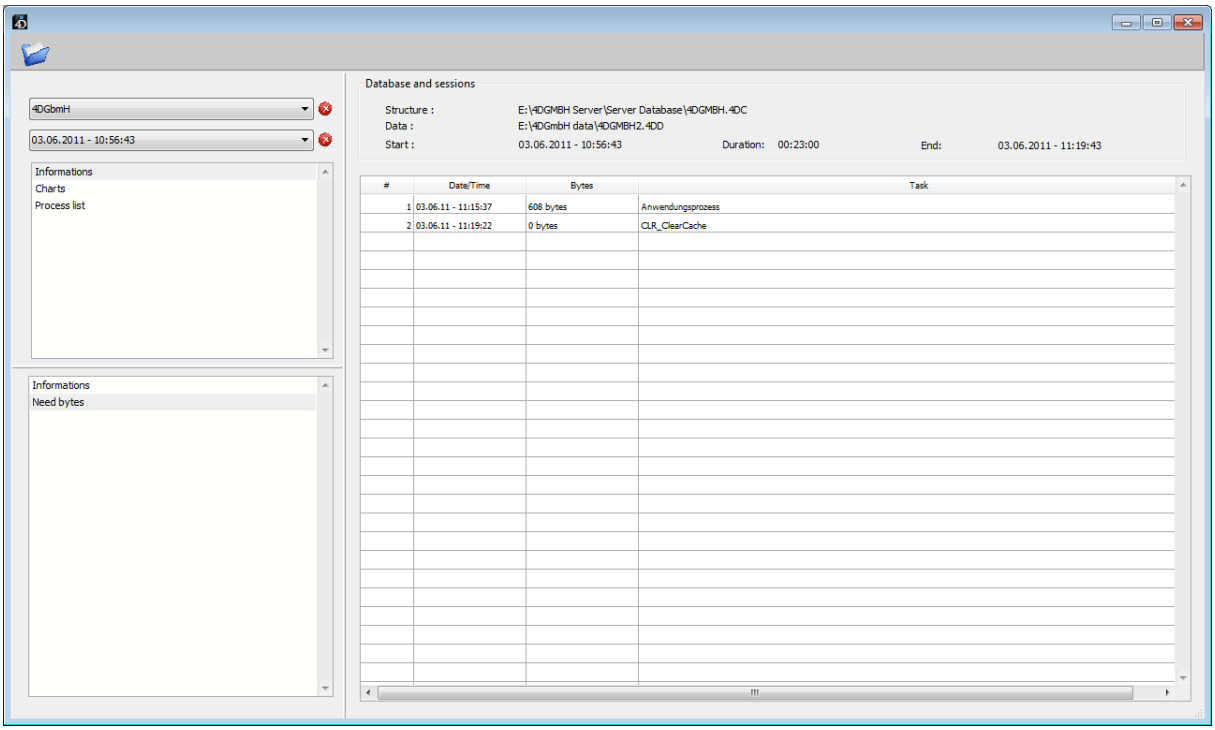
Click on "Used memory" to get details:



This example, with 608 Byte need there was only a tiny request. The next one “cries” for more memory:

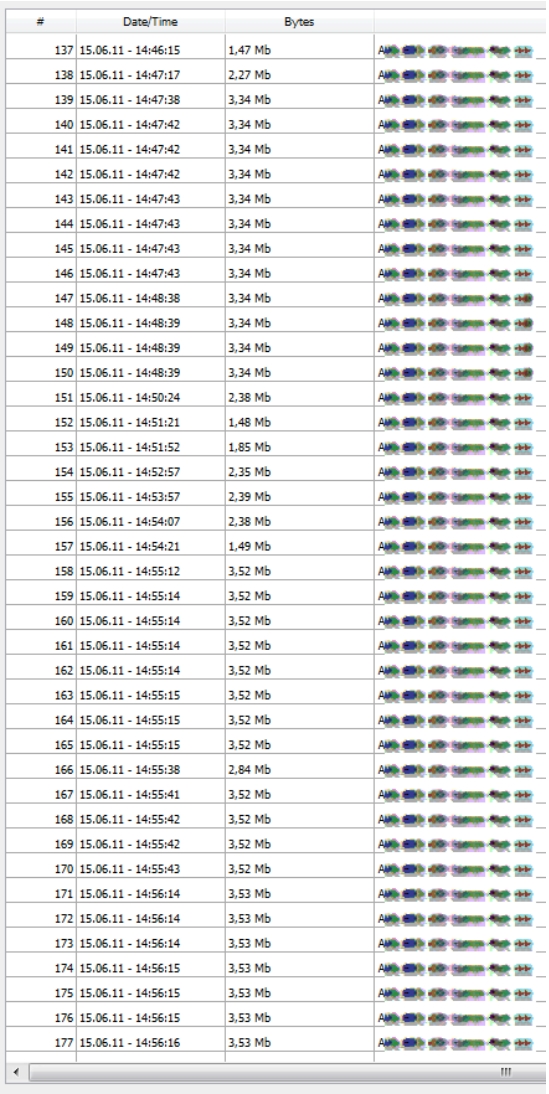


There is a second screen, showing more details, about memory need. Click in the upper listbox on “Informations” and in the lower one one on “Need bytes”:



This screen list every memory request by date, size and process name. The 2nd one in the example above was a call to FLUSH BUFFERS(*).

The same screen for the example with the 45 MB memory need:

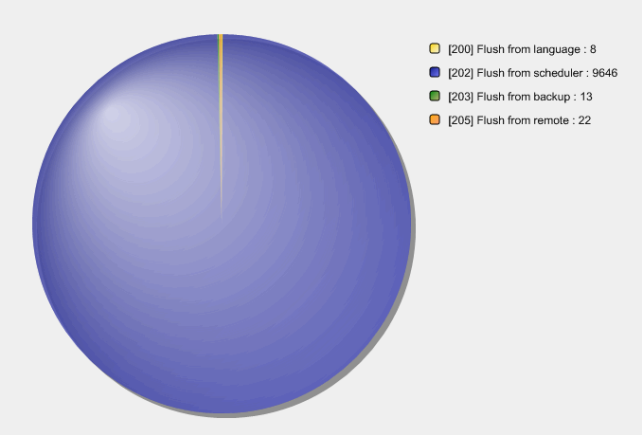


#	Date/Time	Bytes	
137	15.06.11 - 14:46:15	1,47 Mb	
138	15.06.11 - 14:47:17	2,27 Mb	
139	15.06.11 - 14:47:38	3,34 Mb	
140	15.06.11 - 14:47:42	3,34 Mb	
141	15.06.11 - 14:47:42	3,34 Mb	
142	15.06.11 - 14:47:42	3,34 Mb	
143	15.06.11 - 14:47:43	3,34 Mb	
144	15.06.11 - 14:47:43	3,34 Mb	
145	15.06.11 - 14:47:43	3,34 Mb	
146	15.06.11 - 14:47:43	3,34 Mb	
147	15.06.11 - 14:48:38	3,34 Mb	
148	15.06.11 - 14:48:39	3,34 Mb	
149	15.06.11 - 14:48:39	3,34 Mb	
150	15.06.11 - 14:48:39	3,34 Mb	
151	15.06.11 - 14:50:24	2,38 Mb	
152	15.06.11 - 14:51:21	1,48 Mb	
153	15.06.11 - 14:51:52	1,85 Mb	
154	15.06.11 - 14:52:57	2,35 Mb	
155	15.06.11 - 14:53:57	2,39 Mb	
156	15.06.11 - 14:54:07	2,38 Mb	
157	15.06.11 - 14:54:21	1,49 Mb	
158	15.06.11 - 14:55:12	3,52 Mb	
159	15.06.11 - 14:55:14	3,52 Mb	
160	15.06.11 - 14:55:14	3,52 Mb	
161	15.06.11 - 14:55:14	3,52 Mb	
162	15.06.11 - 14:55:14	3,52 Mb	
163	15.06.11 - 14:55:15	3,52 Mb	
164	15.06.11 - 14:55:15	3,52 Mb	
165	15.06.11 - 14:55:15	3,52 Mb	
166	15.06.11 - 14:55:38	2,84 Mb	
167	15.06.11 - 14:55:41	3,52 Mb	
168	15.06.11 - 14:55:42	3,52 Mb	
169	15.06.11 - 14:55:42	3,52 Mb	
170	15.06.11 - 14:55:43	3,52 Mb	
171	15.06.11 - 14:56:14	3,53 Mb	
172	15.06.11 - 14:56:14	3,53 Mb	
173	15.06.11 - 14:56:14	3,53 Mb	
174	15.06.11 - 14:56:15	3,53 Mb	
175	15.06.11 - 14:56:15	3,53 Mb	
176	15.06.11 - 14:56:15	3,53 Mb	
177	15.06.11 - 14:56:16	3,53 Mb	

As you can see there was at some times, as 14:55:14 several requests with 3.5 MB each, all from the same process (name blured here), sometimes several requests per second. Beside increasing cache memory another possible approach would be to analyze this process, using additional logs like SET DATABASE PROPERTIES(Debug Log Recording;3) or insert readable comments into the Cache log using LOG EVENT(4;”Methode XZY line 123”).

FLUSH FROM

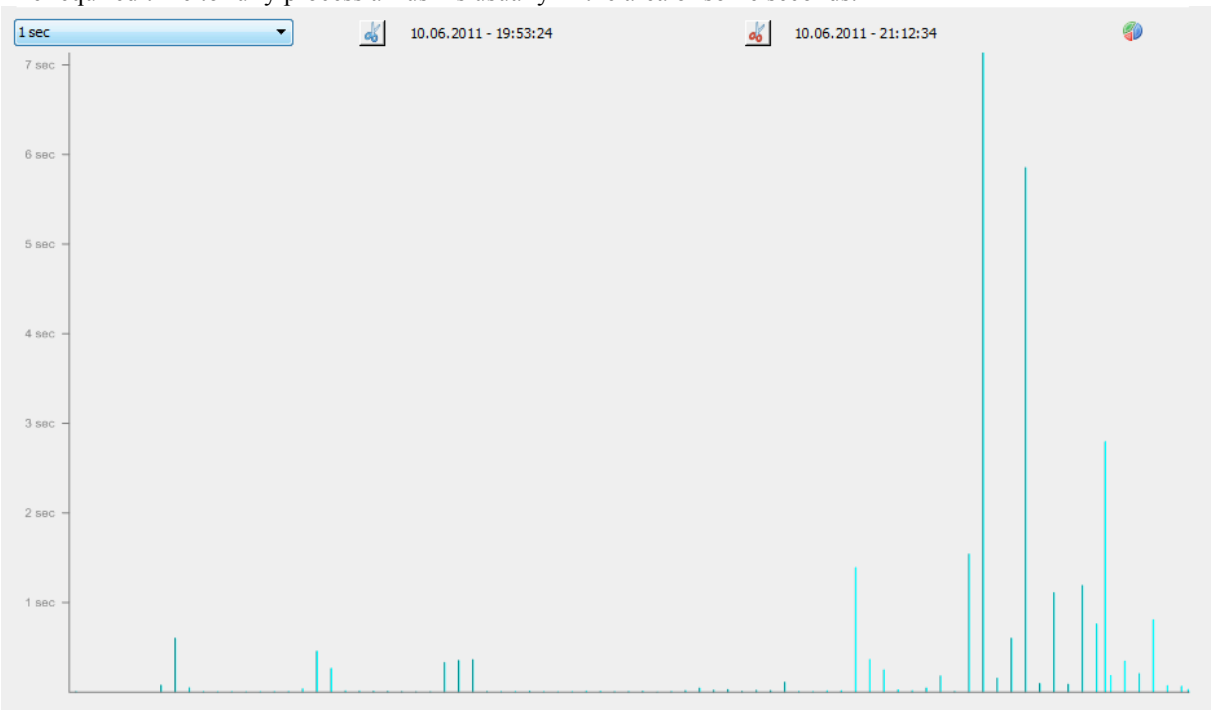
This chart is for statistical reasons only, it shows why Flush Buffers was called:



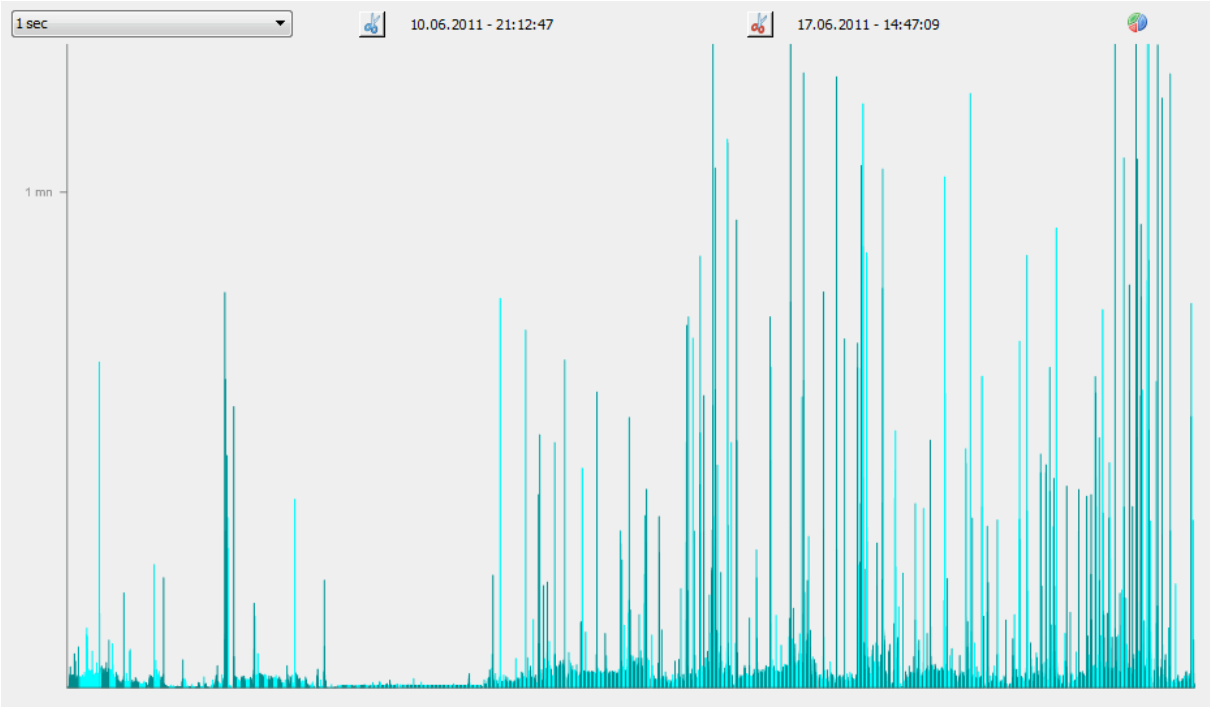
In previous 4D releases, especially Version 6, it was quite common to call FLUSH BUFFERS after important operations to be sure the data was saved on disk. With the integration of 4D Backup this need decreased, with the fully integration of automatic recovery it was not needed anymore and with the change of the cache architecture in 4D v11.7/v12.2 to flush the cache every 10-20 seconds it becomes useless. Still many applications have calls for historic reason. This dialog is a quick way to check.

FLUSH DURATION

The required time to fully process a flush is usually in the area of some seconds.

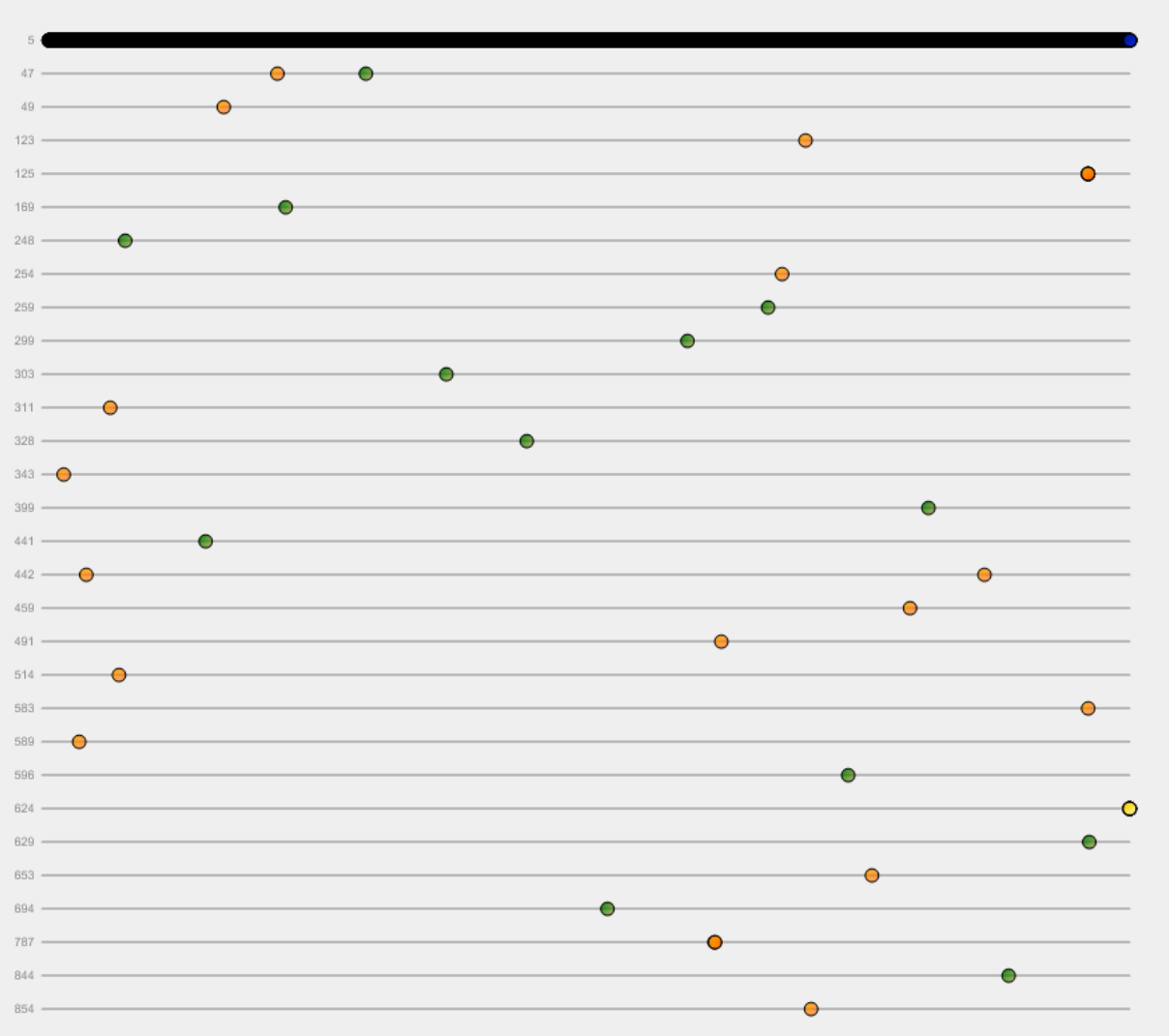


If it requires minutes it could be a signal for a very busy server or too slow hard disk:



FLUSH BY TASK

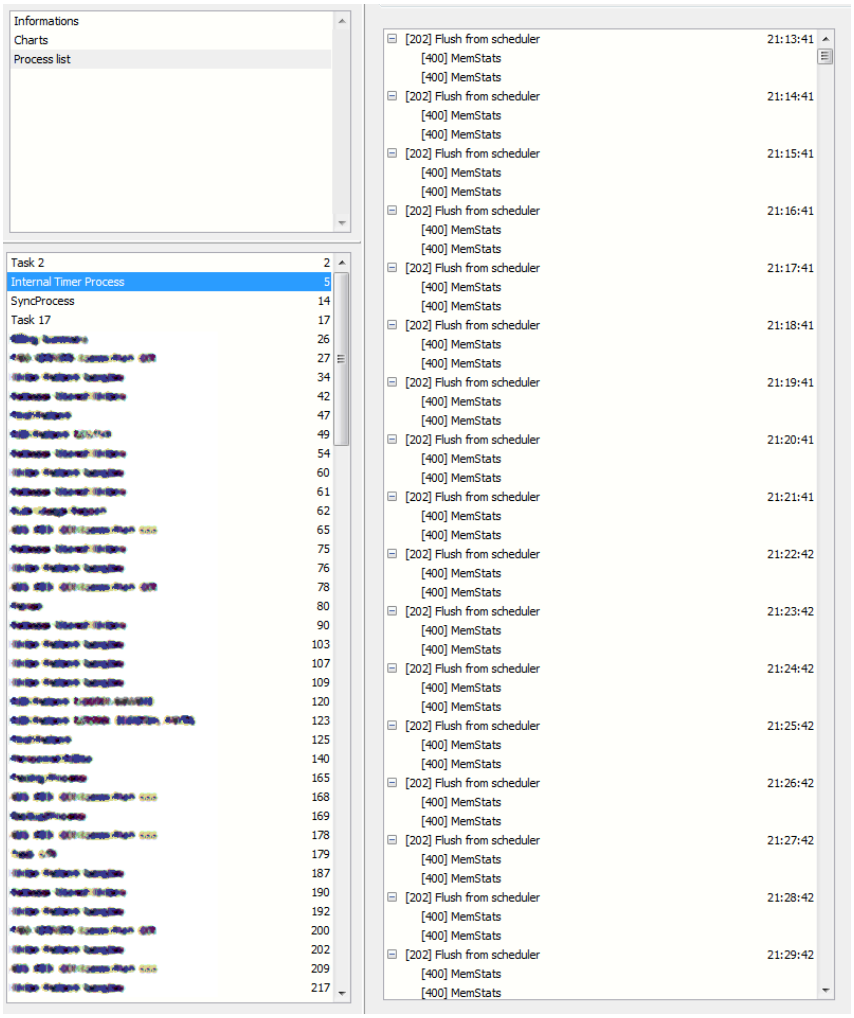
This screen shows who triggered a flush, sorted by process number:



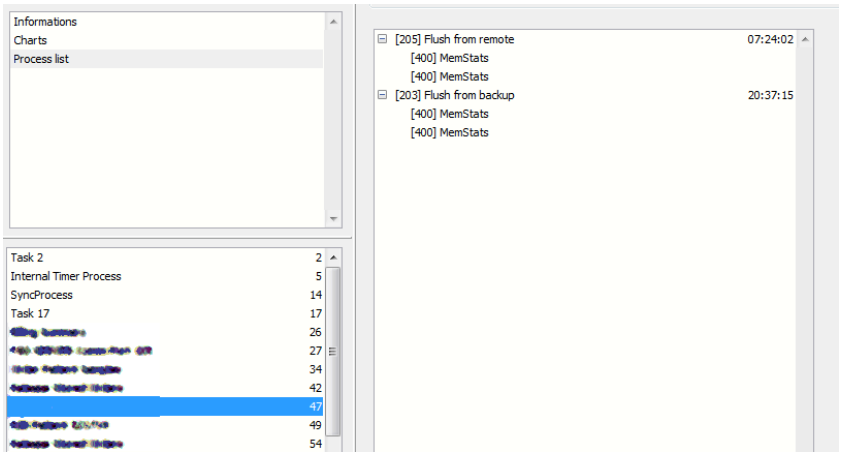
The colors have the same meaning as in the “Used memory” dialog. Process 5 in the example above is the flush scheduler, doing a flush normally every 20 seconds, here once per minute.

The flush times could be analyzed in detail using the last screen of “Overview”, Process list:

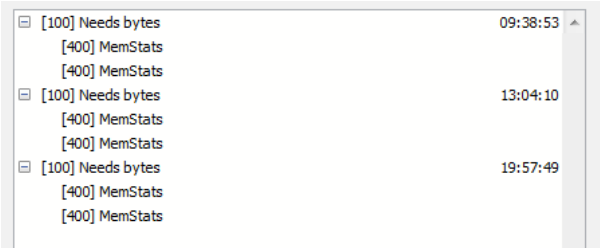
PROCESS LIST



The screen above shows the automatic scheduler, calling Flush Buffers once per minute. The screen below show process 47 from the chart before, calling once Flush Buffers from a client and once starting 4D Backup:

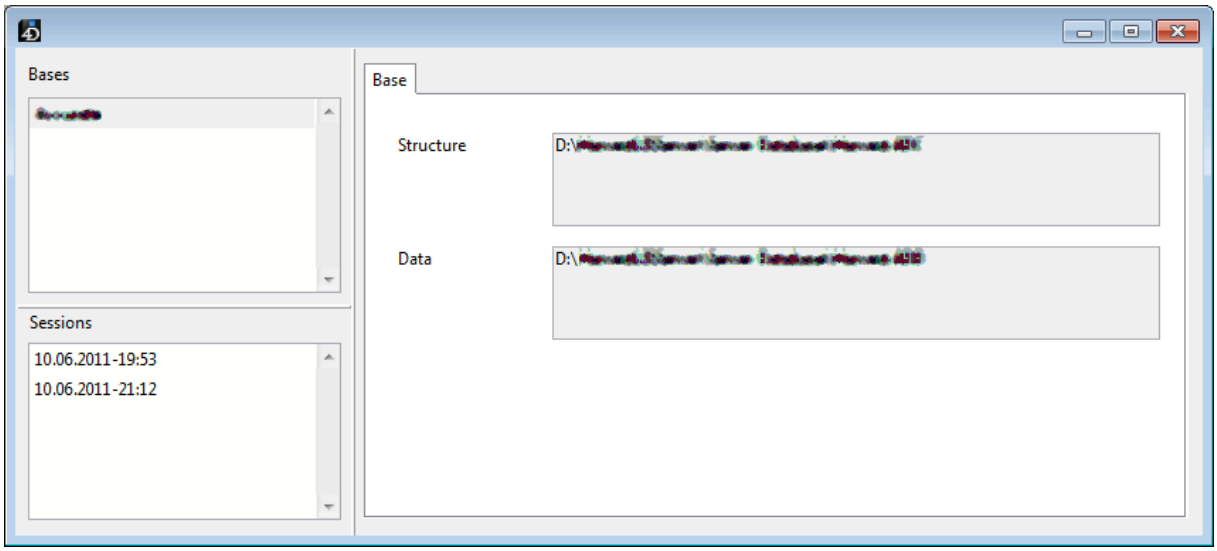


If the flush was triggered from too less memory, the list looks as this:

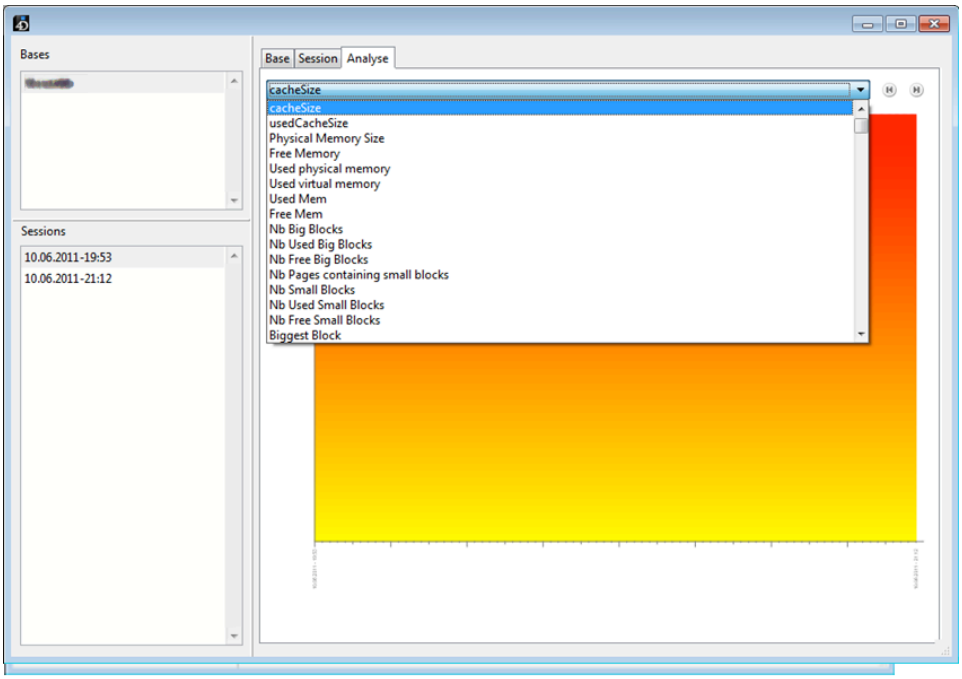


STATISTICS DETAILS

This dialog gives a huge amount of details about the cache situation.



