FullRecord

FullRecord 3.10

A product from Laro Group SRL

by Jorge Alejandro Lavera

FullRecord Reference

Printed: mayo 2014 in Argentina.

Publisher

Jorge Alejandro Lavera

Special thanks to:

To my beloved family: my wife Marina, Vicky, July, and Chris.

Table of Contents

Foreword

This manual is intended to explain to you all the options you have to use FullRecord. Many people never read a manual. You may use the product without reading a word of this manual... but then, you are not going to get all the value you deserve for your purchase. There are always options that are complex enough to read this thing.

Part 1

1 Overview

FullRecord allows you to automatically record all the changes performed on your databases (adds, deletes, changes) and based on that recording, with an inspector program, you can later recover erased records, inspect the modified information and chronologically analyze the operation of your system.
See the "Features" chapter for an comprehensive list of characteristics.

Part 2

2 Features

FullRecord stores the database operations of your system in one audit file:

- It can records the operations from any kind of file (ISAM, SQL (*), even IMDD if needed).
- It can also records any manual hallmark you want to add (like "Special button pressed", or "Entered this particular procedure", even local variables or special events)
- It is compatible with the Clarion IPDriver.
- The auditing file may be ISAM or SQL (SQL currently tested with Pervasive SQL, SQL Anywhere, MS-SQL Server, ODBC-MySQL and ODBC-FIREBIRD). SQL storing supports server-side autonumbering.
- There is only ONE main file for auditing storage.
- Option to compress the audit data, to save disk space used by the Audit file.
- Minimal local code is generated in your procedures. This makes your audit system COMPACT!
- No decisions are made at the moment of recording, and only one record is stored per operation in FullRecord mode (no matter how many fields changed). The only exception is if you change the primary key, then two records are saved (This is to enable FullRecord to locate the original information). This makes your audit system FAST!
- As an alternative from FullRecord 3.0 on, you can store the changed fields only.
- Examine the audited data in functional ways:
  "Field" inspect window (to see the history of a particular field),
  "Record" inspect window (to see the history of a particular record) and
  General Audit Browse.
- You can undelete a deleted record or recover a changed record.
- When you recover a deleted record that was replaced with new data, you may choose to overwrite the new data or renumber the old record to add it to the end of the file.
- The FullRecord Global Template has "basic" and "advanced" options. In the Basic mode some of the complexity is hidden from the developer and it's ok if you set with the defaults. When you select the advanced option you have the ability to customize FullRecord to the maximum.

FullRecord supports all the regular field types, as well as the auditing and recovering of files with:

- Memo fields (unlimited quantity)
- Dimensioned fields (up to 4 dimensions).
- OVERed fields
- Fields inside GROUPs
- Dimensioned GROUP fields
- BLOB fields (See BLOB fields)

(*) SQL back-end operations like cascade-engine updates and stored procedures cannot be audited, as they are out of the scope of the Clarion program.

Part 3

3 History

version 3.10 (21/05/2014)
Fix: Support for Clarion 9.1

version 3.09 (25/02/2014)
Fix: Several minor fixes.

version 3.08 (17/01/2013)
Fix: The "run-time disable" feature were opening the file to read it in non-sharing mode.

version 3.07 (17/01/2013)
Fix: Chance to use the audit file without any Blob field (if used in "fullrecord" mode only). Previously it was requiring a blob field even if not in use.

version 3.06 (16/08/2012)
Fix: The "run-time disable" feature wasn't working for the "mixed" or "field" audit mode.

version 3.05 (1/12/2011)
Fix: Some libraries used by the template were not picked up automatically on Clarion 6.x.

version 3.04 (15/11/2011)
Change: Temporary files no longer created on program's folder. Now taking Windows's temporal folder.

version 3.03 (29/07/2011)
Change: All examples revised.
Fix: Automatic assignment for date, time and user, for added or changed record, were not working.
Fix: "View Record" in the inspect window was not showing the memo content if the memos were not the first fields in the file.
New: Guide to examples.