After selecting a session (time frame), the dialog get's more options:



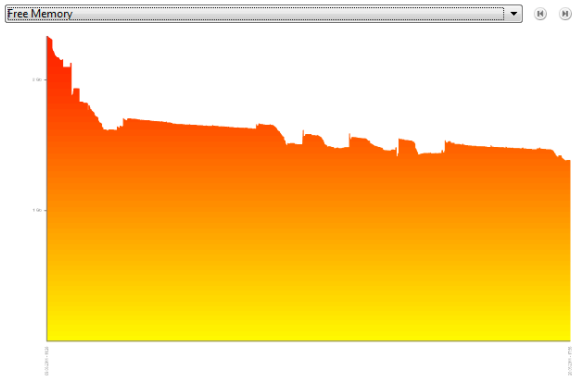
The popup allows to select which information to be displayed. The "CacheSize" is for information only, as it will not change inside the selected time frame.

USED CACHE SIZE

This charts looks like a filled area of the used memory chart, it's based on the same data:

FREE MEMORY

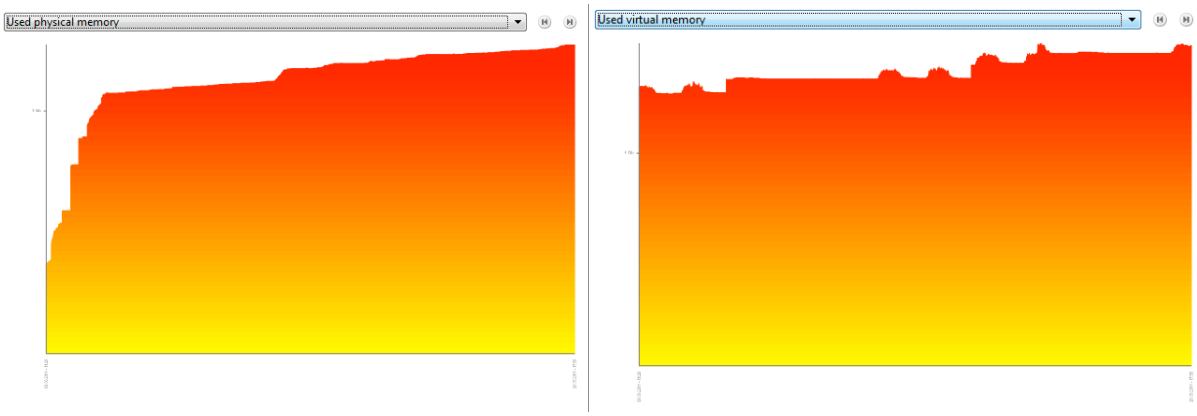
This chart shows the free memory as the OS sees it, similar to the value displayed in Windows Task Manager or Apple Activity Manager:



While Windows thought that more than 60% of the memory is free/unused, the available cache memory for 4D (see above) was very full.

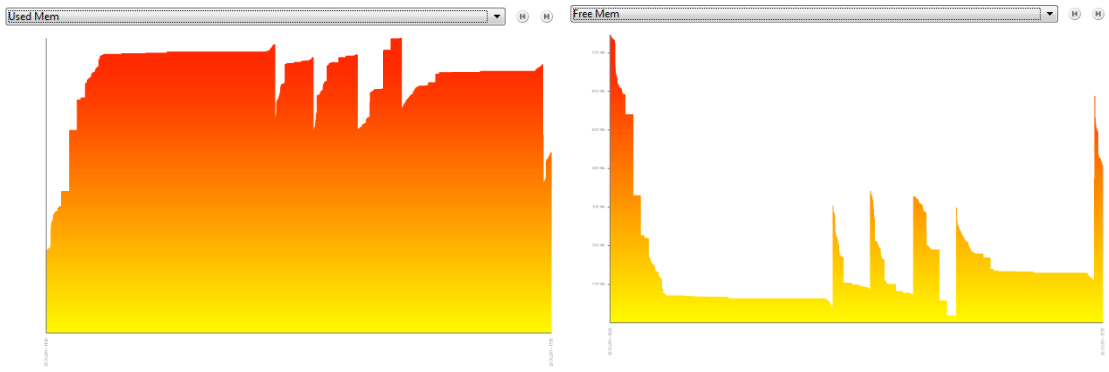
USED PHYSICAL MEMORY AND USED VIRTUAL MEMORY

While the "free memory" contains both physical and virtual memory, the chart "Used physical memory" gives a more accurate view, it shows how many of the physical available RAM is already I use. Used virtual memory allows to compare both:



USED MEM AND FREE MEM

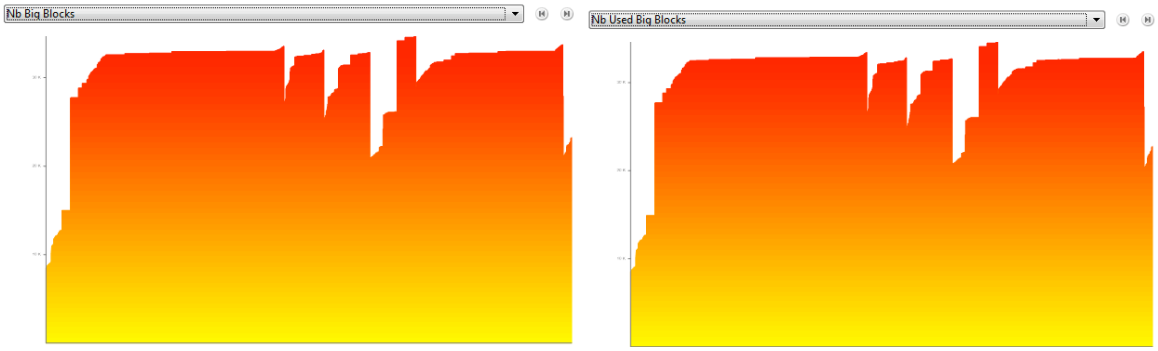
While Used physical/virtual memory is the memory as seen from the OS, Used Mem and Free Mem is the total memory as seen from 4D. Total memory means both Cache and Kernel memory.



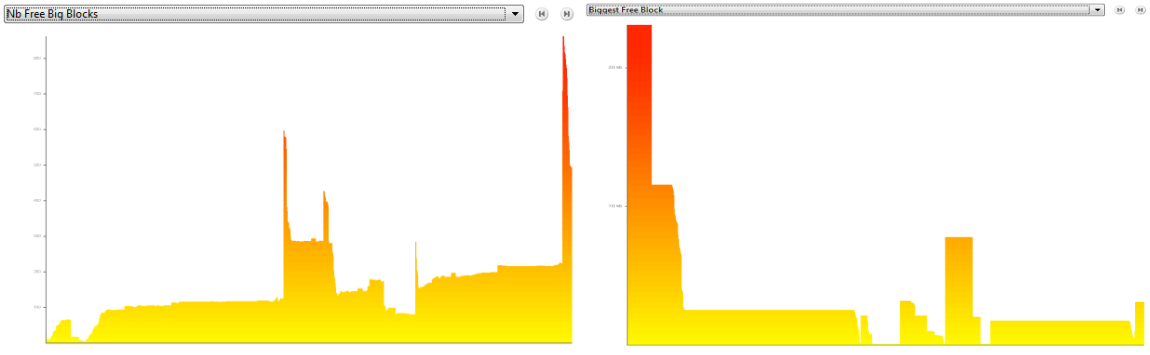
SMALL AND BIG BLOCKS

4D handles very small and big objects differently, simply because it is always easy to find space for a small block, while – even with some empty memory – it could be quite difficult to find one continuous block of large memory. If you run into cache issues, it is most likely because there is not enough space for big blocks (like a large set or named selection).

BIG BLOCKS

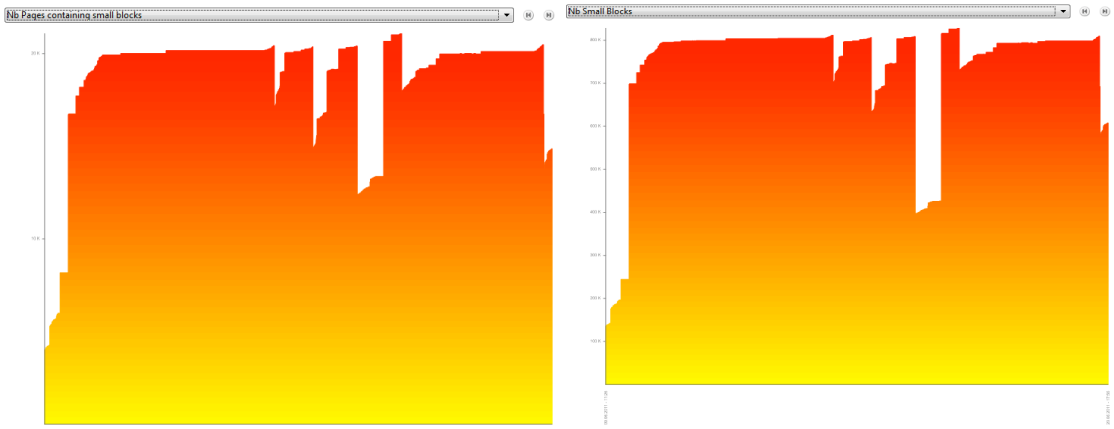


(numbers left are 10k, 20k, 30)

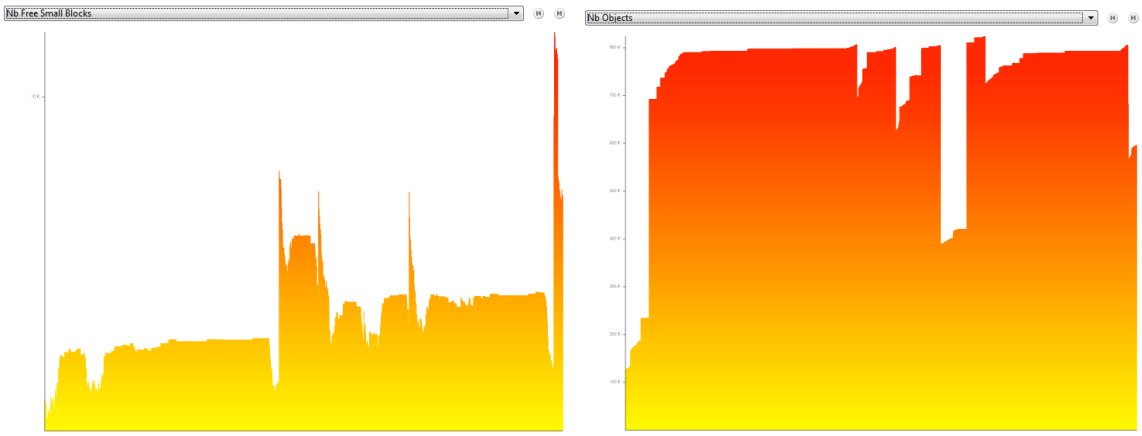


(numbers left are 100, 200, …, 800) (numbers left are 100 MB, 200 MB)

SMALL BLOCKS



(numbers left are 10k, 20k) (numbers left 100k..800k)

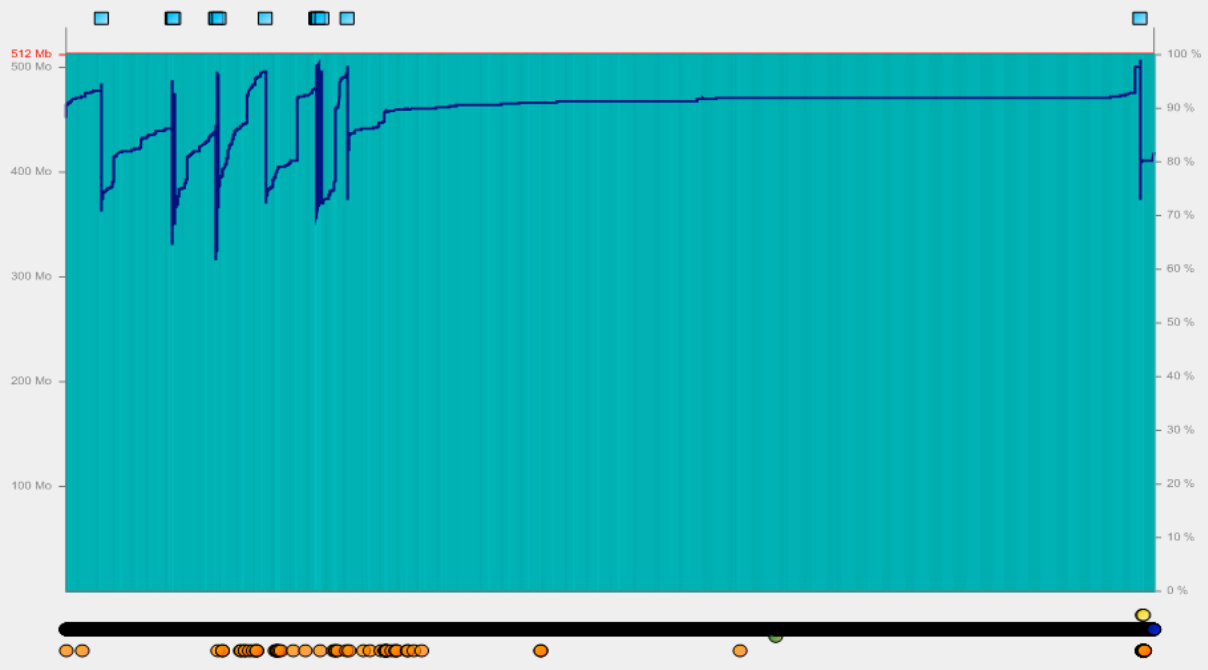


(number left is 50k) (numbers left 100k..800k)

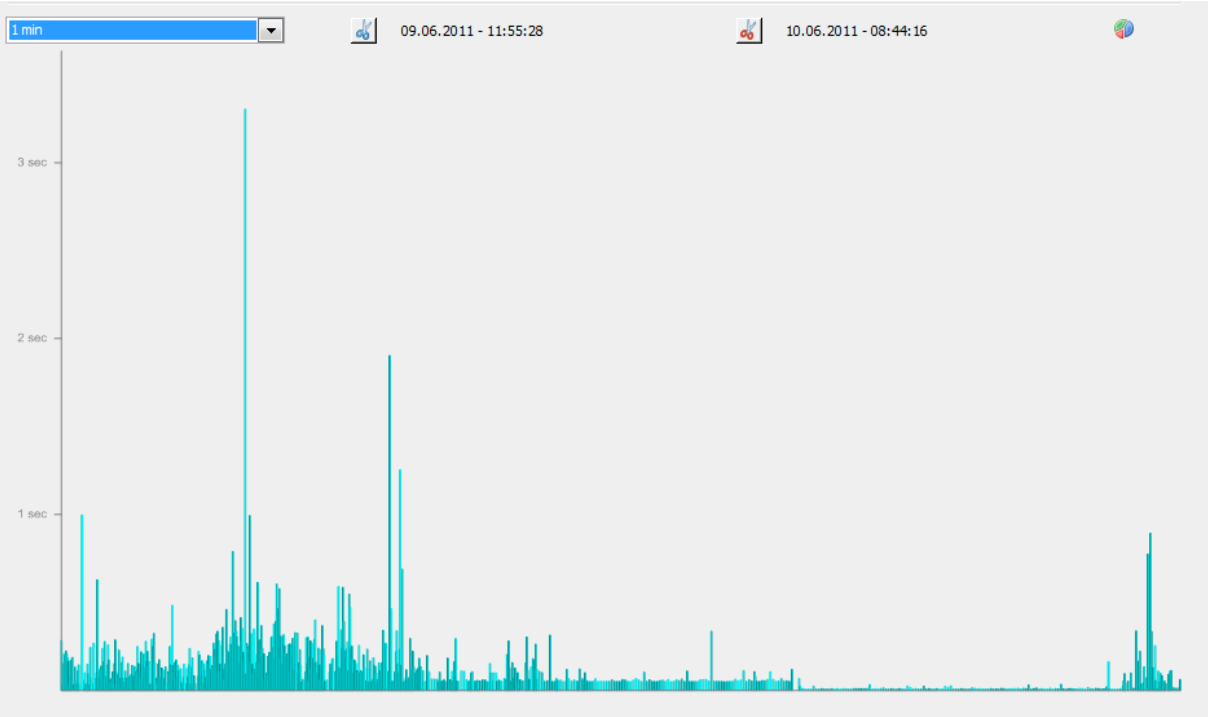
As you saw, you have a huge source of information's; the difficult part is to understand them. To help you mastering this task we will provide some examples, collected with real installations.

EXAMPLES

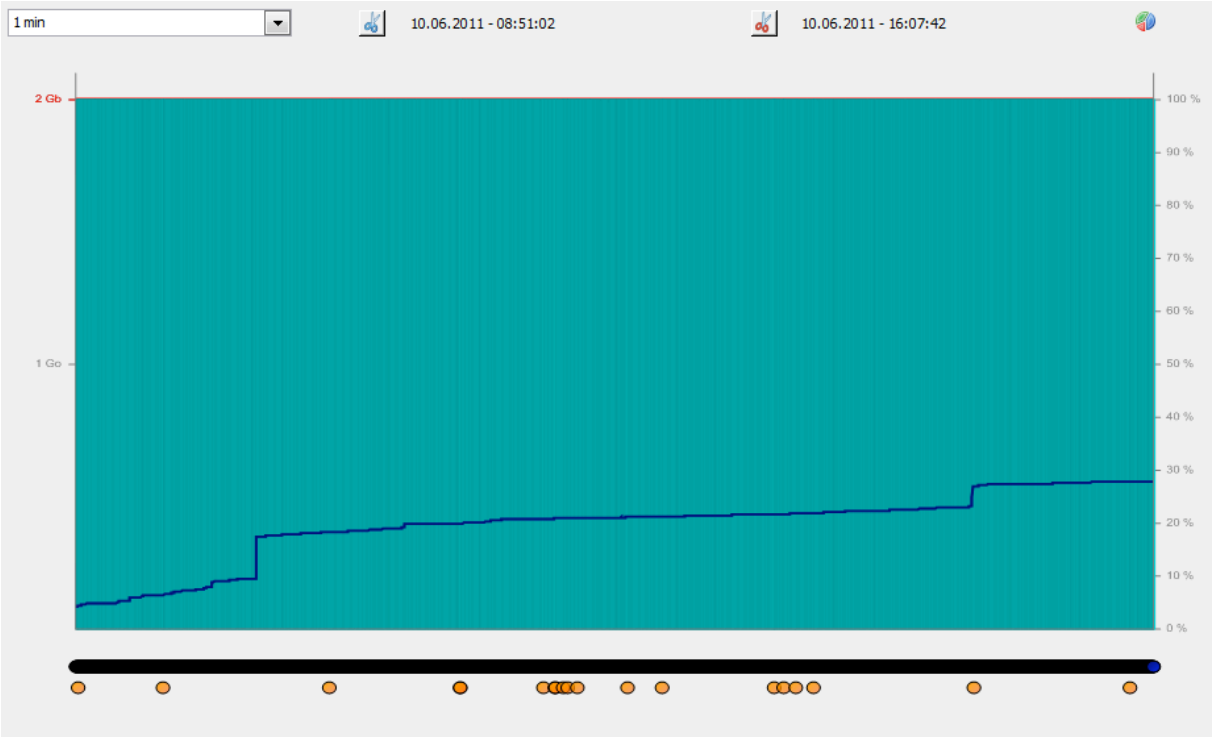
We start with a server hosting 20 users, usually 10 processes per user. The customer runs it on a 8 GB computer with Windows Server 2008 64 bit – but 4D's cache limited to 512 MB:



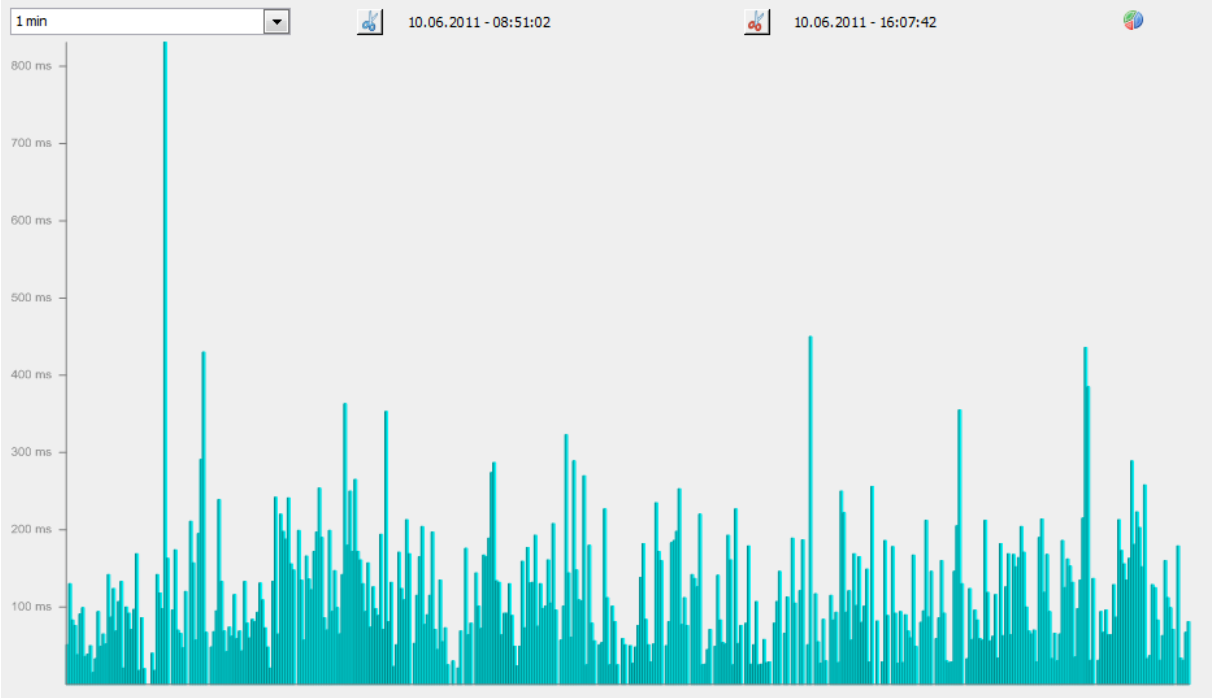
The chart above shows the cache usage. Easy to see that it is often requires more RAM – then it stays constant at 450 MB (this is the night), to continue the "up and down" next morning. The chart below shows the time required to flush the cache. It is usually below one second with some small increases, nothing really to worry:



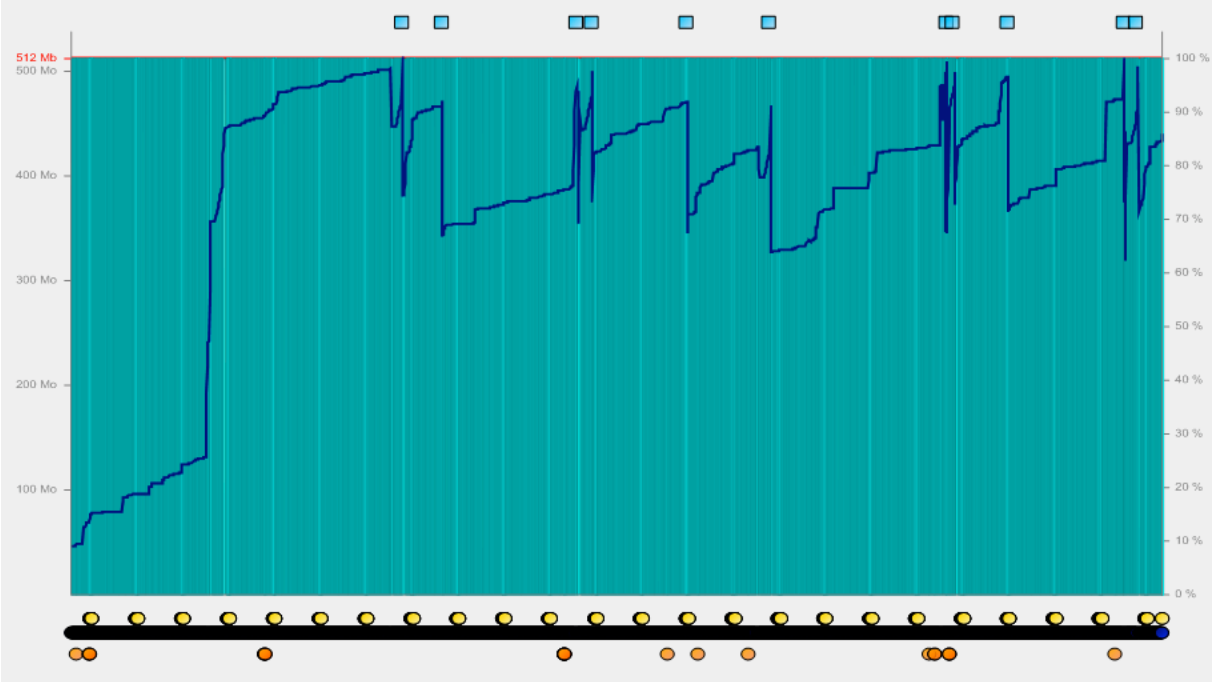
As the customer had a powerful computer, we asked to increase Cache to 2 GB and restart:



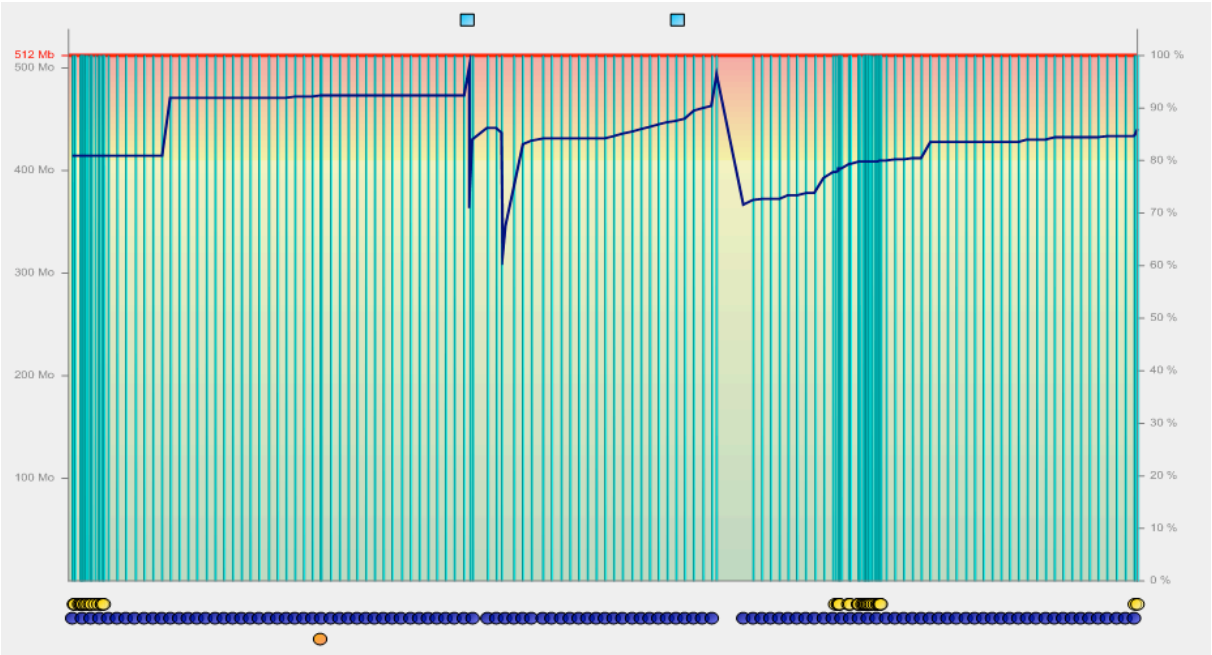
The chart above shows the same solution, same 4D app, same computer, same users, next day with 2 GB Cache. As you can see – it never needed as much and started to build up a cache, to avoid hard disk access. Over the next days it increased up to 60%, never needing to purge data. The chart below shows the time needed to flush. At the first view it looks higher, but you will see that the scale switches from 1-3 sec to milliseconds:



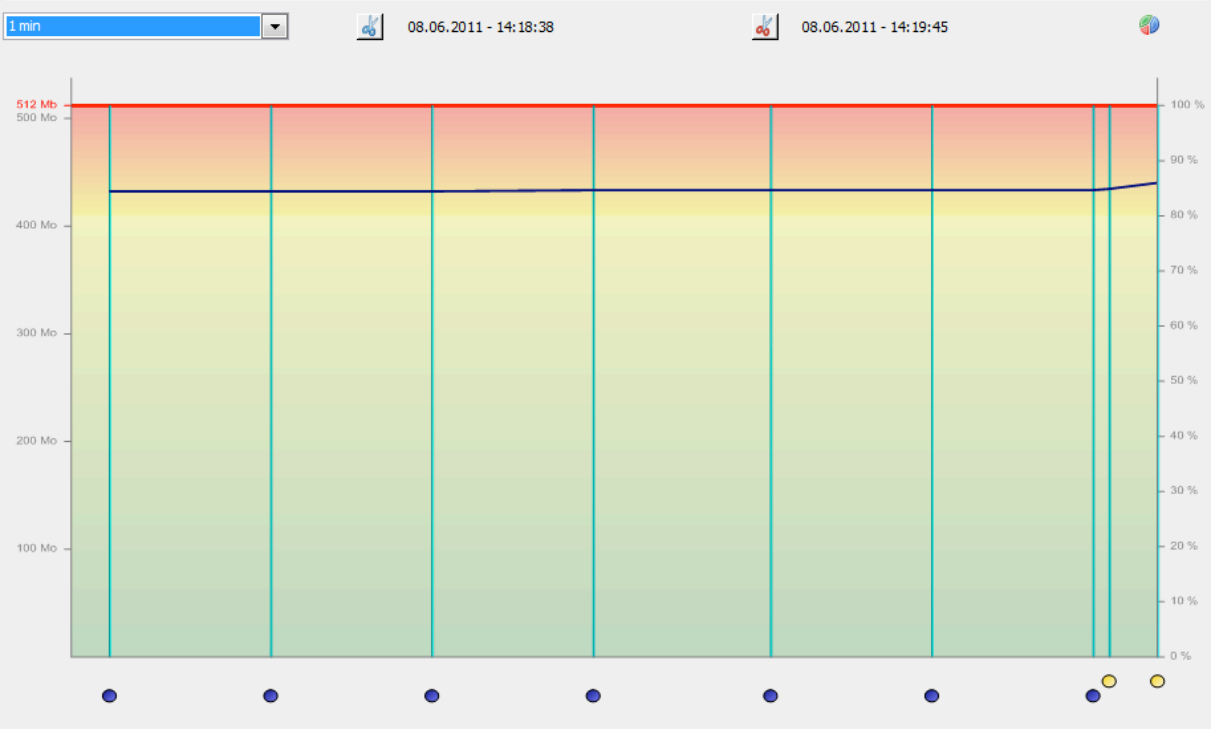
This customer reported random crashes, not reproducible:



We asked to use the cache log recorder to see what's going on. The chart showed that it used only 512 MB Cache (for about 40 users, 600 processes), the cache quickly filled up, needed often more RAM so purged items and reached max again. A zoom in the last minutes shows "wild jumps", "freezes" of the server (the green vertical lines are normally every 20 seconds for automatic flush, the longer distances means that the server was so busy freeing memory that it did not reacted normally), then several yellow bullets where the developer called Flush buffers himself, up to several times per second – and a very last yellow bullet at the end:



A zoom in the very last minute before the crash showed a memory increase at the end and a flush at the very last millisecond:

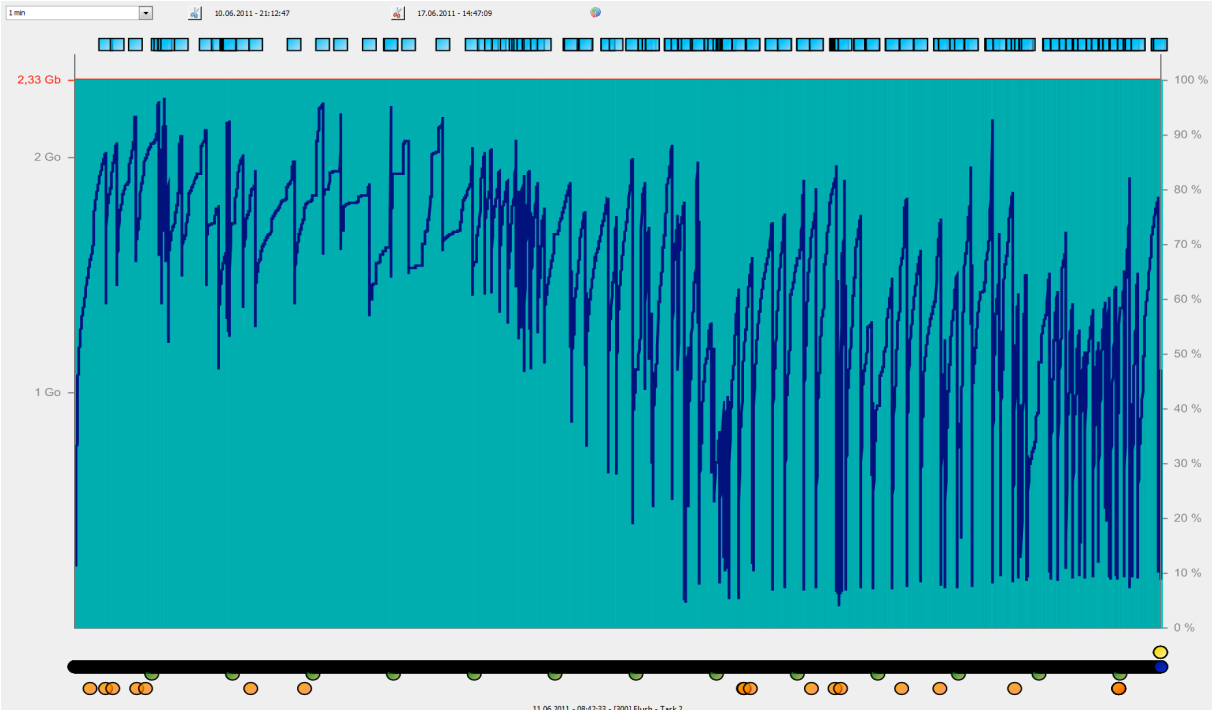


As the customer needed an urgent quick solution we never found the real origin of the crash. As quick solution we recommended:

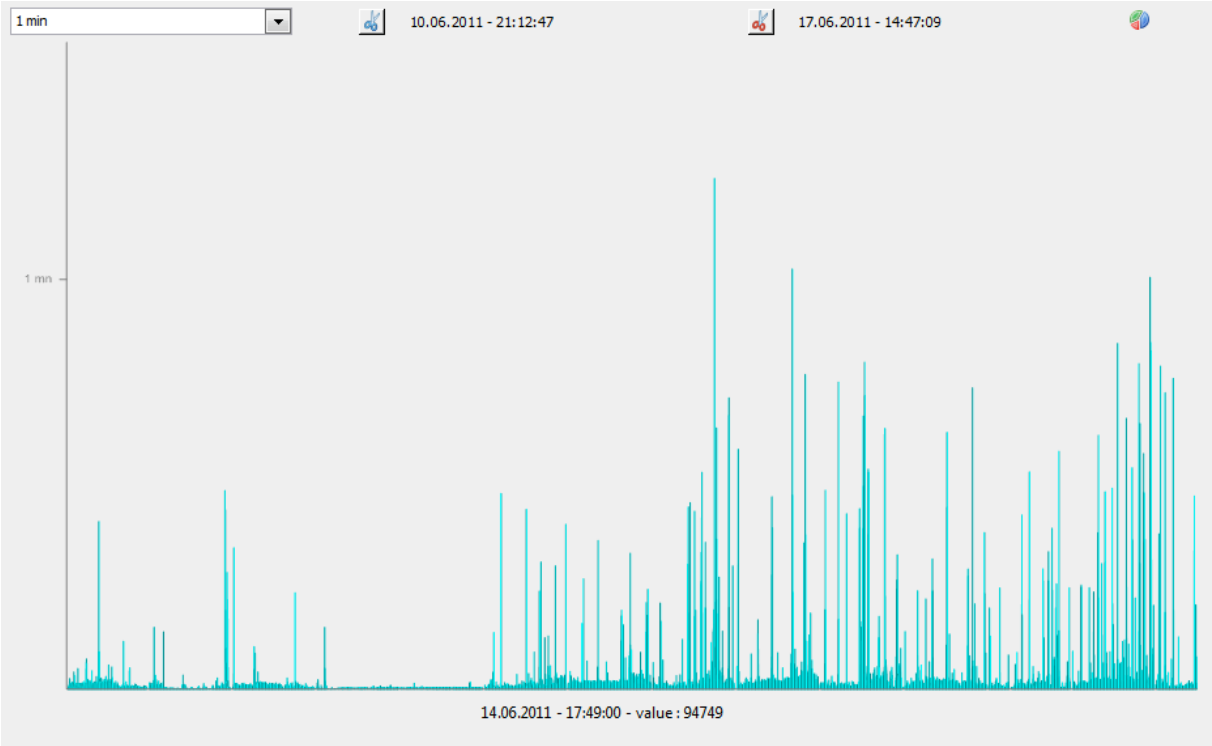
- Increase cache from 512 MB to 1200 MB (computer was a 32 bit OS)
- Remove manual FLUSH BUFFERS calls to avoid additional work in low memory situations
- Upgrade the OS to 64 bit in next weeks to enhance RAM, also to speed up

As the first two steps removed the crashes the customer did not wanted to spend more time to dig into the issue. The upgrade to 64-bit and 8 GB also produced visible speed increase.

Next chart shows an app with 50 users, up to 875 processes. The app was running on a 64 bit OS, but using 4D Server 32 bit. 2.3 GB Cache assigned, which is the max for the 32-bit Server:

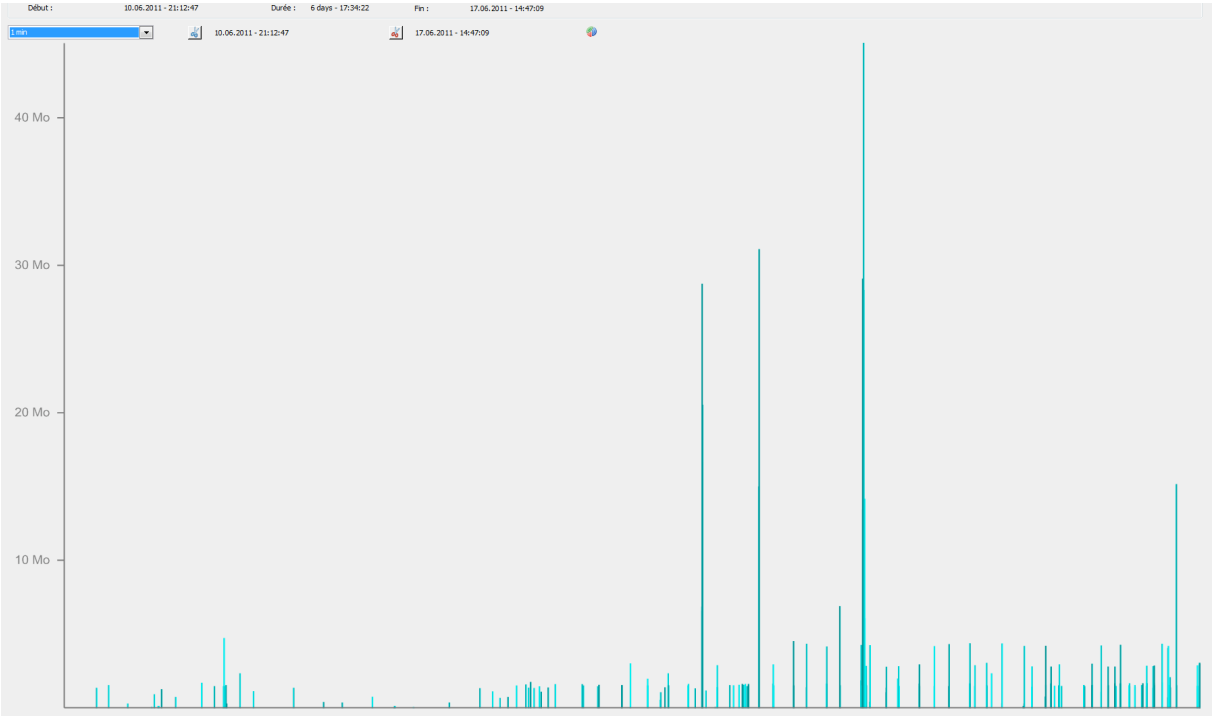


We can see a huge up and down of the cache usage. The blue boxes on top shows that the server quite often requested more memory and needed to dump large parts of the cache to get room for other stuff, just to dump that again some seconds later.

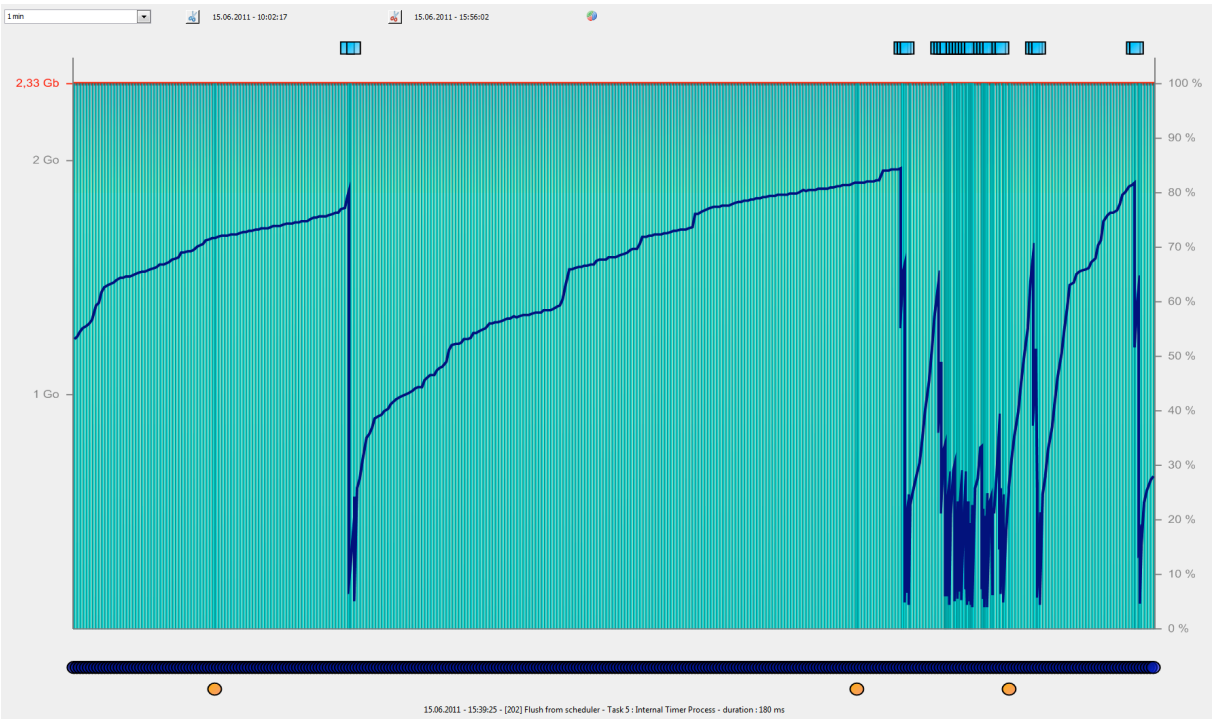


The flush time with more than 90 seconds shows that the server was very busy, I guess because it needed to access the hard disk so often, as the cache was too small to avoid it.

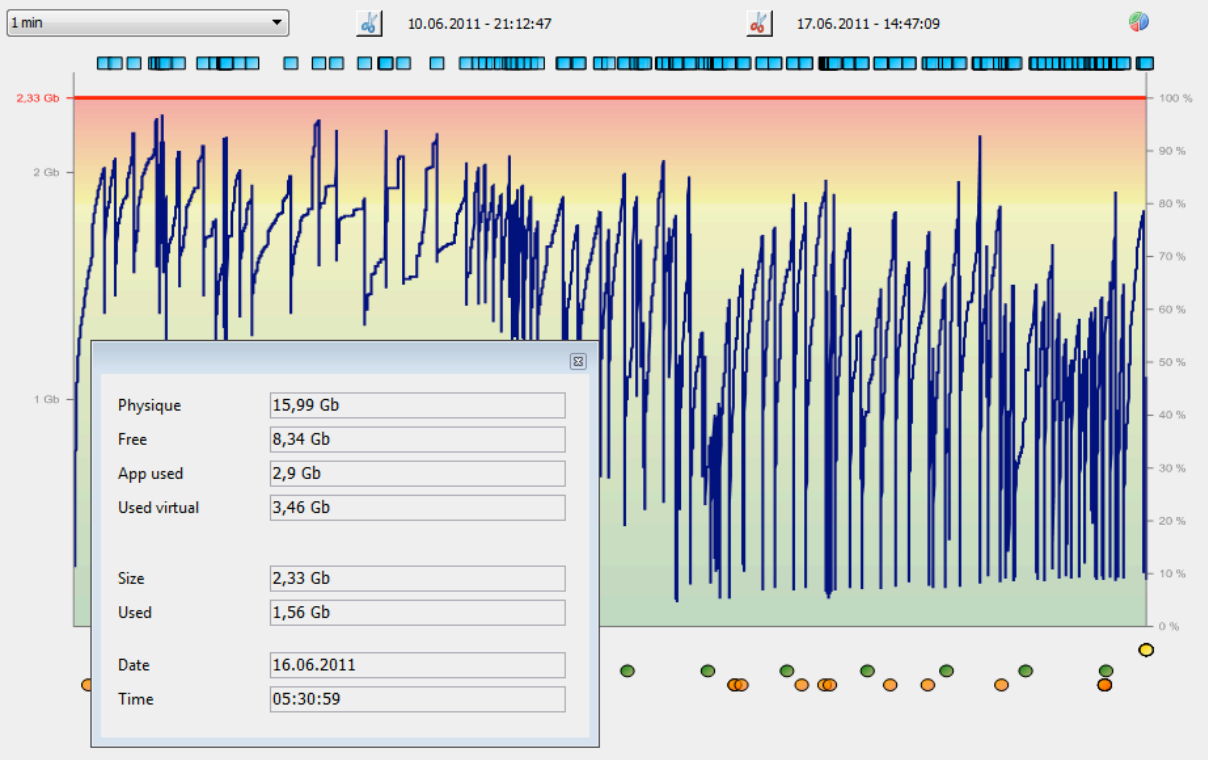
Also the chart for “need bytes” shows that the memory requests with up to 45 MB was quite large:



Zoom in (this is a 5 hours fragment) shows that the cache was often totally purged, then filled up in the next 2-3 hours, to be cleared again. And that it could happen that the memory requests followed so fast that the data was quite often purged:



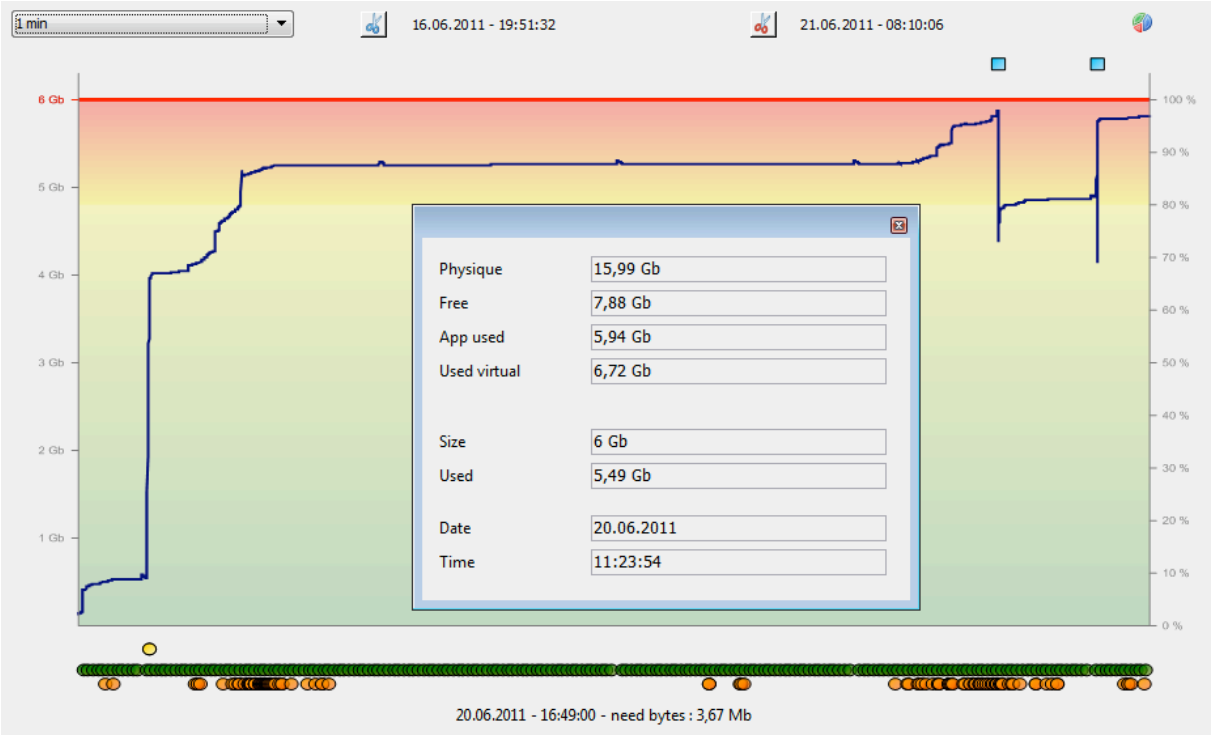
Several times the total used memory went high up:



The highest peak for used virtual memory was 3.805228, which is very close to the max of 4 GB which a 32 bit application can take. As a result we needed to ask the developer to even REDUCE the cache from 2.3 GB to 2 GB or even 1.8 GB, as in no case the 4 GB must be reached (even the chart shows that 2.3 GB is already too small for an app with so many records, users and processes).

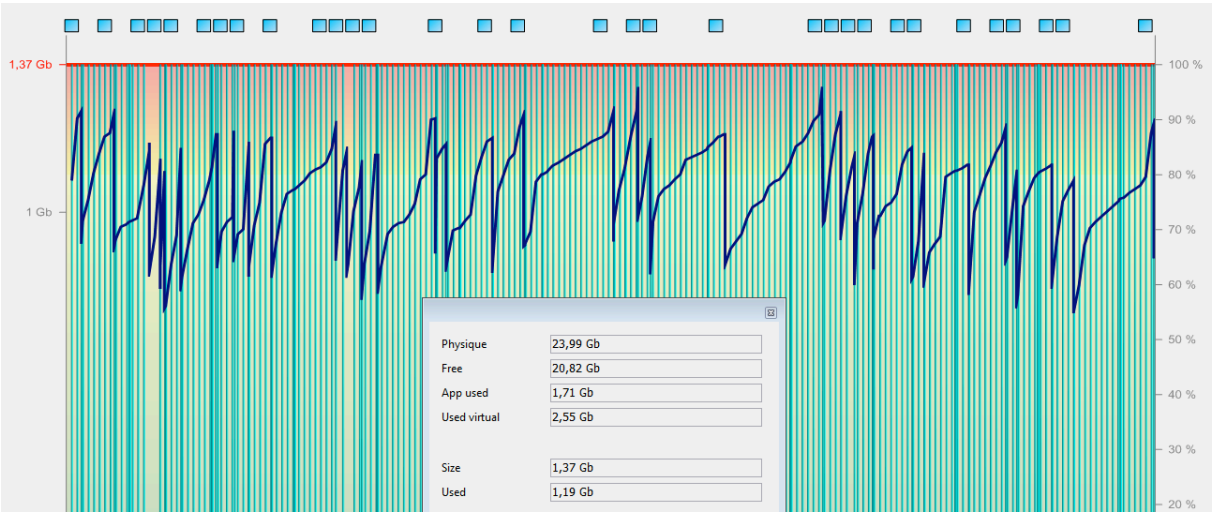
As you can see the computer has 16 GB RAM and most of it was unused, so the real solution is to use the 64 Bit version of 4D Server.

The next example shows a customer with 75 users, about 250 processes, running 4D Server v12 64 Bit on Windows Server 2008r2 64 bit with 16 GB RAM:



The chart above shows a full week running the app with a quite stable memory situation. There was two peeks, with the result of a small purge. As the computer still has about 7.8 GB unused we recommended to increase the cache from 6 GB to 10 GB.

The next chart shows an one hour peak into a 24 GB computer running a 32 bit Server:



It's easy to see that the 64-bit version makes sense for this kind of installation (here with 150 users).

ACKNOWLEDGMENT

4D Cache Log Recorder and 4D Cache Log Analyser was written by Olivier Deschanels and tested and enhanced by Aziz Elgohmari, 4D SAS, France.

4D INFO REPORT COMPONENT

(also refered as “information Component”, “Report Component” or “aa4D_report”)

Technical Support 4D SAS France has developed a very helpful 4D component to provide information’s about the used hardware/OS at customer side and to monitor memory usage. The component is very simple to use and works with both v11 and v12. It provides a huge amount of information’s, we give here an overview and some hints how to use all these data.

Hardware Customer

Computer name: my4Dserver
Computer user: SYSTEM
Manufacturer: IBM
Computer Kind: IBM eServer BladeCenter HS20 -[8832M1X]-

Total RAM: 3968 MB

Number of Processors: 1
Processor Name: Intel Xeon
CPU Speed: 3.06 GHz
Total number of Cores: 2
Total number of CPU threads: 4

Operating System: Windows Server 2003 Standard Edition SP2 (32-bit)

Computer kind can be googled…

Here a computer designed for Linux or Web Cluster. Shipped with 512 MB RAM, ATA-100 drives, 512 kb CPU Cache, currently sold at eBay for $100. Customer said that he uses an expensive ultra high end server with huge amount of memory, cannot be the hardware, and send a huge description of the expensive network system they have.

The report told us that the hard disk is very slow, the CPU not bad but not fast, and most important, running a 32bit system. From the 4 GB in the computer Windows 32 bit usually sees 3 GB. So 1 GB for the system, max 2 GB for 4D.

Another example:

Computer name: xxx
Computer user: Adm
Manufacturer: Dell
Computer Kind: PowerEdge 2950

Total RAM: 4096 MB

Number of Processors: 1
Processor Name: Intel Xeon X5450
CPU Speed: 3.00 GHz
Total number of Cores: 4
Total number of CPU threads: 4

Operating System: Windows Server 2003 Standard Edition SP2 (32-bit)

Again a 32-bit system with 4 GB. Searching for the computer showed that Dell ships it as optimized for database server, high performance, with extra ordinary high memory speed, with up to 32 GB buffered RAM.

So the customer bought an expensive, great computer with minimum RAM on a 32-bit system. Like a Ferrari with a gear lock, limited to 1st gear.

Why is this so important? On a 32-bit System 4D has max access to 2 GB. The operating system get's the remaining 1 or 1.5 GB, never 2. Windows may need more…

2 GB was a lot 10 years ago. 4D Server v11 was designed for 64 Bit systems…

On a 64 Bit System with 8 GB RAM (or more), 4D can access 4 GB, so 4 GB remains for the OS. This means that 8 GB should be minimum RAM if 4D's the only application running on the server, else more.

We come back to memory later…

```
Computer user:                          SYSTEM
Manufacturer:                           VMware
Computer Kind:                          VMware Virtual Platform

Total RAM:                              3072 MB

Number of Processors:                   1
Processor Name:                         Intel Xeon  E7330
CPU Speed:                              2.40 GHz
Total number of Cores:                  4
Total number of CPU threads:            2

Operating System:                       Windows Server 2003 Standard Edition SP2
(32-bit)
```

Oh, customer said he has a 8 Core computer with 16 GB RAM – forgot to mention that he runs several virtual machines on it and 4D's get's only a small part of resources…

```
       ============== 4D Infos ==============

Application type:                       4D Server
Application version:                    v11 SQL release 6  (F0021160)
Internal version build of 4D:           11.6 (74459)
```

Two important information in this section: the exact build of 4D, with clear text (like release 6) and internal build, allowing to identify hot fixes or even intermediate builds. It allows to check if the customer really installed a version..

```
Data File:
"D:\Data\CRM_DATA_PROD.4DD"
      Size:       3 243 968 KB              Modif:    2010_04_09__09_28_03
Involved volume(s) list (free space):

Volume      C:\ (Operating System)              3 588 MB
Volume      D:\                                 4 702 MB
```

This is real, don't laugh. Still on the virtual system. The datafile is 3.2 GB big. On Drive D. D has just 4.7 GB free space. If something fails and 4D runs the automatic recovery, hard disk is fully very soon – and then everything stops. Admin has assigned a small image to the database server to save room on his expensive disk. The used application is the main application for this customer, their office cannot work without. What's the price for 10 GB hard disk today? $1? Imagine…

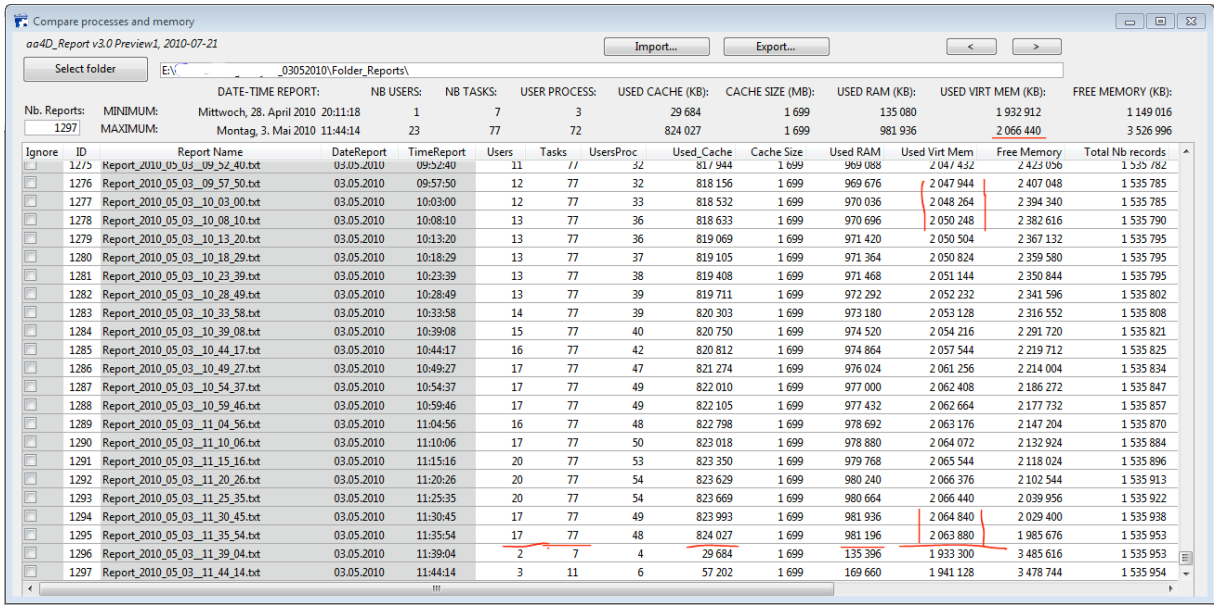
```
Web: (The Web Server is not started)
Count tasks:                                64
Count user processes:                       16
Number of users:                            3

    ------ Database parameters ------

Database Cache:                             1 699 MB
4D Server Timeout:                          3 Minute(s)
4D Remote Timeout:                          3 Minute(s)

Selector 28 (4D Server Log Recording):                                  0
Selector 34 (Debug Log Recording):                                      0
Selector 53 (stack size off preemptive thread):                         1024 KB
Selector 54 (Idle Connections Timeout):                                 0 seconds.
Selector 61 (Maximum Temporary Memory Size):                            0 (Max.)

 ------ other infos via GET CACHE STATISTICS: ------

    Used cache Size :                          29 777 KB
    Free Memory :                           3 526 996 KB
    Used physical memory :                    135 080 KB
    Used virtual memory :                   1 932 912 KB
```

This block shows a real application, real values, Server crashed about once per day. It was running on the IBM Blade above, so a 32 bit System, 2 GB available for 4D Server.

Database Cache was set to 1.7 GB, so 300 MB remaining for kernel. Top was 64 processes; stack size for preemptive thread (Selector 53) is 1 MB, so 64 MB off. We have 240 MB remaining for 4D Server's code, internal variables and all other stuff – and many users to connect… Yes.

When this report was done the virtual memory was at 1.932 GB. 70 MB left. As soon this value comes close to 2000 – boom.

The report contains more details – this is just some highlights to show how to detect problems, all is easy to read.



Again real live data. Customer crashed about once per day. Here the crash was shortly after report 1295. In reality a little before, server started to drop connections 10 minutes before.

It is a 32 bit System, so 2 GB max. 1.7 GB was reserved for the cache, even it never needed more than 824 MB – but it is reserved. The system started with 1.933 GB usage, so only 70 MB remained for jobs. As you can see the memory increased for each new user – so either there was code in on Connection, which locked memory – or a memory leak. There was really a small memory leak, normally not noticeable, but because they had only 70 MB free RAM, they was never able to start more than 80 processes or so till they reached the limit. Normally this leak would not have produced problems for months, but with only 70 MB free…

Solution was to reduce max cache to 1 GB, so they had 700 MB more free RAM. In addition they reduced stack size for preemptive thread (Selector 53) from 1 MB to 256 KB. They usually have about 100 processes. This reduces 100 MB to 25 MB. Not much, but on a 32 bit system every Megabyte counts. A better way (from my point of view) would have been to install an additional 4 GB ($100) and use Windows Server 64 bit, which would have avoided fine tuning, spending time trying to find best values and increase speed.

This information component is mainly designed to provide in readable text documents a description of:

The computer (Hardware / Operating system)
The version of the 4D Application
The Host database settings (and tables summary)

Supported version of 4D:

4D v11 SQL version: Oldest: 4D v11 SQL release 6 (Public release)
4D v12 version: Oldest: 4D v12.1 (including 4D Server v12.1 64-bit)
(4D v13 version: Latest: (including 4D Server v13 64-bit)

Some information on this component:

It does not require or use plug-in or other component (including 4D SVG).

So it can be used with any Host database (4D v11 SQL to 4D v13), or as a stand-alone database (will ask for an initial creation of a data file when used stand-alone).

This component does not create or use table / record in the Host database (or in an external database).

Confidentiality

The component does not access or share any record content, resource content, etc...

It will only describe some basic elements of the structure (number and name of the tables, number of fields or indexes (but not their name or their type), and the number of records for each table. (optional parameter values let you hide some details).

It will get some details on the computer and operating system, and the path names used for the database and the application.

Compatibilty

4D_Info_Report_v3 can handle all reports created by aa4D_Report (since v2.0), with French or English content (the two possible ones).

Supported language for the interface and the content of the reports):

Based in the release of 4D used: in French with a French release of 4D, otherwise in English.

Supported platform and OS:

All Supported Operating systems (+ Mac OS X 10.7). Two versions are provided as

4D v11 component: One with PowerPC code included, the other with Intel code only.

Naming conventions:

Name of the component:

The name of the component now begins with "4D_Info_Report" but is known as the "Information component".
(The component used to be named "aa4D_Report" in previous releases.)

The name can be followed by:
the main version of the component (like "_v3" for version 3)
the name of the provided zip archives can be completed with more details:
(like "_v11" for version 11, "_v12" for version 12, "_v13" for version 13).

Name of the shared methods:

The name of the shared methods always begin with "aa4D_Report_" and are followed by "M_" (a method), or "NP_" (a new process), "F_" is used for the project forms.

LIST OF THE SHARED METHODS OF THE COMPONENT (BY CATEGORY)

Creation of reports

aa4D_M_CreateReport_faceless
aa4D_NP_Util_CreateReport
aa4D_NP_Util_CreateReport_Serv
aa4D_M_Report_CreateOnServerSee (executed on 4D Server only):
aa4D_NP_Shedule_Reports_Server

Display of existing reports

aa4D_M_Get_Last_Server_Report

Display of dialogs

aa4D_NP_Report_Analyse_Display
aa4D_NP_Report_Compare_Display
aa4D_NP_Report_Manage_Display

Utility

DETAILS ON THE SHARED METHODS OF THE COMPONENT

aa4D_M_CreateReport_faceless

aa4D_M_CreateReport_faceless ({ContentMode}; {CommentPointer})

Parameter	Type		Description
ContentMode	Longint	→	Define the mode of the description of report (for tables) Default is to describe all if this parameter is omitted
CommentPointer	Pointer	→	Pointer to a text from the Host DB to be included in the report

Description

The **aa4D_M_CreateReport_faceless** command allows you to create a report within the current process of your Host database. It was mainly designed as a tool to debug your code, when you need for example to get details on the memory and cache usage at a particular condition, or when repeating in a loop a part of your code to detect memory leaks. (it is usual to hide the Table list via the ContentMode parameter in this case).

aa4D_M_Get_Build_4D_Text_call

aa4D_M_Get_Build_4D_Text_call ({ContentPointer}; { PathPointer})

Parameter	Type		Description
ContentPointer	Pointer	→	Pointer to a text variable
PathPointer	Pointer	→	Pointer to a text variable containing the full path of the file.

Description

The **aa4D_M_Get_Build_4D_Text_call** command allows you to display (or retrieve) the detail of the version and build number of an executable or a plug-in.
It works cross-platform, meaning that you can get the build number of an executable file on Mac OS,
or parse the "Info.plist" of a Mac Application on Windows.
(When you parse a file that contains a "/contents/info.plist", this is the file to select in the dialog to retrieve the information).
If no parameter is passed, a dialog to select the file is displayed, and the result is displayed in an Alert.
If you pass a pointer to a text variable as a first parameter, the content of the Alert will be redirected to the pointed text variable.

If you pass a pointer to a text variable as a second parameter, the pointed text variable must contains the full pathname of the file to be examined.

aa4D_NP_Get_Last_Server_Report

aa4D_NP_Get_Last_Server_Report

Description

The **aa4D_NP_Get_Last_Server_Report** command allows you to retrieve the last report created in the Folder_reports of the database, even from 4D Remote, and display it's content in a dialog.
You can then save it in your remote computer.
(if run in Local mode, it will retrieve the last report of your application).

aa4D_M_Report_CreateOnServerSee

aa4D_M_Report_CreateOnServerSee ({ContentMode}; {CommentPointer})

Parameter	Type		Description
ContentMode	Longint	→	Define the mode of the description of report (for tables) Default is to describe all if this parameter is omitted
CommentPointer	Pointer	→	Pointer to a text from the Host DB to be included in the report

Description

The **aa4D_M_Report_CreateOnServerSee** command allows you to create a report on the Server, and after a small wait, retrieve the last created report from 4D Server, and display it in the same dialog as when creating a report in Local mode: But from there, you will be able to save it locally.

(If no previous report was created on the Server, this command might miss the last created report and retrieve the previous one. This is because the first report created after the Startup of the application is doing more processing than for later reports, completing a small document named "Array_profiler.txt".
In this case, just execute "aa4D_NP_Get_Last_Server_Report" to be sure to retrieve the last created report.)

aa4D_NP_Report_Analyse_Display

aa4D_NP_Report_Analyse_Display ({Report Name or Full path}; {ReportContent})

Parameter	Type		Description
Report Name or Full path	Text	→	Define in minute(s) the delay between two reports
ContentMode	Text	→	Define the mode of the description of report (for tables)

Description

The **aa4D aa4D_NP_Report_Analyse_Display** command allows you to display a dialog that let you compare the (last created) report with an earlier one (or a later one), and see for example the variation of records in each table (by value and percentage), to get an idea of what is moving most.

If you path a Full path in $1, the Report will be directly parsed to retrieve it's content.
If you path the Report Content in $2, the content will be parsed without opening the report file.

aa4D_NP_Report_Compare_Display

aa4D_NP_Report_Compare_Display

Description

The **aa4D_NP_Report_Compare_Display** command allows you to display a dialog wih a List box that let you compare the main usage (Number of users, Proces, etc..) and main memory and cache evolutions.
From this dialog, you can export the results in an external document, including Excel.
This is the main dialog to analyse external reports. (see some details later in the documentation).

aa4D_NP_Report_Manage_Display

aa4D_NP_Report_Manage_Display

Description

The **aa4D_NP_Report_Manage_Display** command allows you to display a dialog wih a List box that let you compare the main usage (Number of users, Proces, etc..) and main memory and cache evolutions.
From this dialog, you can export the results in an external document, including Excel.
This is the main dialog to analyse external reports

aa4D_NP_Shedule_Report_Server

aa4D_NP_Shedule_Report_Server ({DelayBetween}; {ContentMode}; {CommentPointer})

Parameter	Type		Description
DelayBetween	Longint	→	Define in minute(s) the delay between two reports
ContentMode	Longint	→	Define the mode of the description of report (for tables) Default is to describe all if this parameter is omitted
CommentPointer	Pointer	→	Pointer to a text to be included in the report

Description

The **aa4D_NP_Shedule_Report_Server** command allows you to manage a Stored procedure on 4D Server, that will create a new report every N minutes, so you keep a trace of the usage of your 4D Server.
Later on, you can check the evolution of this usage by parsing the reports via the Compare dialog (called by the shared method: aa4D_NP_Report_Compare_Display)

If you pass no parameter, the default value of 5 minutes will be used.
To stop the stored procedure, just pass 0 as the first parameter when calling this method.

aa4D_NP_Util_CreateReport

aa4D_NP_Util_CreateReport ({FacelessMode}; {ContentMode}; {CommentPointer})

Parameter	Type		Description
FacelessMode	Longint	→	If >0, hide the dialog after creating the report.
ContentMode	Longint	→	Define the mode of the description of report (for tables) Default is to describe all if this parameter is omitted
CommentPointer	Pointer	→	Pointer to a text to be included in the report

Description

The **aa4D_NP_Util_CreateReport** command is the main command of this component:
This is the one you will ask your customers to use when you need to get a picture of the configuration, settings and volume of the database.
If you ask to execute it on 4D Server, there will be no dialog indicating when it is done, but waiting for a few minutes will let you be sure that the report was created on the Server.
If executed in Local Mode or running Stand-Alone, a dialog will be displayed with the content of the report: from this dialog, you will be able to access the created text file, and send it to a tech support for example.

The first time a report is created:

- A folder named "Folder_Reports" will be created next to the Data file (if it does not exist)
- Some Launch External process will parse some information about the computer
- A document "Array_Profiler.txt" will be created in the "Folder_Reports", with some constant values retrieved (only the first time after starting the 4D Application).

(See a large report content example later in the documentation).

Please pay some attention at the Attention section of the report: we try to remind here what should be noticed (and in some case changed).

aa4D_NP_Util_CreateReport_Serv

aa4D_NP_Util_CreateReport_Serv ({FacelessMode}; {ContentMode}; {CommentPointer})

Parameter	Type		Description
FacelessMode	Longint	→	If >0, hide the dialog after creating the report.
ContentMode	Longint	→	Define the mode of the description of report (for tables) Default is to describe all if this parameter is omitted
CommentPointer	Pointer	→	Pointer to a text to be included in the report

Description

The **aa4D_NP_Util_CreateReport_Serv** command is very similar to the aa4D_NP_Util_CreateReport command.
The only difference is that it will create the report on 4D Server when executed in Remote application.

As the report is executed on 4D Server, there will be no dialog indicating when it is done, but waiting for a few minutes will let you be sure that the report was created on the Server.

If executed in Local Mode or running Stand-Alone, this command behave like aa4D_NP_Util_CreateReport.

DESCRIPTION OF THE MAIN PARAMETER FOR THE CREATION OF REPORTS:

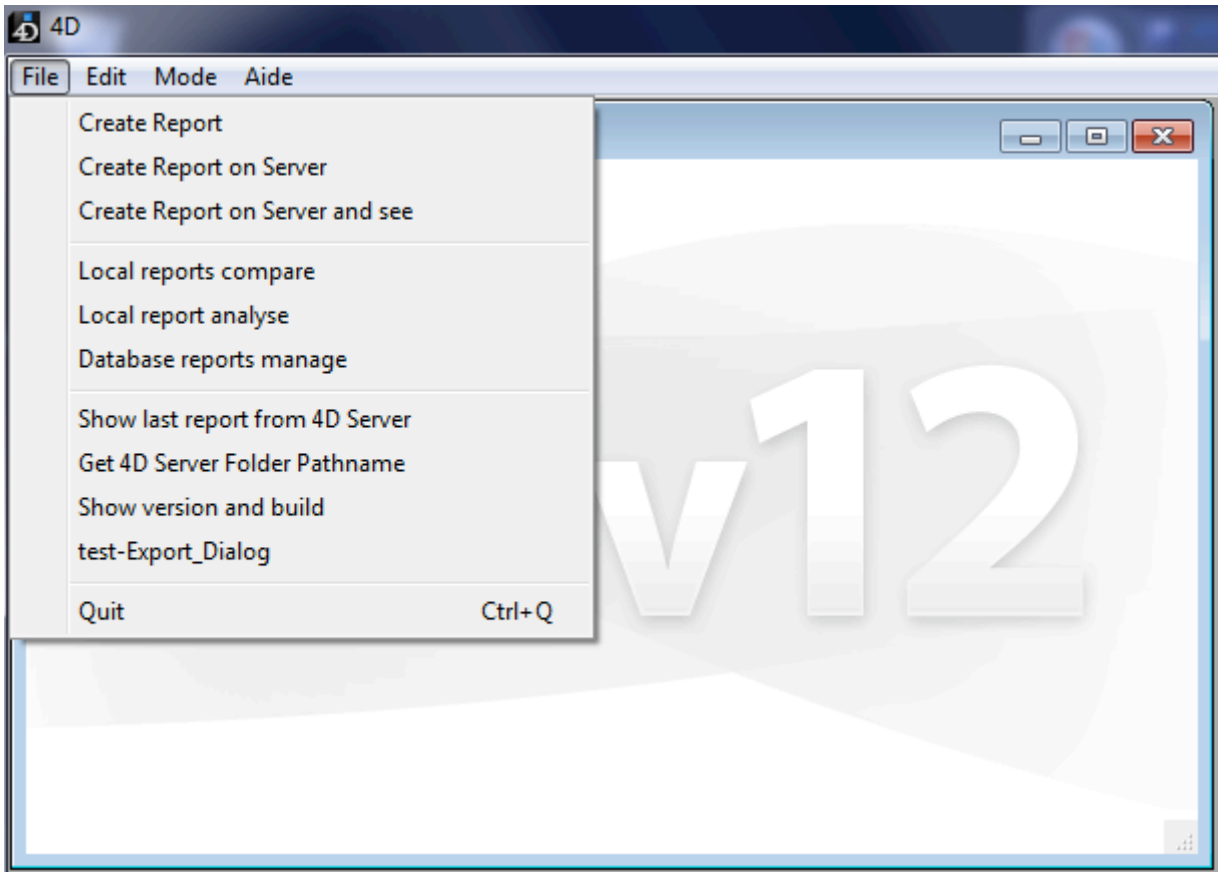
(For all shared methods that create a report, most often the second (optional) parameter)

- ContentMode Value (Type**: Longint)** Define the mode of the description of report (for tables)

Main parameter value for the creation of reports:	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Count total number of records…	X	X	X	X	X	X	X	X	X	X	X	X
except for invisible tables	X				X	X	X	X	X			
Hide Table List	X	X										
(Default if parameter omitted) Show all tables list, with all structure details			**X**									
Show Table List:			X	X	X	X	X	X	X	X	X	X
Show Table number			X	X	X	X	X	X	X	X	X	X
Show Table name			X	X	X	X	X	X	X	X	X	X
Show number of records of the table			X	X	X	X	X	X	X	X	X	X
Show number of indexed fields			X			X	X	X	X	X	X	X
Show number of subtable (if any)			X			X		X			X	
Show triggers			X			X	X			X		

DISCOVER THE COMPONENT RUN IN STAND-ALONE:

(Opening directly the component database, and creating once a requested data file).



Using the custom menu when run in Stand-Alone, you can check how it works:

First section of the File menu:

Creation of one report

- **Create Report** with execute the shared method : "aa4D_NP_Util_CreateReport" (without parameter) generating after a few seconds a new report displayed inside a dialog:
- **Create Report on Server** (When executed on 4D in Remote mode), will execute the shared method : "aa4D_NP_Util_CreateReport_Serv", that will generate a new report on 4D Server (no warning on 4D Remote when the report is created).

  (When executed in local mode, will behave like **Create Report**)

- **Create Report on Server and See** (When executed on 4D in Remote mode), will execute the shared method : "aa4D_M_Report_CreateOnServerSee" (without parameter)

  (generating after a few seconds a new report displayed inside a dialog

Second section of the File menu:

- **Local reports compare** will execute the shared method: "aa4D_NP_Report_Compare_Display" that display a new dialog (see captures in next page)
- **Local reports analyse** will execute the shared method: "aa4D_NP_Report_Analyse_Display" that display another new dialog
- **Database reports manage** will execute the shared method: "aa4D_NP_Report_Manage_Display" that display another new dialog

Third section of the File menu:

- **Show last report from 4D Server** will execute the shared method: "aa4D_NP_Get_Last_Server_Report" that will display in an alert the last generated report on 4D Server (When executed on 4D in Remote mode), or on 4D in Stand-Alone mode.
- **Get 4D Server Folder Pathname** will display in an alert the full path of the folder "Folder_Reports" (located next to the (Host) data file).