version 3.02 (12/05/2011)
New: Support for Clarion 8.
Fix: Auto numbering code were being generated to fields not existing on DCT.
Fix: Sample DCT was missing a variable.

version 3.01 (19/04/2011)
New: Run-time selective tables global now made optional
New: Documentation improved updated
Fix: Some duplicates names removed from Legacy template

version 3.00 (17/02/2011)
New: Support to store changed fields only.

version 2.25 (15/7/2010)
Fix: Thread local attribute removed (Detected by Clarion 7).
Fix: IMDD examples had wrong dct.

version 2.22 (3/7/2009)
New: The install process supports Clarion 7's folders structure.
New: All examples were migrate to Clarion 7; you'll have native C7 apps when working

with C7.
Fix: In the ABC chain, the BrowseAuditFiled prototype was changed to compile with Clarion 7.

version 2.21 (22/1/2009)
New: PostgreSQL examples
Fix: Spurious Read operations were saved if you were using Clarion 6.1 or before.

version 2.20 (22/11/2008)
New: Local extension, now you can locally select which files to call for Record Audit Inspect (BrowseAuditRecord).
New: Several new embed points.
New: SaveAudit: Now it can add records in the audit and historical audit files when the datetime field type is used. Set the Date and Time audit with the same field in global extension.
New: New examples using SQL Anywhere.
New: New examples for MSSQL and Firebird showing the use of Blob fields and without Blob as well.
New: Many examples were updated; also a new guide table for all the examples (See Installation)
Fix: CleanAudit now cut the Audit file correctly when the date field are datetime type (SQL).
Fix: CleanAudit was not working right with TPS AuditNew file.
Fix: CleanAudit, Legacy: Now it can add records to historical Audit even if starts from an empty Audit file.
Fix: CleanAudit: was not working right with MSSQL driver and Clarion 6.3 9057/9058. The file AuditNew is no longer necessary for SQL.
Fix: InspectField was not working right with no audit fields present.
Fix: If some operations in a file were not audited (by locally run-time disabling the auditing), and the user browses the field inspect from a Form so the last audited operation doesn't match the current record, the current value of the inspected field in the form were changed to the last audited value.

version 2.11 (31/8/2008):
Fix: Auto detection when there are no memos in the audit file (It was giving compiling errors, as it was expecting at least one memo).
Fix: Blob handling improved, there was a couple of compilation and run-time errors if there are Blobs and no memos.
Fix: GPF Error handling blobs when using the "View" button.
Fix: TXD Analyzer had a problem recognizing the proper size of fields longer than 32 Kb. You have to pick your TXD again with this new version if you already did it with a previous version of the Analyzer.
Workaround: A bug in Clarion's IDE was causing the global properties window to scroll down each time we change from our template to another. We changed the multi-row tabs by scrolling tabs to avoid it.
Fix: many help texts improved by Nardus Swanevelder (Thank you!).

version 2.10 (19/6/2008):
New/Fix: New feature Browse Audit "Advanced" now working properly (browse the fields changed without pressing the "view" button).
Fix: Some bogus "read" operations were improperly saved on change right after add.

version 2.09 (30/3/2008):
New: Option to workaround Btrieve memo property assignation bug - Using Btrieve driver memo fields sometimes adds "garbage" to the memo field (Acknowledged by SV as #30485).
Fix: ABC Template: "Audit" file name no longer hard coded inside a global procedure.
Fix: Supress "previous" error message when inspecting first "change" record without previous read.
Fix: Some "read" operations, required for changes, were not saved on a fresh audit file.

version 2.08 (24/2/2008):
New: Procedure Name now can be recorded in "source" type procedures.
Fix: ABC Template: was not compatible with Clarion 6.1 and before.
Fix: Procedure Name generation moved to first priority on Init code section.
Fix: Read buffer was incorrectly saved when a change operation was taken place right after an insert.