  (As there is currently no way to change this location or the name of the current reports folder, no need to use a shared method to retrieve it: in you Host database code, just complete the parent Folder of the data file with "Folder_Reports".)

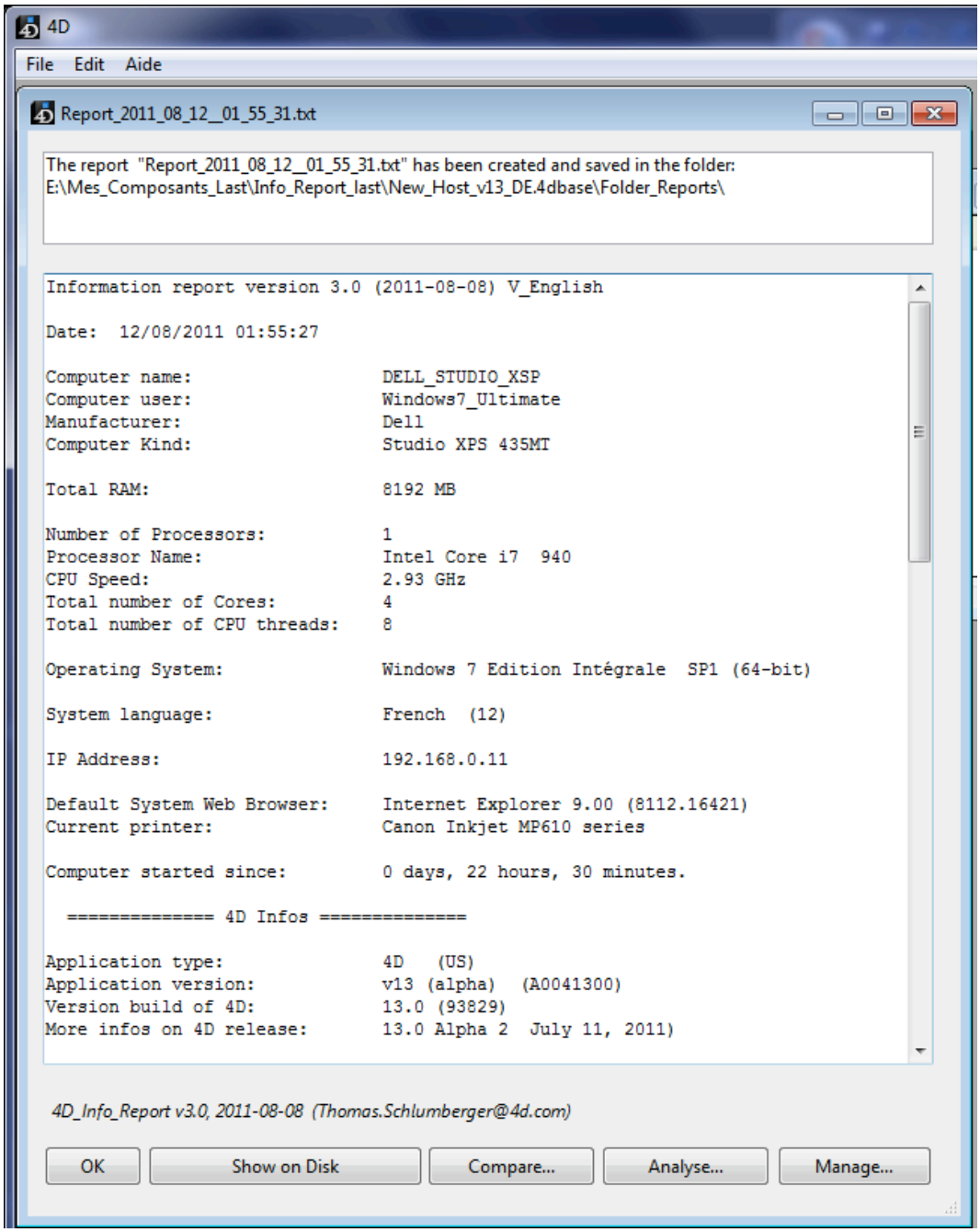
- **Show version and build** will execute the shared method: "aa4D_M_Get_Build_4D_Text_call" that will let you select in a dialog a 4D application or a plugin bundle, and display in an alert:

  - The version of the file,
  - The build number of the file (if it exists)
  - More infos (public version) corresponding to the build number and version,
  - The full path of the selected file.

- **test-Export_Dialog** will display in a new dialog some settings to generate in a new way the export of the values or the current "Folder_Reports"

  (still at a development stage currently in version 3, but the export works if not on remote mode)

When executing the shared method "aa4D_NP_Util_CreateReport", after a few seconds (the first time), a new dialog is displayed including the generated report content:

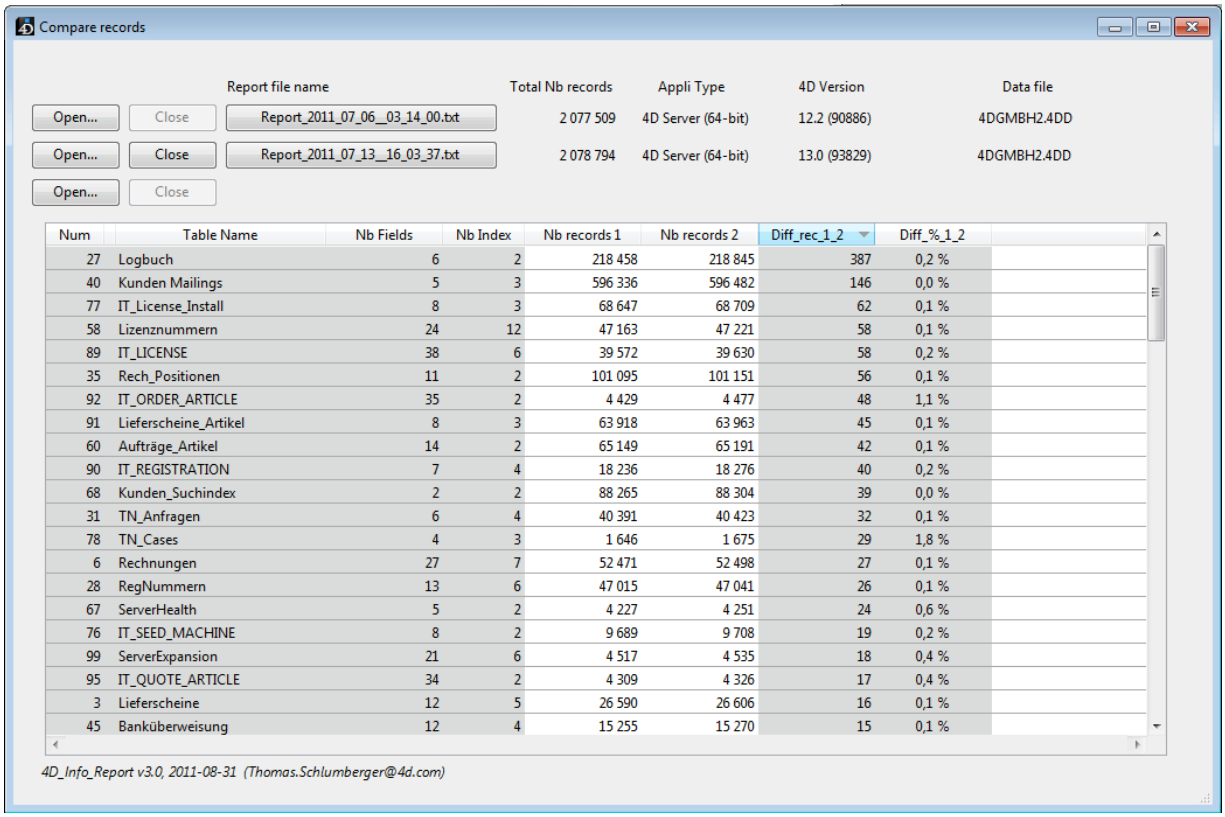


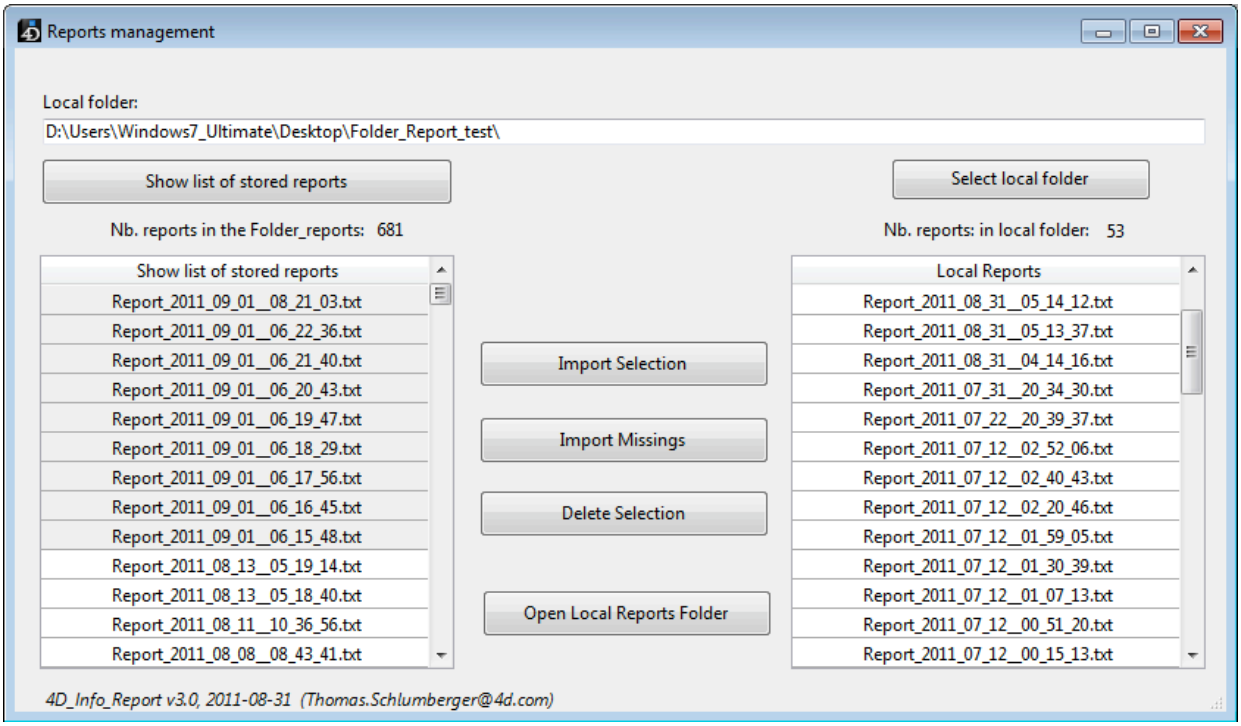
The “Show on Disk” button (when in Local mode) will open the content of the “Folder_Reports”, with the last report selected.

When in Remote mode, if the report was not generated locally, then you can select a local folder to save the imported reports.

The “Analyse” button will display a new dialog, allowing a simple comparision with one or two other reports. (use the shared method "aa4D_NP_Report_Analyse_Display").

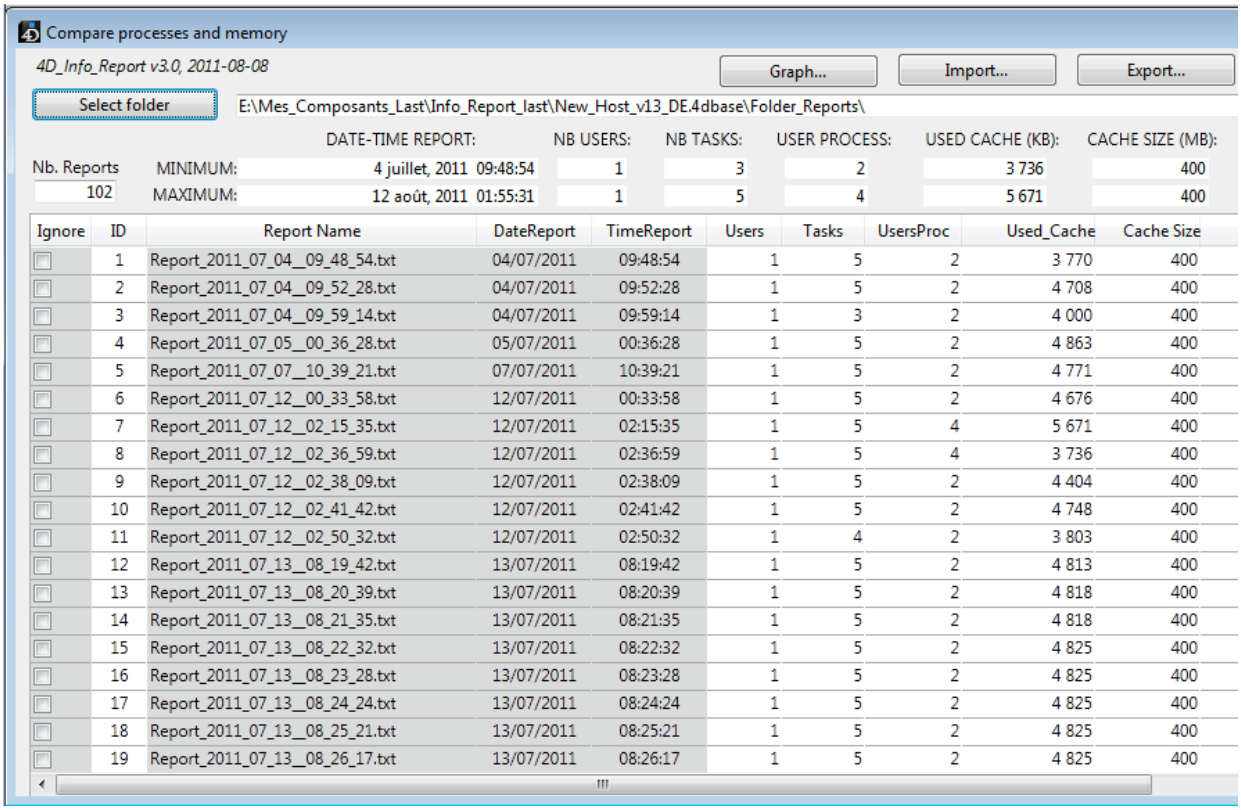
The “Manage” button will display a new dialog, allowing (in Remote mode) to import reports stored on 4D Server, and set the stored procedure that create a new report every N minutes.





The “Compare” button will display a new dialog, allowing to compare the main values of saved reports in the same folder.

(same as when executing the shared method “aa4D_NP_Report_Compare_Display")



Click the “Select folder” button to confirm the opening of the last opened folder, or to select another folder containing reports (can be reports provided by your customers when they have an issue).

When a folder is selected, a parsing of the reports inside this folder will populate arrays in the List box about the usage of the database.

The “Import” button allows to import a Blob generated via the Export button, that only contains the values used in the Listbox, so it is much faster than processing all the reports again.

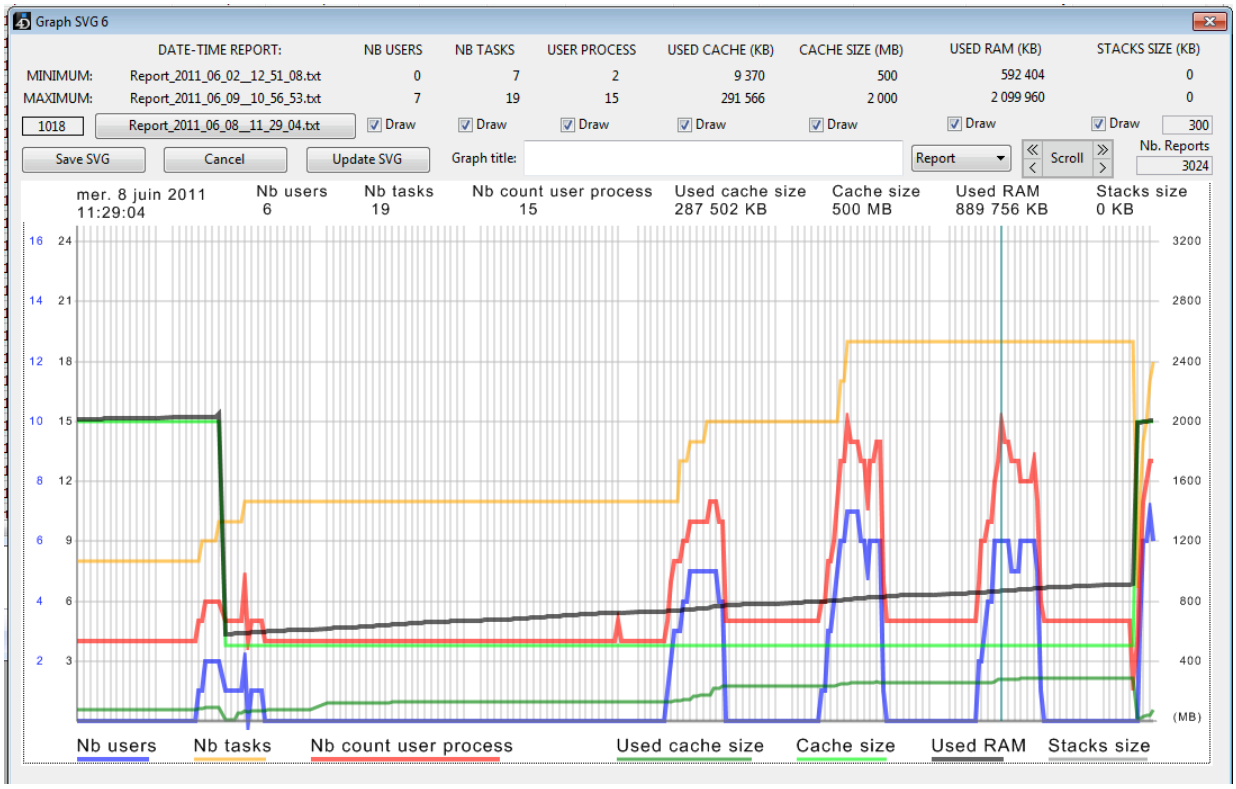
The “Export” button allows to select a format and a name to export the content of these arrays.

(you can for example generate an Excel document (.xml format compatible with Excel 2003 or later)).

- If you double-click one line of the List box, if the document was parsed, it will open in a window.
- If you click one or more "Ignore" checkbox, the corresponding report will be ignored in the MINIMUM and MAXIMUM values calculation, and also when choosing the "Graph..." button.

(To modify many "Ignore" checkboxes at once, just set a selection of Row in the Listbox, and click one of the corresponding "Ignore" checkbox).

The “Graph...” button will display a new dialog, allowing to display these main values in an SVG graph.



- Checking or unchecking the "Draw" checkboxes allow to display or not the values of the corresponding kind of values.

There are two families of values:

- The first one (on the left) is relating to: the number of users, the number of tasks, the number of user process.
- The second one (on the right) corresponds to the memory usage, and is relating to: the used cache size, the cache size, the used ram, the total stack size. (Scaled in the right of the graph).

If you click in the graph, the nearest report will be highlighted, and the values of this report will be displayed in the top of the graph: then just move the mouse X on the left or the right to get these values updated.

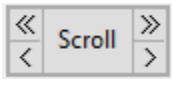
(to escape this mode, move the X position on the left or on the right of the graph part displaying the polygons.

If you click again, the selected report will be frozen, until you click again in another position.

If you double-click, the corresponding report content will be displayed in reduced window.

(when a report name is displayed under the MINIMUM and MAXIMUM report name, you can click the button containing it's name to open this report)

You can use the four kinds of arrow buttons in the scroll part to set a scroll on Timer:



(Double arrow sign will speed-up the scroll compared to single arrow sign).

THE FORMAT AND CONTENT OF THE GENERATED REPORTS:

1/ All generated files have a structured name:

For example, this report was created in May 13, 2011: “Report_2011_05_13__16_37_16.txt”

All reports name begin with “Report_”, followed by a Time stamp formatted this way:

“YYYY_MM_DD__HH_MM_SS” (Y=year, M=Month, D=day, H=hour, M=minute, S=second)

If you need to later process a report via the component, you cannot change the name of the document ("Report_", length, etc).

2/ Each document is generated with a UTF-8 format, with all lines ending with CR+LF.

They are cross platform (Mac and Windows).

(There is a 3 byte BOM at the beginning of the document)

3/ The content of the report is allows the component to find searched items.

(older reports created by the component "aa4D_Report" might have a different structure, but the compatibility of their structure is handled)

Also, Reports created with French releases of 4D will have French labels, with parsing compatibility.

A/ The Header: (general description of the Computer):

```
Information report version 3.0 (2011-08-08) V_English

Date:          24/08/2011     16:37:12  // (Date format depend of the O.S. language setting)

Computer name:                          IP-0A6801EB
Computer user:                          Administrator
Manufacturer:                           Xen
Computer Kind:                          HVM domU

Total RAM:                              69888 MB

Number of Processors:                   2
Processor Name:                         Intel Xeon  X5550
CPU Speed:                              2.67 GHz
Total number of Cores:                  4
Total number of CPU threads:            8

Operating System:                       Windows Server 2008 Datacenter  SP2 (64-bit)

System language:                        French  (12)

IP Address:                             10.104.1.235

Default System Web Browser:             Internet Explorer 8.00 (6001.18943)
Current printer:                        HP P3005 - Q&A (redirected 2)

Computer started since:                 0 days, 8 hours, 43 minutes.
```

B/ Description of the 4D Application: (version, build, 64-bit?):

```
         ============== 4D Infos ==============

Application type:               4D Server (64-bit)   (US)
Application version:            v12 release 2  (F0031220)
Version build of 4D:            12.2 (94807)
More infos on 4D release:       12.2 Hotfix 3   (August 10, 2011)
```

```
Mode:                                   Compiled

Application:
"D:\Tests\4D\v12_2_HF3\4D Server 64-bit\4D Server.exe"

PICTURE CODEC LIST: (11)
  .4pct .jpg .png .bmp .gif .tif .emf .pict .pdf .svg .wdp
```

C/ Description of the Database (size, location, plugins, components, settings):

```
  ========= 4D Database =========

Main UUID:     9F463893A87C478F9F3127EB3AA30DE0       (Structure)
Main UUID:     9F463893A87C478F9F3127EB3AA30DE0       (Data)


Structure File:
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.4DB"
       Size:          6 400 KB               Modif:       2011_08_24__16_29_56

Structure File: (index)
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.4DIndy"
       Size:          320 KB           Modif:       2011_08_24__12_39_18

Data File:
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.4DD"
       Size:          14 654 464 KB            Modif:       2011_08_24__16_37_06

Data File: (index)
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.4DIndx"
       Size:          55 680 KB               Modif:       2011_08_24__16_28_55

Match File:
"D:\BenchSummit\Test64-500\Bench-64-bit-v12.Match"
       Size:          17 KB            Modif:       2011_08_24__12_40_38

  <------- Plugins Bundle ------

    --- Database ---
API Pack.bundle               Executable Key: API Pack              Version Key: 2.3.2

    --- 4D Server Application ---
 4D InternetCommands.bundle     Executable Key: 4D InternetCommands    Short Version Key: 12.2
(94807)
 4D View.bundle                 Executable Key: 4D View                Short Version Key: 12.2
(94807)
 4D Write.bundle                Executable Key: 4D Write               Short Version Key: 12.2
(94807)

    --- (Plugin Server 64-bit) ---
 4D InternetCommands.bundle

   ------- Plugins Bundle ------>

   <------ Components -------

 4D SVG
 4D Widgets
 4D_Info_Report_v3

   ------- Components ------>


         <------ Backup infos -------

Backup.XML location:
"\Preferences\Backup\Backup.XML"

Log File:
 (No Log File found.)


Last Backup:
Last Backup Infos:
```

```
 ( Last Backup date & time (in Backup.XML) : 0000_01_01__00_00_00 )

 ( GET BACKUP INFORMATION;2 : Err.: 0 : No detected error.)

 ( GET BACKUP INFORMATION;0 : 0000_00_00__00_00_00 )


Next Backup:

 ( GET BACKUP INFORMATION;4 : 2011_08_31__03_47_37 )


Preferences Backup Advanced:

<Transaction>
       WaitForEndOfTransaction:  False
       Timeout: 0000_00_00__00_03_00

<BackupFailure>
       TryBackupAtTheNextScheduledDate:  False
       TryToBackupAfter: 0000_00_00__00_01_00
       AbortIfBackupFail: False
       RetryCountBeforeAbort: 5

<Automatic...>
       AutomaticRestore:  False
       AutomaticLogIntegration: True
       AutomaticRestart: True


         ------- Backup infos ------>


Involved volume(s) list (free space):

Volume  C:\ (Operating System)               13 297 MB
Volume  D:\                                  348 809 MB


        ============================


         <------ Comment -------

(content of a pointed text variable ($3) if passed).

         ------- Comment ------>


Maximum number of users (license) :  1301

User list size:                      2
Group list size:                     0

Web: (The Web Server is not started)
Count tasks:                         1248
Count user processes:                1245
Number of users:                     735

    ------ Database parameters ------

Database Cache:                      10 240 MB
4D Server Timeout:                   30 Minute(s)
4D Remote Timeout:                   30 Minute(s)

Selector 28 (4D Server Log Recording):                    0
Selector 34 (Debug Log Recording):                        0
Selector 53 (stack size of preemptive thread):            256 KB
Selector 54 (Idle Connections Timeout):                   20 seconds.
Selector 61 (Maximum Temporary Memory Size):              0 (Max.)
Selector 65 (CacheLog Recording) :                        1
Selector 66 (Cache Clearing size) :                       1 048 576 KB
Selector 67 (Time-out forced disconnections) :            3 seconds.
Selector 68 (Log disconnections errors) :                 1
```

```
    ------ other infos via GET CACHE STATISTICS: ------

    Used cache Size :                  8 172 835 KB
    Free Memory :                     56 092 928 KB
    Used physical memory :            11 286 860 KB
    Used virtual memory :             11 977 700 KB
    Size of stacks :                     518 525 KB


Sheduler (CPU settings):
4D Server:       0 - 8 - 0     Medium (Standard setting), recommended.

        ********************** Attention section **********************

Attention : (any Attention content, that will create an "Attention_Report.txt" file).
Information : 4D Server (64-bit)   (US)  12.2 (94807)
Information : Administration window opened on Server
```

D/ Infos on the Tables (num, name, nb records, fields, indexed fields, subtables, triggers):

```
         ========== TABLES ===========

(Triggers description: N = On saving New record, E = On saving Existing record,
D = On Deleting record)

Total number of records:                    58 232 680

Number of tables:     271            Number of valid tables:       271

Total number of valid fields:         1787

     Num  Name                        Nb of Records Fields (Index | SubT.) Trig

T.:    3  Cities  ---------------------      43 607     4  (   1  |    - ) ----

T.:    2  Companies  ------------------      68 850     5  (   2  |    - ) ----

T.:  110  Companies0005  --------------      40 000     5  (   0  |    - ) ----

T.:  247  Companies0006  --------------      70 000     5  (   0  |    - ) ----

T.:  245  Companies0007  --------------      30 000     5  (   0  |    - ) ----

T.:  243  Companies0008  --------------      40 000     5  (   0  |    - ) ----

T.:  241  Companies0009  --------------      60 000     5  (   0  |    - ) ----

T.:  239  Companies0010  --------------      50 000     5  (   0  |    - ) ----

T.:  237  Companies0011  --------------      80 000     5  (   0  |    - ) ----
```

[…]

The table description is optional, you can avoid listing them, hide structure information (only showing the number of records), hide “Invisible” tables, etc…

The total number of records will always be reported. (with option on Invisible tables).