version 2.07 (27/1/2008):
New: TXDAnalyzer now can import file selection from FullRecord.Fil file.
New: Legacy template: option to keep Audit file opened to improve performance.
Improvement: Record buffers optimized on GenerateSearchKey procedure.
Fix: Inspect field windows was messing with file buffer when called from Form windows.
Fix: Installer were wrongly linking the shortcuts.
Fix: Threading problem with inspect field list (Field list were mixing if opening from concurrent inspect windows).
Fix: ABC template: Inspect window was improperly closing related files.

version 2.06 (12/1/2008):
Improvement: Increased speed of code generation.

version 2.05 (18/11/2007):
Fix: Memo number was incorrectly generated upon call Field Browse inspect.
New: Auto-check for previous record in audit trail, if previous buffer does not exist (because for example is a new Audit file) a "read" operation is added prior to the "change" one.

version 2.04 (13/11/2007):
Enhancement: Template Run-time disable tweaked to improve the performance.
Fix: Some portions of code were still being generated when the template were totally disabled.
New: Example process procedure in ABC and Legacy examples for measuring performance, with and without template activation.

version 2.03 (10/11/2007):
Fix: Alias files were not supported

version 2.02 (03/11/2007):
Fix: Installer were not retrieving serial and maintenance key for next revision.
Fix: Legacy templates: BrowseAuditRecord was not ranging using the proper record.
Fix: Legacy templates: Inspect Procedure and Recover procedures were not properly opening/closing files, leading to Error 37 or Logout errors depending on how they were called.
Fix: Files with more than one blob were auditing only first blob.
Change: "Read" record is now saved when the key information is changed, the "read"

record is still necessary to determine how was the key before the change.
Change: "View" button on "Inspect" window modified to look for the "read" record if it exists, or the previous operation if there is no "read".

version 2.01 (29/10/2007):
Fix: Legacy templates: were not generating BrowseAuditField calls.

version 2.00 (28/10/2007):
New: From version 2.x on FullRecord is compatible con Clarion 6 and later (Clarion 5.x no longer supported).
New: Now only one record is stored per operation, always. One per insert, one per change and one per delete ("read" type records are not longer necessary and were deprecated). (See revisions 2.02 and 2.05).
New: Every insert, update or delete operation is automatically audited, even if it is in source code, so you can rely that any deep hidden or lost operation will be saved to the Audit file. You don't need to add any code by yourself! And for both ABC and Legacy.

version 1.84 (14/8/2007):
New: Global checkbox for disabling Audit Record and Audit field features.
Fix: Cleanup procedure; properly find true last ID number using ID key instead of Date key.
Fix: Proper defaults for "Message" global tab on new app.
Fix: Handling error on third and fourth parameter in arrays, produced a GPF in Inspect procedure if third of fourth array index had less range than first or second index (i.e. 2,2,4,4 produced no errors, 4,4,2,2 produced GPF).
Fix: Call code for Browse Record and Browse Field were generated even if the file were not in the Audit file global list.

version 1.82 (15/4/2007):
New: Record Inspect feature (Inspects changes for a particular record from a button, or Hot key, on a Browse or Form).
New: Local disable of code generation for calling Audit Inspect Browse.
New: New "version" field on Audit structure, on preparation for incoming feature.
Improvement: "Field" inspection window now uses the field picture to show the data.
Change: Global "Variables" moved from "General" Tab to a Tab on their own (Advanced interface)
Fix: On Update procedure for upgrading record structure inside Audit file, proper support for audit record stored in blob field, with or without compression.
Note: On multi-dll systems a full recompilation is needed.

version 1.81 (11/3/2007):
New: Advanced/Basic Settings interface, to hide the clutter out of the template options.
Improvement: Some template options rephrased for better understanding.
Fix: Multi-dll example for C6 - parameter list for BrowseAuditField procedure was wrongly taken from previous beta.

version 1.80 (05/3/2007):
New: Field Inspect feature.
New: Multi-dll Clarion 6 ABC example.
Change: Replaced all LOC: prefixed variables by JAL: prefix to avoid possible conflicts with user variables.

version 1.72 (12/1/2007) (limited release):
Fix: A field name from the Messages file were still hard coded in the template.

version 1.71b (06/12/2006):
Fix: Undefined symbol found when you delete and repopulate the global extension in a dll of a multi-dll system.

version 1.71 (02/12/2006):
New: In the MSSQL C6 example, a new inspect browse showing how to use a SQL query to find the data on the Audit file.
New: German TRN file.
New: MSSQL Legacy example for C6.
Improvement: "Messages" file name and fields are no longer hard coded.
Change: Audit and storage routines related with user messages were moved from the FullRecord Template to the NeatMessage template.
Change: Remove of external name in "group" fields of the MS-SQL example to prevent warnings from FM3.
Change: Rename of external name in keys of the audit files of the MS-SQL example to prevent warnings from FM3.
Change: in the audit.txd file, changed the name of the "GlobalData" area to "FRGlobalData" to avoid common conflicts when importing.

version 1.70 (16/10/2006):
New: FAQ section and several enhancements in the documentation.
New: Full Blob fields support for both Audit and Recover now for C55 as well as for C6.
New: Now you can see the stored Blob size in the Inspect Window.
New: Global option to allow to skip specific fields from the inspect window (two methods - either by the "NoPopulate" option or by adding a "FRSkip=True" option in the DCT).
New: "Recover" control template now generates global procedures, being able to support up to any number of files (tested with 960+ files).
New: Global option to "Suspend" the template action, so the template is disabled but you still can compile your program without errors (even if used manual calls).
New: "Cleanup" process now available for MSSQL files.
New: "Cleanup" process now can handle completely erasing of the Audit file more than once.
New: Now you can optionally frame the "Cleanup" process within a transaction.
New: Portuguese translation file.
New: Install generation of .ini file compatible with Clarion Desktop (http://www.clariondesktop.com/)
Fix: Were some problems with server-side autonumbering of the Audit file.
Change: MSSQL example now shows how to set server-side autonumbering for the Audit file.
Fix: Error found in processing file with Blobs.
Fix: In the Multi-dll example, the "process" procedure was declared in the wrong module (from the caller exe).
Fix: Installation program, now you can set the install path (missed feature).

version 1.62 (23/8/2006):
Fix: TXD Analyzer 1.02: former version file name limit of 30, proven to be too short. Expanded to 40 characters.
Fix: TXD Analyzer 1.01: former version was not taking ALIAS files into account.

version 1.61 (11/8/2006):

Improvement: The message showing the "memo" information now will have the field name as Title.
Improvement: From this version on, your installer settings will be memorized in the registry (i.e. no longer look for your original e-mail to get the serial and code again!).
Improvement: In the "recover" control template, an automatic window refresh was added after the recovering (to immediately see the recovered entry log in the browse).

version 1.60 (7/8/2006):
New: TXD Analyzer, a new companion program that will let you analyze your dictionary and easily pick the correct values for FullRecord's variables. It will also generate a "file list" for you to pick it up from your template global settings. Source code included.
New: Now pressing a "View memo" button in the "Inspect Window" you can see the full contents of the memo fields.
New: Now global option to select if group fields are skipped from Inspect window.
New: Embed for manually handling events inside the inspection window.
New: Example auditing IMDD 2.2 files (IMDD driver required).
Improvement/Fix: The name of the procedures is now handled in a global threaded queue, to avoid loosing the name when a lookup is performed. This way, the recorded name of the procedure is completely accurate. You have to change the DCT and global properties if you are migrating from a previous version of FullRecord (See the Installation over previous versions chapter).
New: Global option to disable or enable the new "View memo" button inside the "Inspect Window".
Change: Several changes in the documentation to reflect the new features. Pay special attention to the changes in the General Tab, Inspect Tab, Files to Audit Tab, Translation options and the inspection window chapters, and the whole new Using TXD Analyzer chapter.
Change: Window structure changed to hold a new "View memo" button to "view" the full contents of the memo fields.
Change: Translation files changed to add a new equate text for the "View memo" button .
Change: The local procedures generated associated with the "View record" control, are now deprecated. The inspect procedures are now only global.
Note: A full recompilation of your system is needed if you upgrade from previous versions.