DEPLOYMENT OF THE COMPONENT:

To use the component, add it to the “Components” folder next to the Structure file of your application.

(Create this folder if it does not exist)

This component does not create Table(s) in the Host database. The Host database remains unchanged. Only the folder “Folder_reports” is added next to the data file, and filled with created reports.

ACCESSING THE CODE OF THE COMPONENT FROM THE HOST DATABASE.

While in development, or if you allow access to the execution of methods, the shared methods of the components can be executed directly.

(you can also see them in the Explorer (project methods), after deploying the component list, and selecting this component)

If you plan to deploy your application, we recommend to avoid tokenizing the shared methods of the components in your Host database, in case you want to remove the usage of a component.

You can use this example of code to avoid the “hard link” of the component with your Host database

```
  ` ----- to execute a shared method and be able to compile after removing the component ---
ARRAY TEXT($at_Components;0)
COMPONENT LIST($at_Components)
If (Find in array($at_Components;"4D_Info_Report@")>0) ` was "4D_Info_Report@" with previous
name.
       EXECUTE METHOD("aa4D_NP_Shedule_Report_Server";*;5;-2)  ` or another shared method.
End if
```

You can also set an interprocess Boolean to avoid parsing the Component list each time, like <>vb_Is_Component_Report_ready, and only test at startup if the component is available, and use this alternative code:

```
If (<>vb_Is_Component_Report_ready)
   EXECUTE METHOD("aa4D_NP_Shedule_Report_Server";*;5;-2)  ` or another shared method.
     ` EXECUTE METHOD("aa4D_NP_Shedule_Reports_Server";*;0) ` to stop the stored procedure.
End if
```

- Remind that all shared methods beginning with “aa4D_NP” create a new process, so you don’t need to create one in the Host database to execute these methods. Also, via some dialogs of the component, you can execute directly some other shared methods (if allowed).
- You can limit the access of reports stored in the Server, or the right to set or change the setting of the stored procedure ("aa4D_NP_Shedule_Report_Server”) from a remote user, via this code to implement, as the component execute this shared Host method if it exists, and expect a True boolean result :

```
     ` Host method: aa4D_M_Host_Allow_Report_access (example code)
C_BLOB($1)
C_BOOLEAN($0)
C_BLOB($vxBlob)
C_BOOLEAN($Allow)
C_LONGINT($Offset)
If (Count parameters>=1)
    If (Type($1)=Is BLOB )
        $vxBlob:=$1
        ARRAY TEXT($at_infos;0)
        BLOB TO VARIABLE($vxBlob;$at_infos;$Offset)
          ` - ($at_infos{1}:=Current user)
          ` - ($at_infos{2}:=(local) I.P. address
          ` - ($at_infos{3}:=Current machine)
          ` - ($at_infos{4}:=Current machine owner)
          ` --- set your own conditions, depending of the 4 arrays elements ---
          ` ...
        ALERT("aa4D_M_Host_Allow_Report_access OK")  ` Comment this line after tests.
        $Allow:=True
    End if
End if
$0:=$Allow
```

It is up to you to design the conditions test (for example limiting this access to local network users, such Current user, etc…

We recommend using the “4D_Info_Report_Host_Template_v3” database

(provided as a 4D v11 SQL interpreted database)

To get some help on implementing the component in your Host database.

ACKNOWLEDGMENT

4D Info Report Component was written by Thomas Schlumberger, 4D SAS, France.

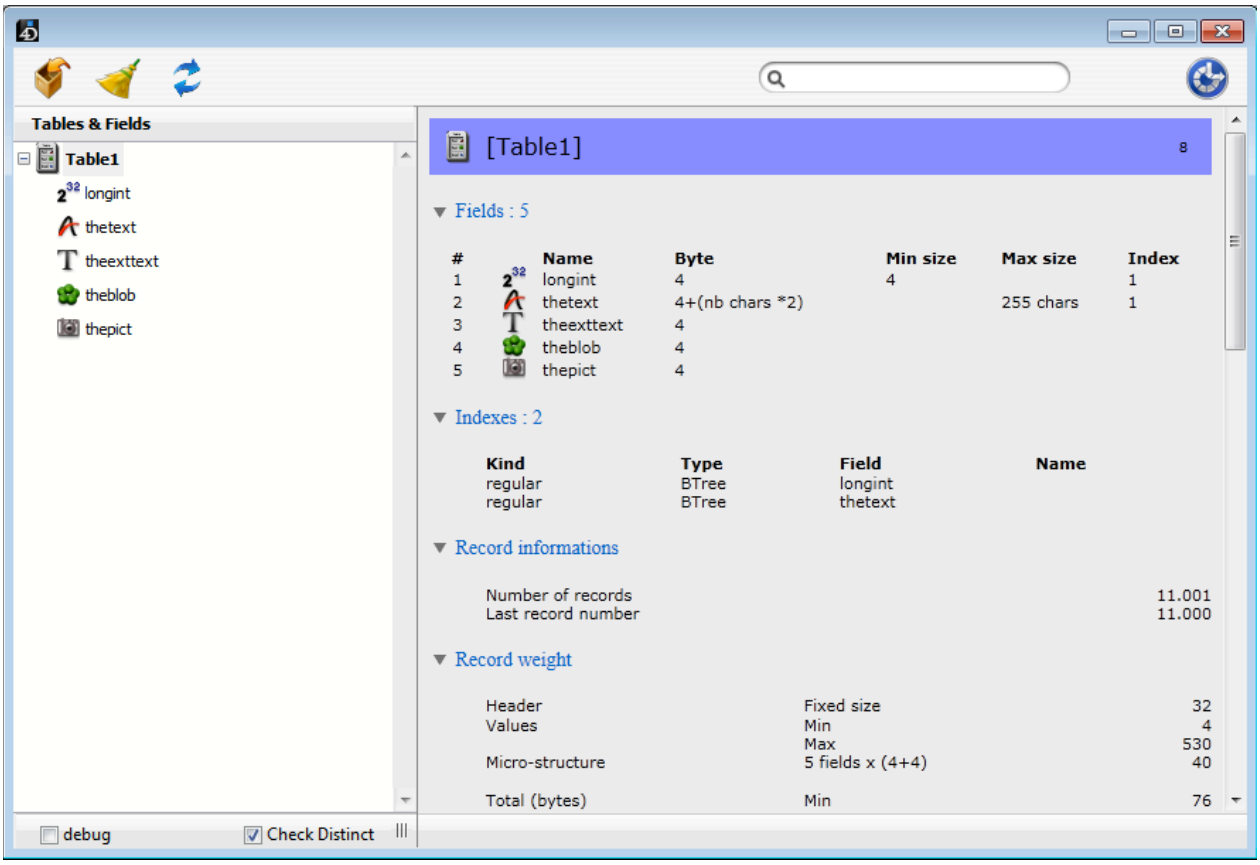
4D POP DATA ANALYZER

4D Pop DataAnalyzer is a component to analyze the size and content of 4D datafiles. This tool helps to understand the size minimum and maxium requirements of a record and analyses the size requirement depending of the real data.

Note: the component is not part of the “4D Pop” component group, don’t be confused by the name. But it has the same author…

The component is delivered as v12 version, but section Usage explains how to use it with datafiles from v11, v12 or v13.

OVERVIEW



Selecting a table shows basic data, as table name, table number, list of fields, each with field type, storage size, min and max size and if an index exists:

[Table1] 8

▼ Fields : 5

#		Name	Byte	Min size	Max size	Index
1		longint	4	4		1
2		thetext	4+(nb chars *2)		255 chars	1
3		theexttext	4			
4		theblob	4			
5		thepict	4			

▼ Indexes : 2

Kind	Type	Field	Name
regular	BTree	longint	
regular	BTree	thetext	

The column “Byte” allows to calculate manually the required size, such as for an alpha field with 20 characters entered 4 + 20*2 = 44 byte. The column “Min size” lists if a field has an minimum size required for storage, similar for the column “Max size” for a maximal size.

The block Record informations lists the total number of records and the last record number (internal record number starts with 0). If there are 10.000 records and last record number is 20.000, there are 10.000 “holes” (=deleted records).

▼ Record informations

Number of records	11.001
Last record number	11.000

▼ Record weight

Header	Fixed size	32
Values	Min	4
	Max	530
Micro-structure	5 fields x (4+4)	40
Total (bytes)	Min	76
	Max	602
Total (blocks)	Min	1
	Max	5

The block Record weight calculates the total minimum and maximum bytes a record can theoretically use. The needed size is divided in 3 parts, Header, Micro-structure and Values. The header size of a record could be different depending of the 4D version (32 byte in v11-v13), but is independent of the number of fields in the record. This part, named “Micro-structure” describes the field used in the record and needs in v11-v13 8 byte * number of fields. Finally we have the data itself, where we can calculate the minimum and maximum size.

It is important to know that 4D calculates Text, Pictures and Blobs all as “blob”, kind of variable data from 0 byte to 2 GB. Their size is not calculated in “Record weight”, as it depends on the real data, but included in “Table weight” (see below).

Remember that 4D handles Alpha and Text identical, as long Text is stored inside the record. As a result ‘text’ fields may be displayed from Data Analyzer as Alpha, when stored in the record. In this case the value for max size is 0.

In the example displayed above a record will take anywhere from 76 to 602 byte. Similar as a hard disk is organized in sectors a 4D data file is organized in blocks, with a block size of 128 byte (in v11-v13). As a result each record of this table will take from 1 to 5 blocks.

(see: http://en.wikipedia.org/wiki/Block_%28data_storage%29)

Table weight analyzes the real data of the data file and calculates:

Record weight sum: Total in bytes of the data files used by records from the selected table. This is real size, such as 100 byte, even it needs minimum one block with 128 byte.

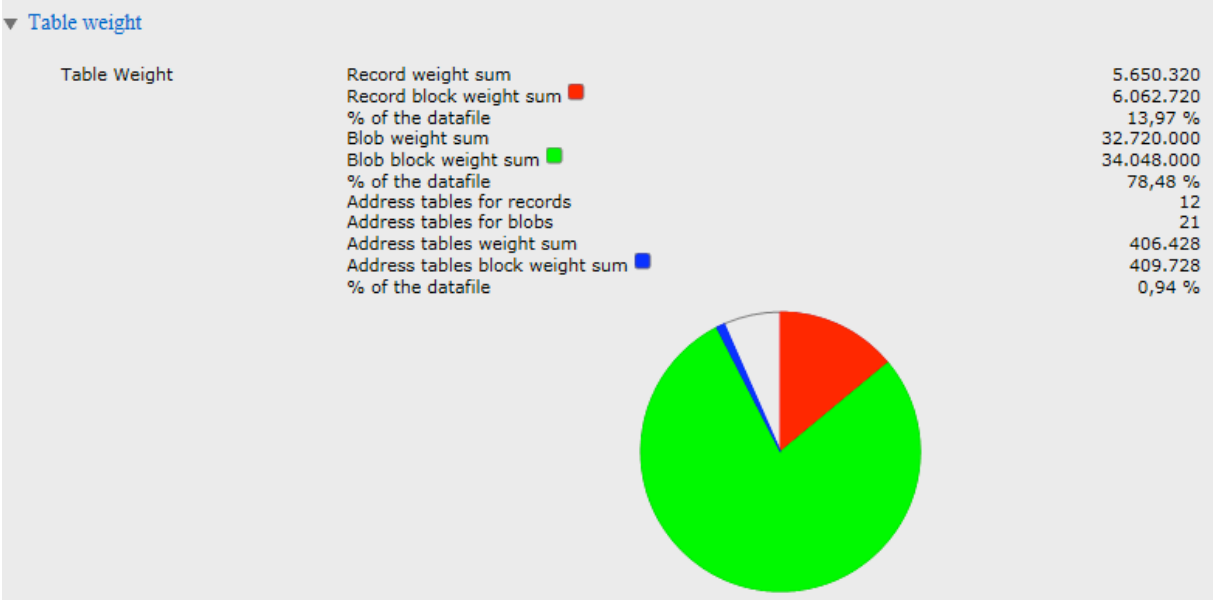
Record block weight sum: As above, but counting blocks, which is the real size needed in the data file. This value is displayed in red in the chart below, to show how much space of the data file is used from this part.

% of the datafile: calculation of the record block weight sum in relation to the total data file.

Blob weight sum: Total in bytes of the data file used by blobs (Text, Pictures or Blobs stored outside of the record) in real size.

Blob block weight sum: As above, but counting blocks. This value is displayed in green in the chart below, to show how much space of the data file is used from this part.

% of the datafile: calculation of the Blob block weight sum in relation to the total data file.

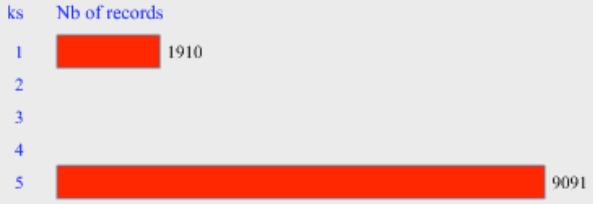


Finally the table needs Address tables, both for the records and the blobs, calculated as real data and in blocks, displayed in blue in the chart.

The next block includes statistical data based on the real record sizes, like smallest record, largest record and average, also counted in block sizes.

The chart below shows the distribution of the record sizes, in this example 1910 record needs exact 1 block while 9091 records needs 5 blocks (there is no record needing 2, 3 or 4 blocks):

Record address table	Nb of records	11.001
	Nb of entries	11.001
Record size	Min	92
	Average	513
	Max	602
Record bloc	Min	1
	Average	5
	Max	5



The last part displays information about fragmentation and real sizes of blobs.

Most 4D developers thinks “empty spaces” is the same as fragmentation. While it also could lead to speed reduces, fragmentation here is about “hard disk movement to read record 1 and then record 2”.

In a perfect world records would be written on disk as:

A01 – A02 – A03 – A04 – A05 – A06 - A07

(with A = table 1, 01 = record number 01, assuming each record needs 1 blob)

When 4D needs to read several records, such as Modify Selection or Selection to Array, it automatically does a “read ahead” operation, trying to read several records at once. This would work fine in the example above and reduce the amount of disk accesses drastically. In this example only one disk access is needed to access all records.

Now assume we add some characters into field “thetext” of record A02, so the record grows and need 2 blocks:

A01 – xxx – A03 – A04 – A05 – A06 - A07 – A02 – A02

As the old position did not allowed to grow, the record is moved to the end and the old space is marked “free”. If 4D tries to read A01 and A02, they cannot be read at once and (assuming the space between A01 and A02 is too big for a read head), it would be a “long jump” and 4D needs two disk access.

It would be simple if we enter some text for record A01:

A01 – A01 – A03 – A04 – A05 – A06 - A07 – A02 – A02

But let’s assume it is a little bit more text, then we get:

xxx – xxx – A03 – A04 – A05 – A06 - A07 – A02 – A02 – A01 – A01 – A01

Now we got a “back jump”. This is the worst case; in no way 4D could do a read ahead. To read A01, A02 and A03, it would need to do 3-disk access.

The next part of the analyze calculates both back jump and long jump for records and for blobs, as well as a percentage for both to get a quick overview:

Record fragmentation	Percent	8,26 %
	Nb back jump	909
	Nb long jump	0
Blob address table	Nb of blobs	20.000
	Nb of entries	20.000
Blob size	Min	1.036
	Average	1.636
	Max	4.036
Blob bloc	Min	9
	Average	13
	Max	32
Blob fragmentation	Percent	5,00 %
	Nb back jump	1000
	Nb long jump	0

Field Details

Selecting a field shows details about this object, such as name and number.

The part Basics informations displays:

Type: Field type, such as long

Nb distinct values: the number of distinct values in this data file. If there are many records and the field is not indexed it may need a while to retrieve this information (see check box “Check Distinct”)

Nb values: the total amount of records

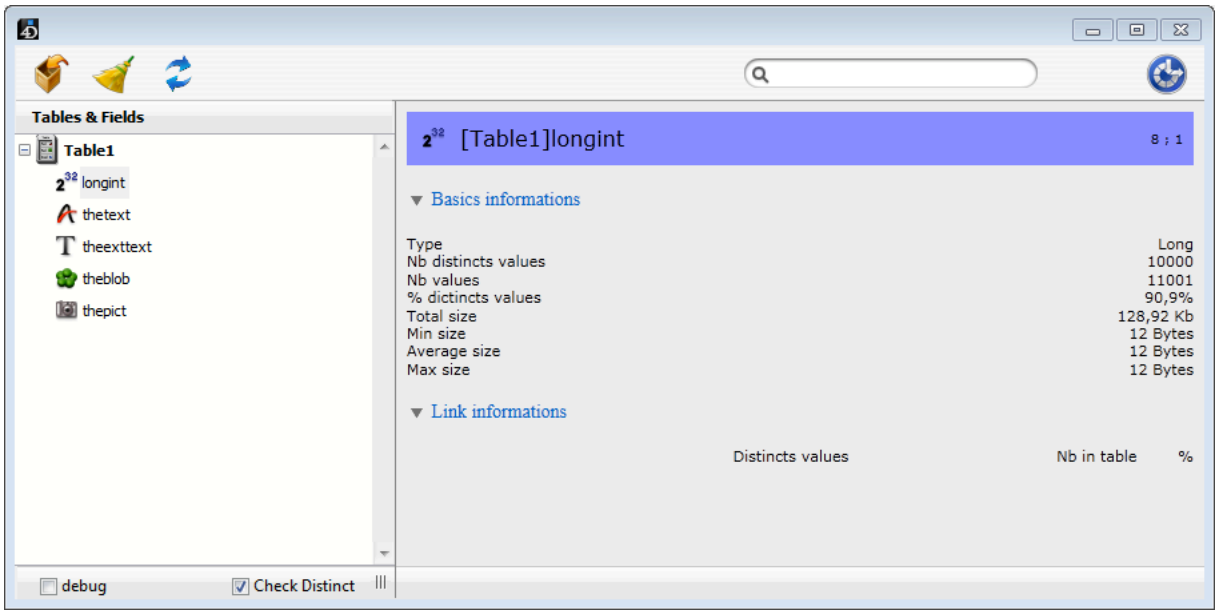
% distinct values: Percentage of distinct values. If the value is small it makes more sense to use Cluster Index than B-Tree index.

Total size: total space needed in the data file for this field, calculated based on the real values. If there are many records it could take a while to calculate, as each records needs to be loaded.

Min size: smallest record

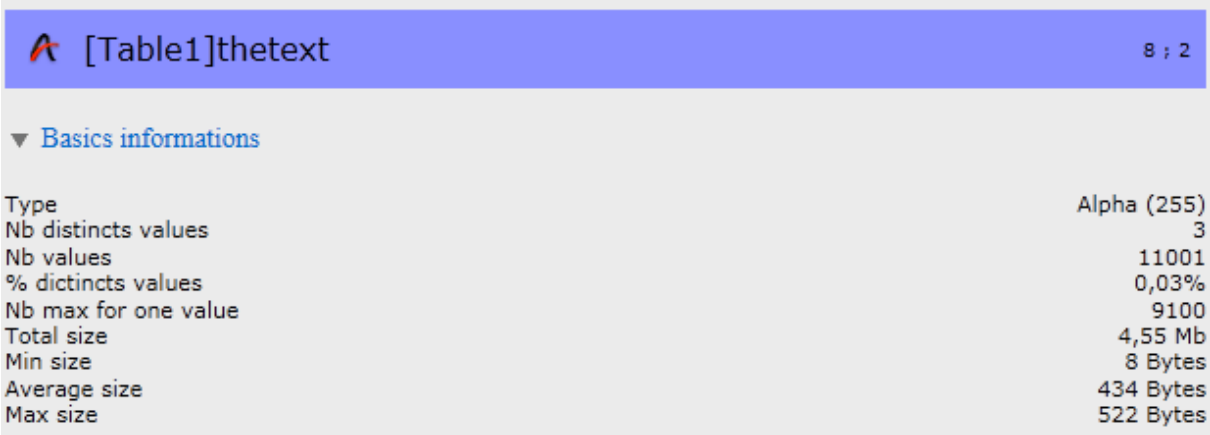
Average size: average (for the real values)

Max size: largest record



The part Link informations is empty in this example, as the data file has only one table, so there are no relations. We will show later an example.

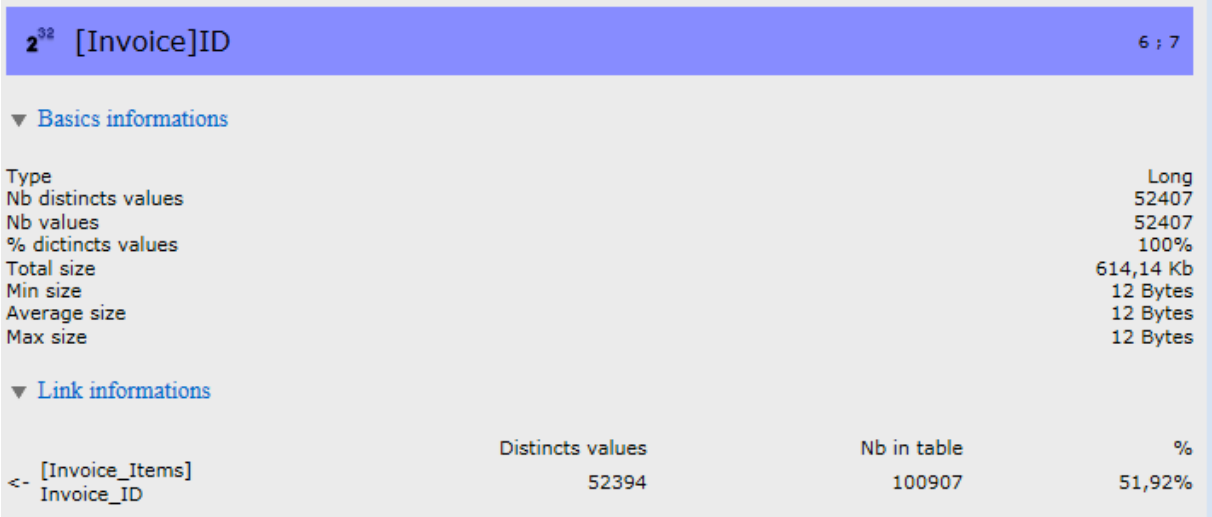
Now let's take a look on the alpha field:



Nb distinct values says that we have 3 different values in 11001 records, so most records shares the same value. Nb max for one value means that 9100 records have the same value. This data is a perfect example for a Cluster Index, while it uses a B-Tree (see the very first picture on page 1).

LINK INFORMATIONS

If the selected field has one or more releated fields, they are listed here:



USAGE

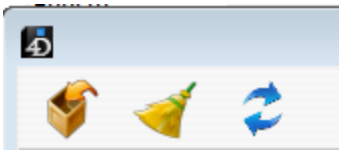
Export Structure XML

Duplicate Structure + data

If necessary upgrade v11 to v12 or v13. Component can be used with both v12 or v13.

Install component

Execute DataA_Explorer_Open:

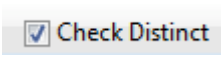


Click the very first button and select the exported structure .XML file to read data structure and index information's. This action will create 5 tables in the structure to store internal data

Click the 2nd button to delete the 5 tables and records (clean up)

The 3rd button will reparse the data file.

If your data file is very big (many records, many fields), you may want to disable checking for distinct values. In that case deselect the check box at the bottom:



Parsing the data file will be faster, but you miss some information's.

Important:

If you modify your structure (add/delete/modify fields, tables or index), use the Clean button to delete everything – then the import button to recreate and reparse the data. The component is not written to be used beside working with the structure, it was created to analyze and existing structure.

If you modify the data, such as deleting/adding records, use the 3rd button to reparse the data file. The component stores calculated data, such as how many field values are distinct, to speed up data presentation and analyzes the data only once per field (when you select a field for the first time). This button deletes all stored data.

Acknowledgment

4D Cache Log Recorder and 4D Cache Log Analyser was written by Olivier Deschanels and tested and enhanced by Aziz Elgohmari, 4D SAS, France.

DEBUG LOG

This is the first step to do for a case of "it crashes somewhere"

The Debug log shows every command executed on a Client or Server (server procedure, Trigger, Web method, etc). It is created inside the Logs folder, named "4DDebugLog_1.txt" such as:

```
1927 p=1 puid=23       (0) cmd: New list (DoFillHierList).
1927 p=1 puid=23       (0) cmd: APPEND TO LIST (DoFillHierList).
1927 p=1 puid=23       (0) cmd: List item position (DoFillHierList).
1927 p=1 puid=23       (0) cmd: GET LIST ITEM (DoFillHierList).
1927 p=1 puid=23       (0) cmd: SET LIST ITEM (DoFillHierList)
```

Notice that each line ends with a dot ".". Before executing a command 4D writes the line to disk – except the last dot. After executing the dot is added. This means if a line ends with a dot, the line was executed successful. If the last dot is missing, the application crashed during execution of this line. This is a huge help to track down random crashes.

The log also helps to understand what was executed – and how long 4D needed. The first value is the time in milliseconds. That could be a huge help to understand where/when/why a part of the application is slow – it shows where the time is spend.

The debuglog is enabled by language with: SET DATABASE PARAMETER(Debug Log Recording;1), the result is a logfile as above.

Using 2 as third parameter enhances the debug log with information about plugin calls. This list is quite confusing for several developers, as it contains many strange looking numbers:

```
25755 p=9 puid=12      (1) plugInName: 4D Write; externCall: -621.
25755 p=9 puid=12      (1) plugInName: 4D Write;
cmd: _4D Write; eventCode: 43.
```

These entries were created while displaying (and using) a form with a 4D Write area. The list shows two different kinds of entries, externCall and eventCode.