version 1.55 (12/7/2006):
New: Extended support for saving assorted information to the Audit file (See the new chapter Manual save of custom information in the documentation).
New: Global switch to show only changed data in the Inspect window.
New: Embeds points before and after queue generation for the Inspect Window.
New: Group fields now skipped from Inspect window.

version 1.54 (05/7/2006):
Fix: When you have a GROUP with some field dimensioned and others not dimensioned, all were treated as dimensioned.

version 1.53 (02/6/2006):
New: Local override to avoid to track unwanted procedure names (like for example Report Previewer).
New: Global setting to avoid the declaration of the zlib library (for compatibility with other templates that also uses it).

Fix: Some local code from extension templates were generated anyway when the template was globally disabled.

version 1.52 (23/5/2006):
Change: Autodetect NeatMessage and avoid the procedure name to take the name of the "message" window; thus the recorded message now shows the name of the calling procedure.
Fix: When storing record in Blob field was not clipping the data, therefore occupying more space than necessary on the file.

version 1.51 (18/5/2006):
New: New example, for MS-SQL Server (Clarion 6.x).
Fix: Some operations where not compressed/uncompressed when the Stored Record were stored in a Blob type field (Clarion 6.x only).

version 1.50 (15/5/2006):
New: Compression scheme for all read/write operations on the Audit file.
New: Compression flag in the Audit file, allows the storing of mixed compressed and uncompressed records.
New: New layout for the Audit file, now stores the record in a memo field and add information about the workstation and calling procedure.
New: New global variables and new audit variables referenced in the template global extension.
Note: If you want to use the new features, changes need to be made for the audit files. You will have to convert your existing audit files to the new format.
Improvement: Examples and documentation updated to include new fields and compression support.
Improvement: New default names for all fields in the Audit files (Set in DCT and TXD files).
Improvement: New default values for operation codes (Set in TRN file).
Note: If you want to use the new operation codes, you will have to convert your existing codes in your audit files to the new values.
Improvement: "Search key" now named as a more descriptive way of "search key"; changed in documentation, dictionary and all the examples.
Improvement: New default names for all global settings to allow a quick-start implementation.
Change: Cosmetic; populated "view" and "recover" buttons no longer Flat.
Fix: "Messages" file name were hard coded as "Mensajes" in one place in the Legacy FullRecord template.
Fix: Recover control template were referring to global audit file, preventing to recover a record from a subsidiary audit file (local declared).

version 1.33 (28/4/2006):
Improvement: Now you can choose how many global inspect procedures will be generated by module, making the compilation way faster! Prior to this change was one procedure per module, now the default is 20.
Improvement: Now you can set up globally the name of the external translation file, thus your file is safe for future upgrades. Also, because of this you no longer need to externally rename the appropriate file to the default filename.
Improvement: Now you can set up globally the name of an external file to hold the "view" windows definitions, therefore your custom windows will be safe against future upgrades.

Change: The two new entries in the TRN file (From version 1.31) are now also in the Spanish and in the French translation files.
Fix: Dimensioned GROUPs were generating an error when using global inspect procedures instead of local (incomplete implementation of improvement from version 1.32).

version 1.32 (23/4/2006):
Improvement: Dimensioned GROUPs were not considered; now fields inside dimensioned groups inherits the dimensions of the group and the group itself is skipped from processing.
Change: When storing the record in a memo field, a memo-to-record comparison in clarion 6.2 produced a GPF. It was changed to avoid the error and the new method is compatible with any version.
New: Search engine in the on-line reference.