ExternCall means the plugin has called 4D and executed a 4D command. In this example -621:

```
#define  EX_PASTEBOARD_IS_DATA_AVAILABLE                    -621    //
PA_IsPasteboardDataAvailable
```

The 4D Plugin API contains a list of available entry points in the file named "EntryPoints.h". While the text (EX_PASTEBOARD_IS_DATA_AVAILABLE) gives an idea what is done, a detailed describtion can be found in the plugin API documentation unter (PA_IsPasteboardDataAvailable):

http://sources.4d.com/trac/4d_4dpluginapi/wiki/XKCMDUS.HTM

Clicking in the upper right corner on "Sources" allows to download "EntryPoints.h" as well as "PublicTypes.h", which describes all possible eventCode ID's. These files are also provided on the USB stick with this document (look inside the folder Lookup_Files).

```
eAE_UpdateEditCommands                         = 43
```

Using 2 as third parameter drastically increase the size of the debug log if you uses forms with plugin areas. As long you do not see issues with plugins, we recommend not to use this option, as the debug log grows a lot and is more difficult to read. But if you have issues with a plugin involved, the log gives an idea what's going on, what calls go from 4D to the plugin and vice versa. If a not or only difficult to reproduce crash occurs during plugins usage, you can at least see when it happens or what the plugin was doing, which could help the vendor to debug the problem.

Using 3 as third parameter is a greatly enhanced feature of v12, it lists in interpreted mode the parameter details:

```
2979 p=1 puid=3        (8) cmd: New list.
2979 p=1 puid=3        (8) cmd: APPEND TO LIST(3152;"Entry";2001).
2979 p=1 puid=3        (8) cmd: List item position(3150;2000).
2979 p=1 puid=3        (8) cmd: GET LIST ITEM(3150;2;$EntryRef;$EntryText;$sublist2).
2979 p=1 puid=3        (8) cmd: SET LIST ITEM(3150;2000;"Phone";2000;3152;False).
```

In compiled mode some items are still readable, like static values, but variable names are replaced with their internal addresses, which looks scrambled. Still the static values are a great help to understand which part of the code was processed.

```
8776 p=1 puid=23       (0) cmd: APPEND TO ARRAY(8çÿÿ;"Some text")
```

As the log file could grow very large, it could be difficult to find a specific part inside the log. Searching for method names could help. A very useful trick is to write your own comments inside the log, such as:

```
LOG EVENT(Into 4D Commands Log; "User selected menu Accounting/End of year statistic")
```

These entries will be inserted into the log, which helps to identify critical parts, as for benchmarks.

4D NETWORK LOG ANALYZER

RECALL ABOUT THE NETWORK LOG

The network request log includes a huge amount of detailed information's, about everything sent or received as network packet:

```
Log Session D4B4091574A8B14F8A66FE119C52DEF3
1 nth log opened on Tuesday, August 10, 2010 04:26:45 PM
time    task id component      proc    request   bytes in     bytes out      duration
08/10/10, 16:27:37     8       'dbmg'  2       11014   25      26      0
08/10/10, 16:27:37     8       'dbmg'  2       11059   103     2585    0
08/10/10, 16:27:37     16      'dbmg'  5       11059   103     32115   0
08/10/10, 16:27:37     12      'srv4'  2       86      285     30      86
08/10/10, 16:27:37     12      'srv4'  2       3       586     22      140
08/10/10, 16:27:37     12      'srv4'  2       87      16      10      38
08/10/10, 16:27:37     8       'dbmg'  2       11059   103     48507   0
08/10/10, 16:27:39     8       'dbmg'  2       11009   29      51      0
08/10/10, 16:27:39     8       'dbmg'  2       11014   46      26      0
08/10/10, 16:27:39     8       'dbmg'  2       11014   25      26      0
08/10/10, 16:27:39     8       'dbmg'  2       11059   103     48507   0
08/10/10, 16:27:40     8       'dbmg'  2       11014   25      26      0
```

Such a logfile could need hundreds of Megabytes or even Gigabytes, if you try to run it during the day, making it quite useless. But it becomes interesting if you want to track down specific issues or want to understand how much data is transferred over the network, as for benchmarks. So let's take a closer look to the content.

The request log can be enabled directly on the server computer in the admin window, tab Maintenance, by clicking "Start Request Log" or by using SET DATABASE PARAMETER(4D Server Log Recording;1) in a server process. The file "4DRequestsLog_1.txt" is created inside the "Logs" folder beside the structure.

The first column is clear. It is a readable timestamp. Then we have a task ID, a component, a "process info index" (here shorten to proc) and a request ID. We come back to this cryptic ID's in a minute.

The last 3 columns are easy to understand. How many bytes are sent for this request to 4D Server, answered with how many bytes and how long it took. Duration base depends of "component". It is microseconds for all answers from "srv4", while milliseconds for "dbmg" (1000 microseconds = 1 millisecond).

The other columns are more difficult to read, we need look up tables to understand request ID and to assign the process ID to a Client and process name.

OVERVIEW

The 4D Network Log Analyzer presents the data in a more readable form. Most important, it filters the data by grouping it. Packets are grouped in interval (5 minutes frames), or presented by largest or slowest package.

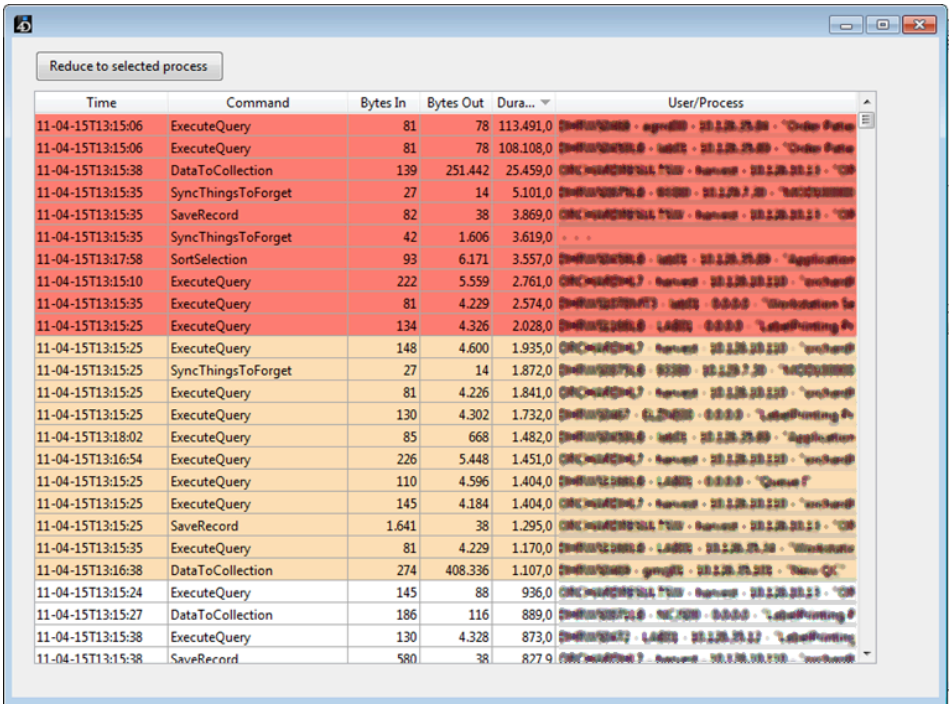
The network log is way too big to read it line by line, so we will focus on either slow or large packets and then try to understand when/where/why they was sent. Based on this idea, we do not read the log from start to end, but focus on special packets. As an example a 90 minutes period of a heavy used server (which had some huge response problems):

Time	# Packets	Total Size	Max Size	Max Answer Time	Avg Answer Time
11-04-15T11:45:	50723	70.192,8	403,6	15.319,0	5,5
11-04-15T11:50:	70102	103.058,7	403,6	3.620,0	0,8
11-04-15T11:55:	75150	140.673,6	405,8	56.176,0	4,1
11-04-15T12:00:	83454	124.243,3	403,6	2.496,0	0,7
11-04-15T12:05:	29186	42.725,8	538,5	1.840,0	0,8
11-04-15T12:50:	57791	119.400,0	8.796,3	1.513,0	0,8
11-04-15T12:55:	75708	114.560,3	818,7	827,0	0,6
11-04-15T13:00:	68102	97.890,1	405,8	2.075,0	0,5
11-04-15T13:05:	77714	112.846,6	646,6	2.340,0	0,8
11-04-15T13:10:	61246	99.609,0	1.761,4	38.767,0	75,5
11-04-15T13:15:	279421	653.157,5	4.381,9	113.491,0	1,4

The packets are grouped in 5 minutes packs, each is displayed with the total amount of packets received or sent, their total size, max size, max answer time and average answer time (both in milliseconds – 1000 ms = 1 second).

Packets with an answer time over 1 second or larger than 1 MB are displayed in yellow, over 2 seconds or 2 Megabyte in red. The example above is quite extreme as nearly everything is red, so needing attention.

Double click a line to display all packets from this 5 minute group – and use the header to sort by duration, bytes in, out or time:

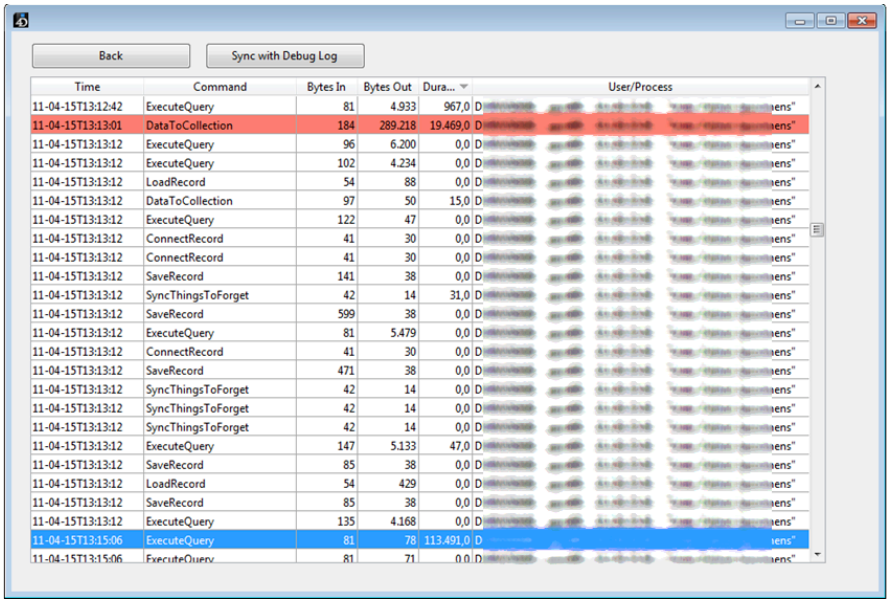


This screen, here sorted by duration, shows for each request:
Timestamp (in Second, Server time)
Request clear name (see below for details about request name)
Bytes In
Bytes Out
Duration (in Milliseconds)
User/Process (in Form Computer name – User name – IP – Process name

As the data above is real customer data, we needed to blur the IP/Process name.

This list gives us a clear “what to do next” list, as in this example two queries, each needing with 113/108 seconds way too much time, should be the first issue to check. Either they did a sequential query in a large table with slow hard disk – or this shows other issues. The 3rd largest job was DataToCollection, so either a bulk data request by Selection to Array, Listbox or Modify Selection, needing 25 seconds for 251kb of data. This last one is a hint that the server has an issue, 25 seconds to get 250 kb means either the server has a hard disk issue (check with a tool like HD Tune) or – if it happens only from time to time - a cache issue (use Cache Analyzer). This could also explain the slow query.

Double click a packet to filter the network list to this specific user/process:



Time	Command	Bytes In	Bytes Out	Dura...	User/Process
11-04-15T13:12:42	ExecuteQuery	81	4.933	967,0	D
11-04-15T13:13:01	DataToCollection	184	289.218	19.469,0	D
11-04-15T13:13:12	ExecuteQuery	96	6.200	0,0	D
11-04-15T13:13:12	ExecuteQuery	102	4.234	0,0	D
11-04-15T13:13:12	LoadRecord	54	88	0,0	D
11-04-15T13:13:12	DataToCollection	97	50	15,0	D
11-04-15T13:13:12	ExecuteQuery	122	47	0,0	D
11-04-15T13:13:12	ConnectRecord	41	30	0,0	D
11-04-15T13:13:12	ConnectRecord	41	30	0,0	D
11-04-15T13:13:12	SaveRecord	141	38	0,0	D
11-04-15T13:13:12	SyncThingsToForget	42	14	31,0	D
11-04-15T13:13:12	SaveRecord	599	38	0,0	D
11-04-15T13:13:12	ExecuteQuery	81	5.479	0,0	D
11-04-15T13:13:12	ConnectRecord	41	30	0,0	D
11-04-15T13:13:12	SaveRecord	471	38	0,0	D
11-04-15T13:13:12	SyncThingsToForget	42	14	0,0	D
11-04-15T13:13:12	SyncThingsToForget	42	14	0,0	D
11-04-15T13:13:12	SyncThingsToForget	42	14	0,0	D
11-04-15T13:13:12	ExecuteQuery	147	5.133	47,0	D
11-04-15T13:13:12	SaveRecord	85	38	0,0	D
11-04-15T13:13:12	LoadRecord	54	429	0,0	D
11-04-15T13:13:12	SaveRecord	85	38	0,0	D
11-04-15T13:13:12	ExecuteQuery	135	4.168	0,0	D
11-04-15T13:15:06	ExecuteQuery	81	78	113.491,0	D
11-04-15T13:15:06	ExecuteQuery	81	71	0,0	D

The list is automatically scrolled and the packet is preselected (here in blue). Other packets could be colored in red or yellow, if they are too slow or big.

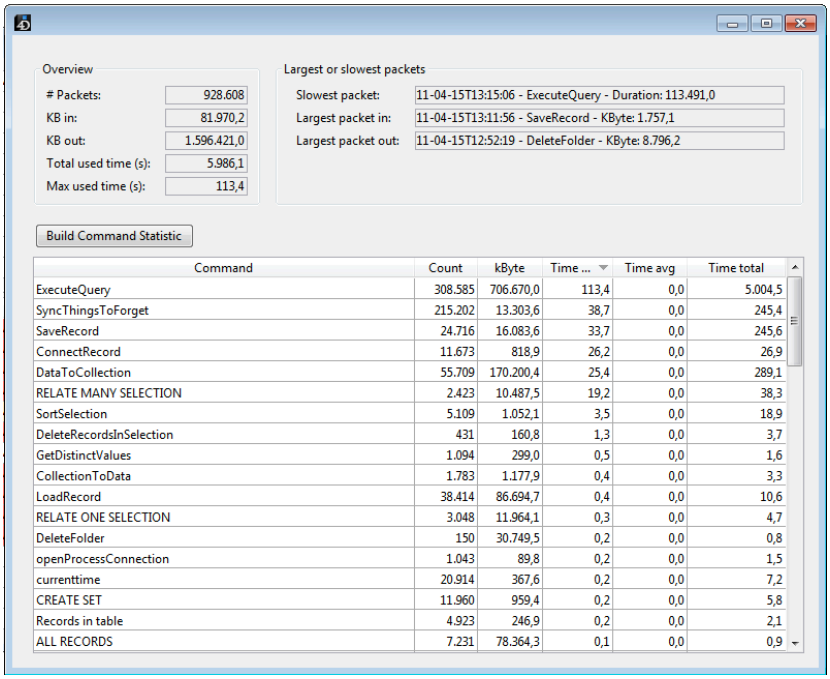
The commands before the slow packet could give an idea (together the with process name) what was done, which part of the code was used. Here we have several commands in a row, all with the time stamp 13:13:12.
Query, Query, then Load Record (which could be also a Next Record or Goto Record command, anything which requests the loading of a record). DataToCollection could be Selection To Array or similar. Then again Query, Save Record. Another Save Record (another table), Query, Save Record. Query, Save Record, Load Record, Save Record, 3 Queries in a row, with the middle one needing 113 seconds.

It is obvious that in a large structure these information do not lead you directly to the method. But in combination with the process name, time, user name (and maybe some information’s received by asking this user) should help to reduce the amount of possible methods, helping to find the code.

We could also try to get more information’s by using other log files in combination. One-way would be to use the Debug Log from the client, we will talk later about this idea. In this specific case the Backup log should also be a great help. See below for tools to read the content of the backup log. If you check the log for 13:13:12, with the specified user name, you will find 5 Save Record entries – and you can see which table was modified and the content of these records. This should make it easier to find the method doing this change.

STATISTIC

Using the Button Statistics” of the main dialog opens this screen:



Command	Count	kByte	Time ...	Time avg	Time total
ExecuteQuery	308.585	706.670,0	113,4	0,0	5.004,5
SyncThingsToForget	215.202	13.303,6	38,7	0,0	245,4
SaveRecord	24.716	16.083,6	33,7	0,0	245,6
ConnectRecord	11.673	818,9	26,2	0,0	26,9
DataToCollection	55.709	170.200,4	25,4	0,0	289,1
RELATE MANY SELECTION	2.423	10.487,5	19,2	0,0	38,3
SortSelection	5.109	1.052,1	3,5	0,0	18,9
DeleteRecordsInSelection	431	160,8	1,3	0,0	3,7
GetDistinctValues	1.094	299,0	0,5	0,0	1,6
CollectionToData	1.783	1.177,9	0,4	0,0	3,3
LoadRecord	38.414	86.694,7	0,4	0,0	10,6
RELATE ONE SELECTION	3.048	11.964,1	0,3	0,0	4,7
DeleteFolder	150	30.749,5	0,2	0,0	0,8
openProcessConnection	1.043	89,8	0,2	0,0	1,5
currenttime	20.914	367,6	0,2	0,0	7,2
CREATE SET	11.960	959,4	0,2	0,0	5,8
Records in table	4.923	246,9	0,2	0,0	2,1
ALL RECORDS	7.231	78.364,3	0,1	0,0	0,9

It displays an overview about the whole network log, as a count of packages, total Kilobytes in and out, total used time and max used time (for a single command).

On the right side it lists the slowest and largest packets.

Click the button “Build Command Statistic” to create a summary of each used command. Note that build this list may need some time for a large log. You may want to sort the list by count (how often a command was processed), by size or by time.

This statistic can help to optimize an application by showing critical parts. By example the application used to create this log file called 20.914 times Current Time(*) on a single day. All Records was used 7.231 times, producing a total load of 78 MB for the network. As this application has the slowest packet with 113 seconds a total time of 5000 seconds for Query, this is obviously the first command to start with.

USING NETWORK LOG ANALYZER - CREATING AND IMPORTING LOGS

Network logs can be created on 4D Server by clicking the Start Request Log button in the administration window, page Maintenance. It can also be started through the language.

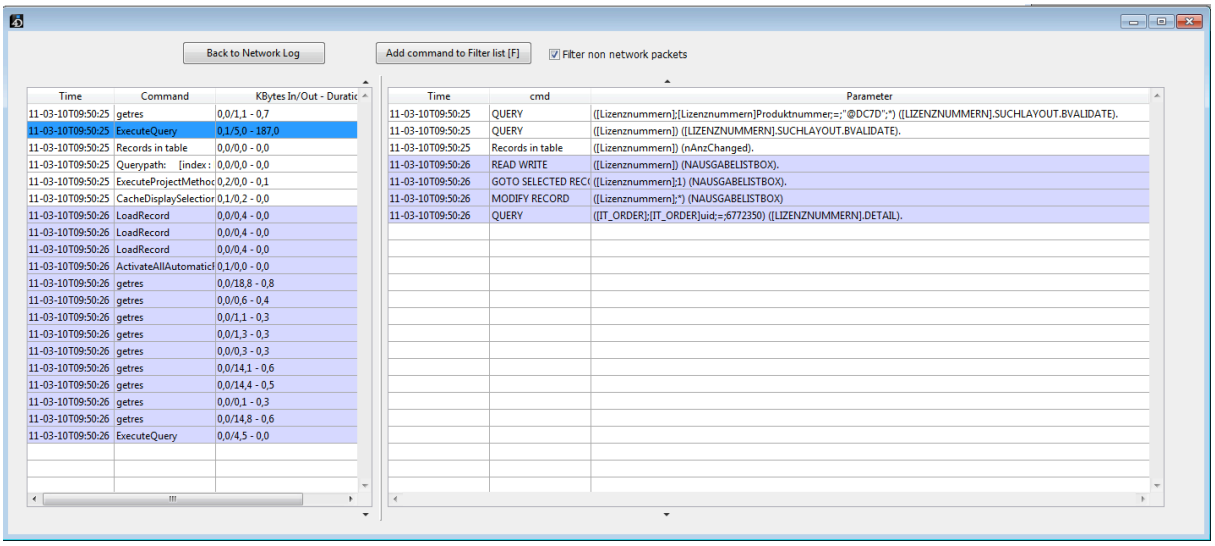
```
SET DATABASE PARAMETER(4D Server Log Recording;1)
```

4D Server will create text files named “4DRequestsLog_X.txt”, where X will start by 1, with a new file created every 10 MByte and a file named “4DRequestsLog_ProcessInfo_1.txt”.

These files can be imported using the menu Report command Import, which may take a while, depending of the size of the log. It is recommended to create a new .4DD data file for every session.

ADVANCED USAGE – COMBINATION WITH 4D DEBUG LOG

A thrilling experience is to combine the network log with the debug log. Imagine we could see every request executed by 4D Server, with size and time required – and beside the 4D commands triggering these requests. And to get a screen like this:



Bad news, there is no hidden command to enable this kind of log. There are some problems with this idea.

First, the debug log is written on the Client, the network log is written on the server.
Second, there is no direct link between a 4D command and a network packet. For some commands are, such as QUERY. But even QUERY is not as simple, as QUERY, QUERY BY FORMULA, QUERY SELECTION, and so on, all use the network command "ExecuteQuery". We know it is a Query, but not exactly what triggered it. To make it more complicated, commands like QUERY are a combination of commands. ExecuteQuery is a combination of UNLOAD RECORD (as the current record is released), QUERY…, LOAD RECORD (as the first record is send to the client). Only one network packet is sent for all this.

Commands such as NEXT RECORD, LAST RECORD, GOTO RECORD and so on, all are linked with the network command LoadRecord. There are many more similar examples.

Many commands, such as Substring, don't send anything over the network. Some, such as Current time, are depending of parameters.

And some commands do not send anything over the network – yet. READ WRITE, READ ONLY or START TRANSACTION does not send anything. What, may you ask. That are network commands. Yes, they don't have an immediate action required. Read Write impacts the next reading operation, as Start Transaction. So 4D Remote automatically caches them and send them together with the next network packet.

Although this makes it more complicated, it is still not impossible. We could create a log on the client and one on the server. We could include some entries (using LOG EVENT) to synchronize both, as both may have different time settings. It will not be exact, as the client log is stored in Milliseconds and the server log in seconds, so may have a small time offset (+- half a second). But still, usually we need this kind of logs for slow operations, which reduces this issue. It is useless to use this technique for a method executing 1000 packets per second. But then there is no need to optimize; it is a different story if a command needs 2 or even 50 seconds.

Still we need to be able to identify a Client debug log, to know to which server process it belongs. We do so by inserting a message with User name and current time in both logs, in the Client debug log and in the server network log. This code needs to be executed in On Startup and everything a new process is started.

The Network Log Analyzer contains some methods (in the folder "DebugLog…") which helps with this task. Either you copy this method in your application or use the Network Log Analyzer as a component.

Run method Debuglog_Sync in On Startup and at the beginning of every new process. If you want to copy the methods into your application, please note that method "DebugLog_LogOnServer" needs to have the "Execute on Server" attribute set.

If you have imported network log AND debug logs from one or several clients, you can select a packet and click the button “Sync with Debug Log” to get a screen as displayed above.

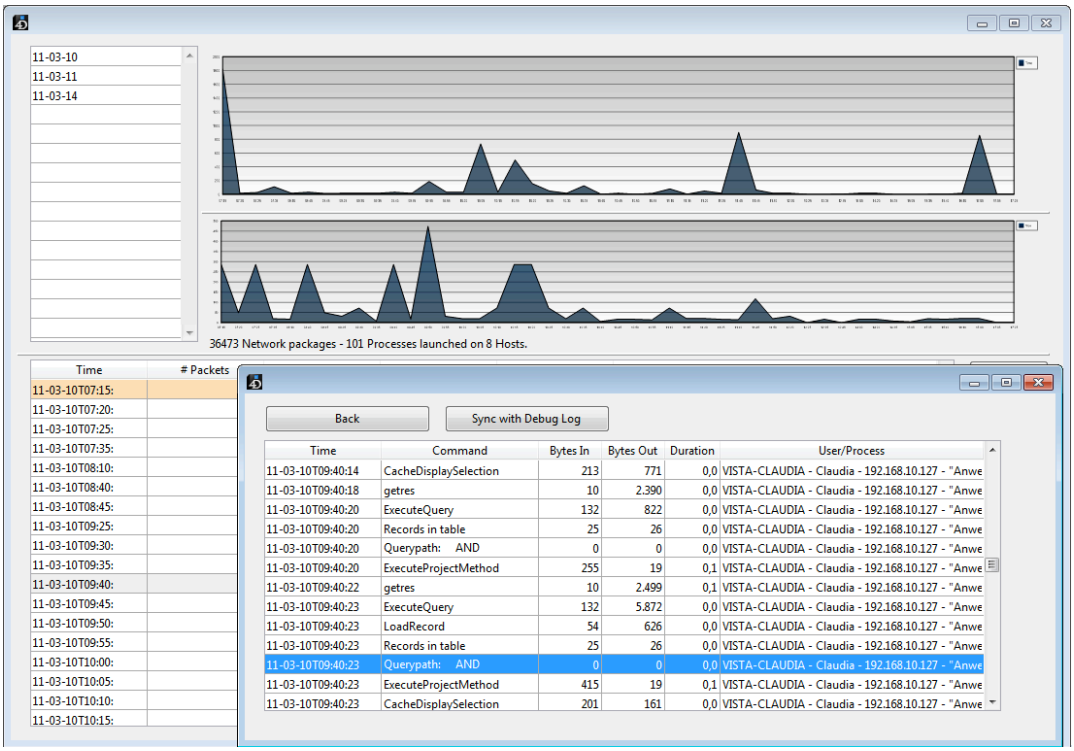
The tool tries to scroll both logs accordingly. It automatically scrolls so the selected packet is visible and displays, for both the network log and the debug log, one second before and one second after this packet (the second before and after the current second is colored in pink). If you need to see older or newer part of the log, click the little arrow buttons above and below the listbox to display more data.