version 1.31 (09/4/2006):
New: Better handling of undelete operations. Now if you try to recover a record which already exists, you may choose to renumber it to add the recovered record at the end of the file, instead of deleting the current record. See the updated Recover chapter.
Change: there are two new entries in the TRN file.

version 1.30 (22/3/2006):
New: Save recover information (what, who and when the record was undeleted)
Improvement: Better handling of Audit file open errors.
Improvement: The name of the global DCT variables is no longer hard coded.

version 1.25 (10/3/2006):
New: (Clarion 6.x only) Support for storing the audited record in a BLOB field; that's make possible to use Firebird as audit file database.
New: (Clarion 6.x only) Example app using ODBC and Firebird as storing audit file.
Fix: For the "inspect" button, now you can see the content of a "message" record (UM) when browsing an audit file different from the main one.

version 1.21 (19/1/2006):
New: Extension template to automatically manage record updates in a process (See Changing record layout).
New: Example .app and .dct to show how to implement the record update in the audit file.
New: TRN French file (Merci to Jean-Pierre GUTSATZ)
Fix: Audit message wasn't properly generating Search key.

version 1.20 (15/1/2006):
New: Support for recovering records using direct SQL syntax for ODBC (mySQL) and Pervasive SQL.
New: Support for FinalStep 2.x template in the Inspect window, so it will have the same dynamic wallpaper than the rest of your app.
New: New chapter in the help file, about changing the record layout of your files.
New: Debug option for assisting remote support.
Imp: Recover control template - now no longer requiring that the primary key were the first key.
Imp: Extended information on record recover error (native errors for SQL drivers).
Imp: For global procedures, now will generate them with full long names (instead of short 8.3 ones). Delete the old .clw and .obj files prior to recompile to clean up the garbage.

Chg: Embed name after open internal inspect window changed (See Installation over previous versions).
Chg: Inspect window - Wallpaper was mandatory, now its optional.
Fix: Global tabs were not showing when trying to use the control templates in the exe of a multi-dll system.
Fix: Key-in input picture @k is now supported for Search-key generation and inspect window display.

version 1.19 (14/12/2005):
New: A multi-dll example app.
Imp: Inspect window - Default inspect window wallpaper changed to blank instead of 'Fondo.gif'.
Fix: Inspect window - should not process message file when its name is not specified in the global extension of the template.
Fix: Inspect window - "Memo" fields memory buffer was assigned like file variable with a fixed name. It should refer to global variables to allow for not-uniform databases field names.
Fix: Internal automatic control to handle the use of a "Memo" field to store the record in the audit file.
Fix: Inspect window - Global inspect control template generating compile error in the export list when used in a main data dll in a multi-dll system.
Fix: Inspect window - Module and pragma definitions were incorrectly generated in non-data dll in multi-dll system.
Fix: Inspect window - Global queue not generating export list when used in a main data dll in a multi-dll system.
Fix: Inspect window - Not referenced global queue to data dll when used in a main data dll in a multi-dll system.
Fix: Inspector window - global option - not properly showing or comparing memo fields with previous value.
Fix: Global extension - Auditing Tab should be active for multi-dll apps to specify files to inspect.

version 1.18 (12/12/2005):
Imp: User variable in the audit file no longer hard-coded (Was USOPER). See Installation over previous versions.
Imp: Record variable hard-coded in a couple of messages (Was DERECO).
Imp: Memo variable in the audit file no longer hard-coded (Was DEMEMO).

version 1.17 (12/12/2005):
Fix: some "Operation" codes were left hard coded.
Fix: Inspector window - Column options not working (compiler error) when generation set to local instead of global.
Fix: Inspector window - local option - not properly showing or comparing memo fields with previous value.

version 1.16 (12/12/2005):
Imp: FullRecord.TRN - "Operation" equates were added to translation file; they are no longer hard-coded.
Fix: Rsa55 Example - Windows background changed to supplied .GIF
Imp: Rsa55 Example - dct path removed from example app
Imp: Fra55 Example - dct path removed from example app
Imp: Fra55 Example - Example message keyword filled in global extension
Chg: FullRecord.DCT - field comments translated from spanish to english