By default all commands that do not produce network packets are filtered (such as Substring, Position, etc). You can display them by unchecking “Filter non Network packets”.

Finally the button “Add command to Filter list” allows you to add a 4D command to the filter list, which may be necessary if a command was forgotten or for new commands, introduced with newer versions of 4D.

ADVANCED USAGE – INCLUDING QUERY PATH

4D’s feature GET LAST QUERY PATH is an important command to analyze unexpected slow queries. As the Network Log Analyzer allows to check server behavior offline, it is an obvious idea to include Query Path in the log.



The log above shows an entry “Querypath:”, a double click displays the full path:

```
Querypath:    AND
    [index : Auftrage.LNr ] LIKE 0   (434 records found in 0 ms)
    [index : Auftrage.Nur Angebot ] LIKE false   (1 record found in 0 ms)
  --> 1 record found in 0 ms
```

4D does not create a Query path automatically, nor is this stored in the log, this is something we need to do our self. Which seems to be a huge amount of work is in most practical cases just some lines of code. Most applications have thousands of Query/Query Selection/Query by Formula commands distributed over their code, which would make it very time consuming to add a Query path handler. As soon we have a combination of debug log and network log, as described above, this need drastically reduce, as the debug log (with option 3) already contains all parameters, such as QUERY([table];[table]field=”hello”). If we build our own query dialog, we still use code to build the query

(using several QUERY/QUERY SELECTION commands), so again, we have the query condition in the debug log without additional work. The problem is only the standard 4D query editor. We can see the query command in the debug log, but don't know which parameter the user entered, so we don't know why the query was slow.

So we only need to identify these queries, which opens the standard Query dialog, which are often centralized in one standard method. We need to change this methods to start the Query Path recorder before and get the result after doing the Query, such as in this example:

```
DebugLog_BeforeQuery (<>CreateDebugLog)
 // true to prepare the log, false to do nothing
QUERY([table])
DebugLog_AfterQuery(<>CreateDebugLog)
```

ADVANCED USAGE – COLLECTING DEBUG LOGS FROM CLIENT COMPUTERS

The so far discussed code allows us to synchronize logs created on server and clients, but a last task is open, how to collect all these logs automatically, to have them stored on a central place?

The component provides a method, to be called in On Startup, to compress existing logs, upload them to a file (or FTP) server and then delete them from the client hard disk, with a single call such as:

```
DebugLog_SaveLogs (logOption;compressOption;uploadOption;uploadPath)
```

Parameter logOption is similar to SET DATABASE PARAMETER(53;x), with x = 0 for no logfile, 1 for normal log file, 2 for logfile with plugins and 3 for logfile with parameters. 3 is the most enhanced option and should be used for this tool, except your application is still using Subtables, in this case you should use option 1 (option 3 is not compatible with subtables and may crash).

Parameter compressOption requires version 12 or newer and uses PHP to zip the logfile, to speedup the log transfer. 0 just moves the log into a subfolder (using a time stamp as name), without compressing. 1 compresses the subfolder (using PHP) and 2 compress the subfolder and delete the uncompressed files.

Parameter uploadOption allows an automatic upload, either to a file server (needs to be mounted or allow guest access) or using FTP (requires 4D Internet Commands to be installed). 0 = no upload, 1 upload with ftp and 2 upload using Copy Document to a file server.

Parameter uploadPath specifies the target for the upload, if uploadOption is 1 or 2. For file server upload use a valid path name, on Windows such as \\\\server\folder\folder, for FTP such as ftp://user:password@ftp.example.com/path/folder/.

Example to compress and upload to file server:

```
DebugLog_SaveLogs (3;2;2; "\\\\NAS\\Archive\\Logs")
```

NETWORK PACKET NAMES

The Network Log Analyzer automatically displays human readable names for the packet ID's. As not every network packet has a one to one relation with 4D commands, some name are identical (such as ALL RECORDS), such as close (such as SortSelection, which could be an ORDER BY or ORDER BY FORMULA), some allow to guess the reason (such as RebuildIndexByRef, CreateFullTextIndexOnOneField or CancelShutdown) and some appears weird (such as DeleteFolder or writers).

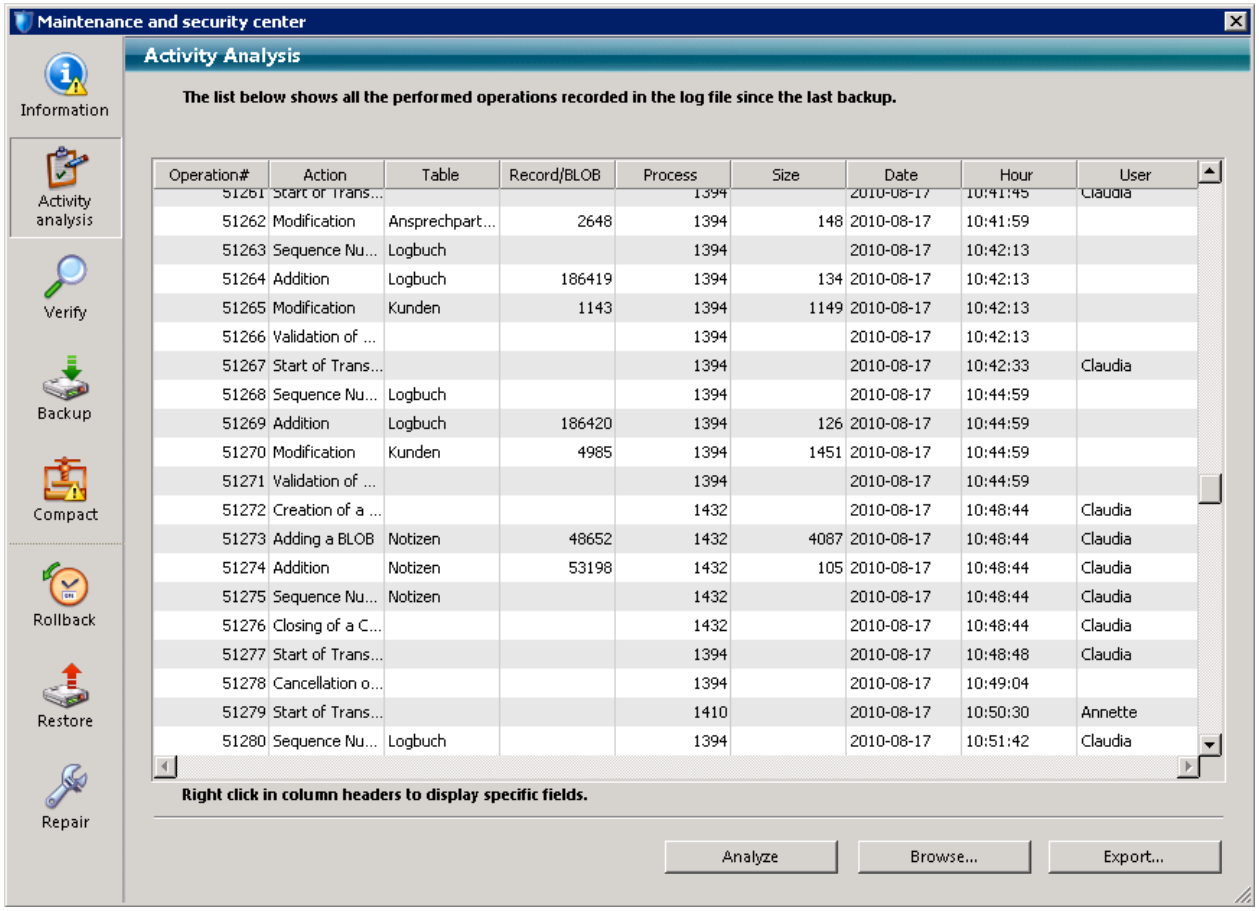
The analyzer contains a table [Log_Commands] which assign human readable names to each ID. The names are used from 4D's development team internally, in some cases, where they are one to one related, such as ALL RECORDS or CREATE SET, we have replaced them with the 4D name. The list of replacements is in table [Command_Network]. For new created data files both tables are automatically filled with data based on the file \Resources\Cmd_network.txt and \Resources\Cmd_table.txt

You may see during the launch phase of a client many packets with "getres" and "DeleteFolder", sometimes quite large and/or time consuming. This is related to the transfer of the Structure, Plugins, Resources and Extras folder to

the client. As they are cached, the transfer should only happen for the first launch. A bad design (such as a folder with several very large files and some small ones, which are modified by server code, and so forcing an update to the client) could lead to performance loss as such folder always needs to be recompressed by the server and retransferred to every client. Optimize your folder organization, by avoiding top level folders with thousands of small files (such as html files), move them a folder level below and similar avoid to modify regulary the file content of a file in a subfolder, as these triggers the recompress and transfer of the whole subfolder.

BACKUP LOG – LOG TOOLS

4D's backup log can also be used for debugging/understanding what was going on, even it contains only Write/Delete operations (no Reads/Query).

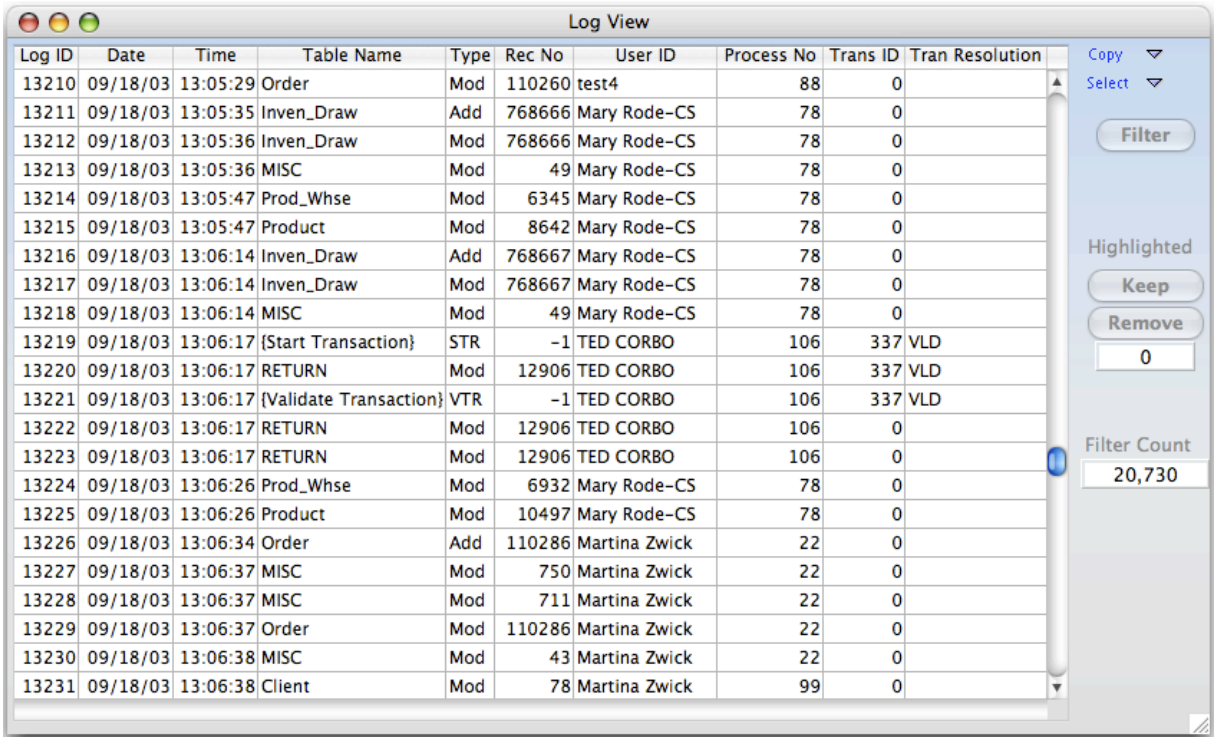


Data can be exported and then opened in MS Excel for checking.

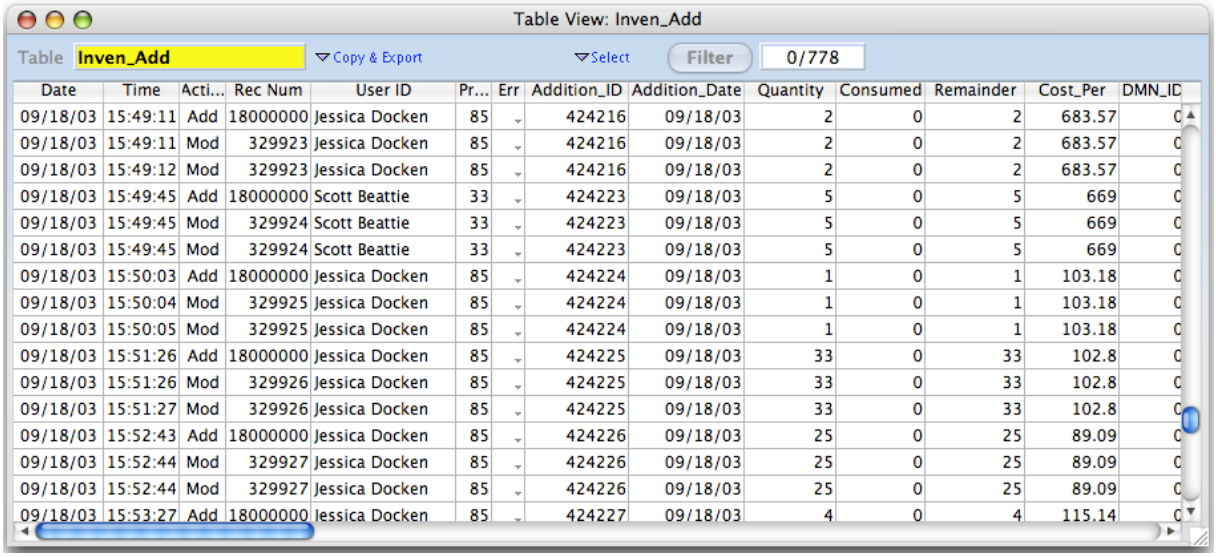
In a heavy used app with 100 users the log file can be very long. Issue could be spread over several log files. Logs could be old, so not so easy to open.

LogTools from Business Brothers Inc (http://bbsp.com) allows to open v2004-v12 backup logs – even without existing/available full backup. It allows to go back in history.

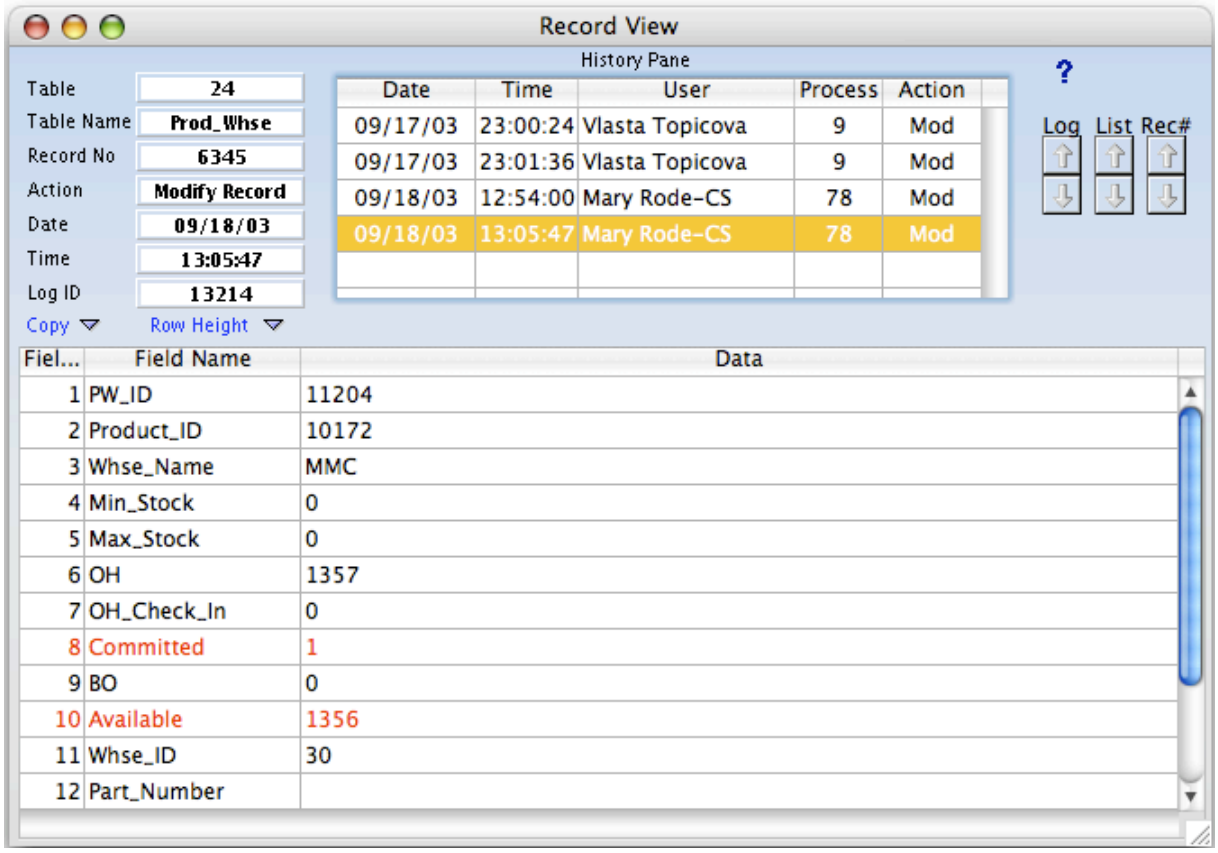
The main dialog shows all transactions, not much different from MSC:



But you can filter all operations based on table, time frame or user, such as:



Sometimes really useful: you can see the history of a record:



This allows to find strange problems, as “application looses data, records suddenly change the value, etc”.

We had a case “4D looses records”. Customer said that sometimes all records of a table are suddenly lost, 4D has a major problem, he needs a fix asap.

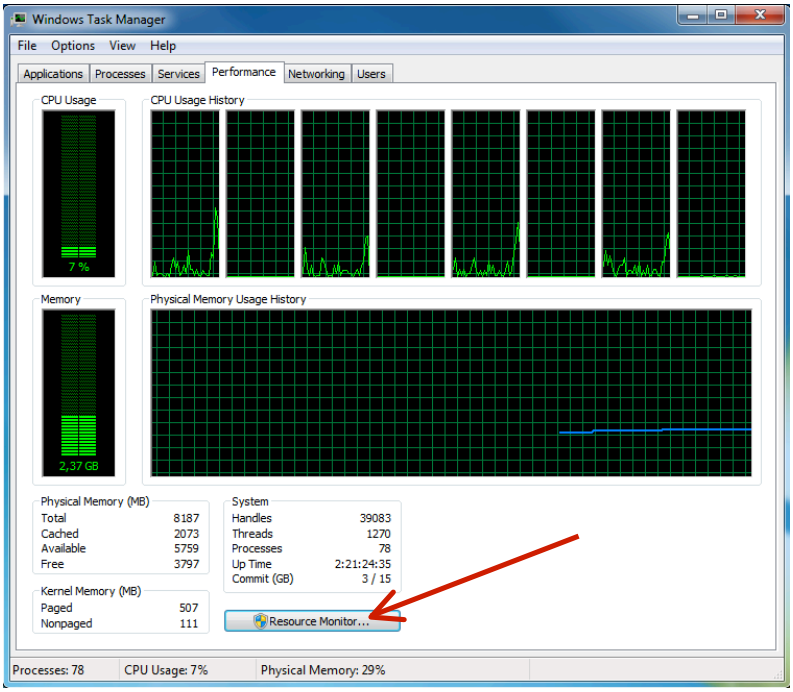
Checking the logfiles of the last week with LogTool we found for this table a TRUNCATE TABLE entry. Customer answered “I never use this command”. 4D uses it internally, if possible (because faster), if you do a Delete Selection on the whole table. I never do this…

Checking the operations from the same user directly before the TRUNCATE TABLE entry gave a glue what the user was doing. Finally it was a Search dialog saying “which project should be deleted”, which was accepted without any data entry from the user. The developer added a @ Character without checking, run the Query (which returned all records) and used DELETE SELECTION.

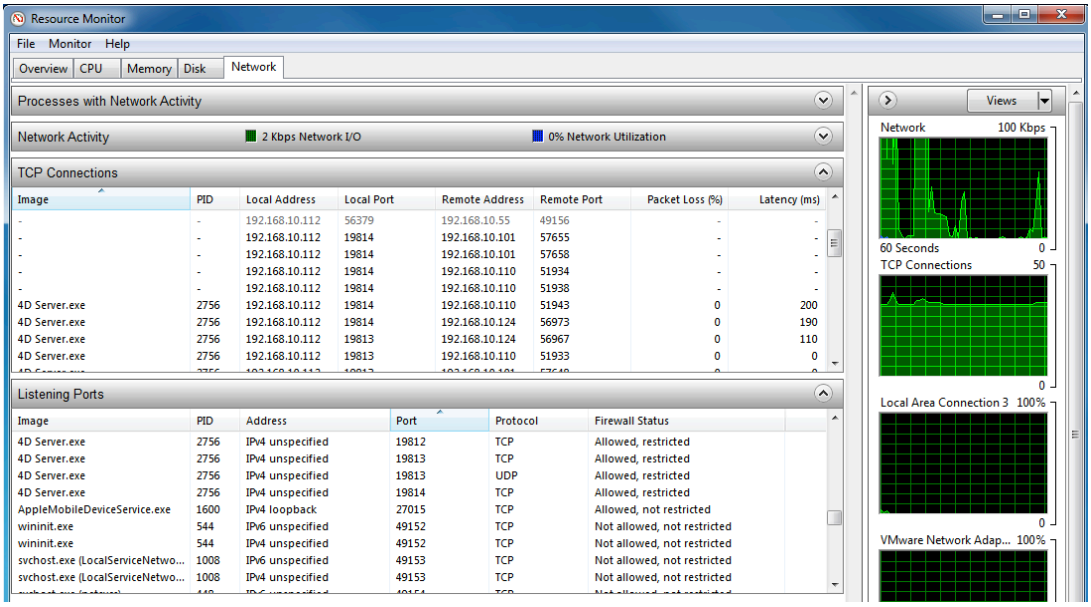
What started to look as a very difficult case was solved in less than an hour.

WINDOWS RESOURCE MONITOR

Everybody knows (and uses?) the Windows Task Manager, to learn about CPU load – and monitors memory and virtual memory load of 4D and other apps. Windows 7 and Windows Server 2008 Enterprise Edition enhances the Task Manager with a Resource Monitor:

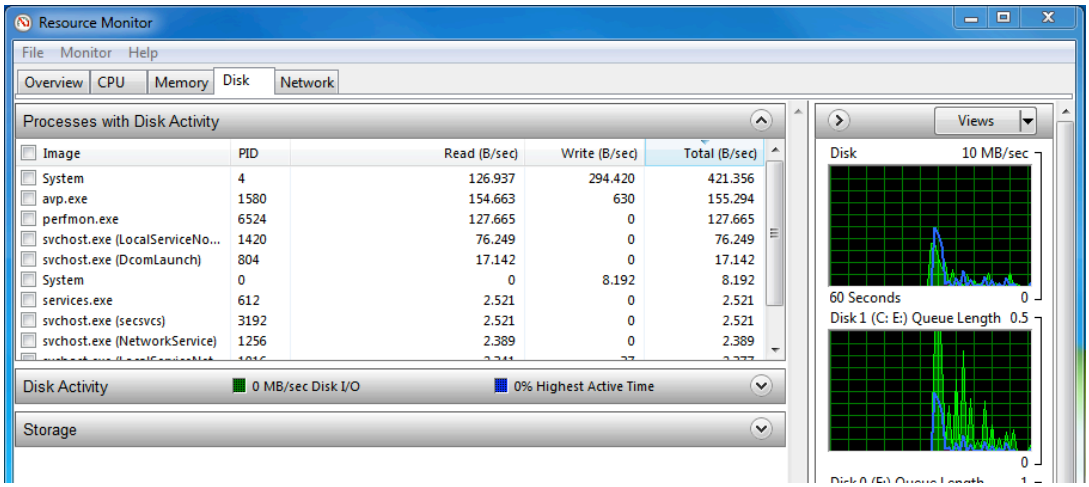


The Resource Monitor supports more detailed information about CPU, memory, disk and network, such as:



This tool allows to see which ports are used, as example by 4D Server and the settings of the Windows firewall for these ports. It also shows all open connections, like 4D Server to remote 4D, with status information like packet loss or latency time.

The Disk page does not only shows the load of the disk, but also the queue length, grouped by physical hard disk, allowing to detect bottlenecks:



Especially interesting on the disk page is “Highest Active Time” and “Queue Length”, which is displayed per physical disk. If the activity is about 10% or the queue length is less than 0.2, all is fine. If it goes to 0.5 for some seconds or even above 1, the server has problems. The disk has more jobs to do than it is able to perform, so new jobs goes into a waiting queue. If 4D Server needs to read data, say for a sequential search or reading a record while the flush manager is storing the cache to disk, the read needs to wait, slowing down everything.

The old solution for this problem was to purchase a RAID controller with onboard cache memory. These days we suggest to switch to SSD technology, which greatly enhance the amount of I/O per second (IOPS). While normal SATA disks handle about 100 IOPS, expensive (15rpm) SAS about 200 IPS, old SSD about 5,000 -10,000 we see with current (as of 2011) SSD around 50,000 IPS, while special server SSD’s go > 500.000 and some are announced with more than 1 Million operations per second. Compare that with 200 for a SAS drive.

This dialog helps you to identify if the disk is a problem. When we switched from SATA to SSD in 4D Germany we saw a change in Queue Length from 0.5 to 0.01.

The network page may help to identify router/switch issues. It lists every open 4D Server connection – to every Client and every process. Watch a server with 1 Client connected to see normal behavior. Start a 2nd process on the client, to see another line for a 19814 port appearing in the list. Wait a while (without doing anything on the client) to see it disappear (4D set it to sleep, depending of timeout). Click something on the client –the process appears again. If there are network issues at the customer site, it may help to compare this screen with the monitor window of the firewall/gateway/switch.

This screen also helps to demonstrate the “normal” behavior to the customer administrator, helping him to understand how 4D set’s processes to idle and that they are fully closed from 4D Server (as they don’t appear anymore in Microsoft Windows Resource monitor).

On Mac you can see this behavior using Activity Montior, select 4D Server, then Information, Open files and ports. Scroll down, you get a list for every client and every port.

It can be also checked by Command Console (DOS) on Windows and Terminal on Mac, see tech tip: http://kb.4d.com/search/assetid=76029