Imp: FullRecord.DCT - Reference from unknown SNC file changed to AUD file.

version 1.15 (5/12/2005):
New: Local alternative to global generation for "Inspect" control template (If global option is not active, then local generation will take place, instead of issuing an error).
Fix: In global "inspect" control template: Support for inspecting other audit files than the global declared one (for example the historic audit file).

version 1.10 (5/11/2005):
New: "Inspect" control template completely recoded to avoid the "segdef" error with any DCT/file size. Tested with 1700+ fields files, with a DCT of 150 files. New global options to handle the generation.
New: Run-time deactivation of the auditing is now possible.
New: Quick translation option. New TRN file outside the template for translating all messages and buttons. See "Translation options".
New: Inspection window now described in the manual.
Deprecated: "Big dct" control templates and extensions, no longer neccesary as the new "inspect" control template can handle much more then the "big dct" one. Important: If you were using these controls, you have to read Installation over previous versions and Generator: Unknown template type.
Imp: New embeds and better description on old ones.
Imp: Fields with array elements now handled inside a loop instead of field by field (code optimized).
Imp: Retry on error 83 (Record Is Already Held) saving audit record.
Chg: Global options now available to use control templates on multi-dll apps.
Chg: Some variables and internal names were in spanish, now they are in english. You should change a few variales in your DCT. Please see Installation over previous versions.

version 1.02 (11/10/2005):
Fix: Missing "primary file" on ABC english version of the template on RecoverAudit control template.
New: Documentation improved and on-line reference.

version 1.01 (9/10/2005):
Fix: Missing "primary file" on ABC english version of the template on ConsultaAudit control template.
Imp: Better handling of error 86 on audit file.

Part 4

4 Installation

FullRecord is a set of templates that are installed in your Clarion development environment with the aid of an installation program. Run the installation program and accept the defaults offered by it. The installation program offers you the option to register the FullRecord template in Clarion, but if you choose not to make use of this functionality, you have to start Clarion and manually register the FullRecord.TPL template (for ABC) and FullRecordLgcy.TPL (for Legacy). You will find these template files in the 3rdparty\template folder inside your Clarion's 6.x folder, or the Accessory\Template\Win folder for your Clarion's 7.x installation.

All our examples applications are installed in the examples\FullRecord2 folder, but the folder may be in different places according to your OS. Pay attention to the installer, it will propose a default folder - take note of where it is, and change it at that time if you don't like the default location.

<u>List of Example Applications</u>
All the examples for Clarion 7.x also work with Clarion 8.x.
All the examples for Clarion 6.x also work with Clarion 7.x or Clarion 8.x.

<u>Folder:</u> FullRecord\

Example	Clarion version	File driver	Comments
Fra6.app , fra6.dct	C6.3 – ABC	TPS	With BLOB fields and MEMO fields.
FRAIM6.app , fraIM6.dct	C6.3 – ABC	TPS	With BLOB fields , MEMO fields and InMemory tables.
frdata6.app , frinspect6.app , frmod6.app , frexe6.app , fra6.dct	C6.3 – ABC	TPS	Multi .DLL , with BLOB fields and MEMO fields.
frl6.app , frl6.dct	C6.3 – Legacy	TPS	With BLOB fields and MEMO fields.
frldata6.app , frlinspect6.app , frlmod6.app , frlexe6.app , frl6.dct	C6.3 – Legacy	TPS	With BLOB fields and MEMO fields.
rsa6.app , fra6.dct	C6.3 – ABC	TPS	With BLOB fields and MEMO fields. + NeatMessage template
Fra7.app , fra7.dct	C7.3 – ABC	TPS	With BLOB fields and MEMO fields.
FRAIM7 , fraIM7.dct	C7.3 – ABC	TPS	With BLOB fields , MEMO fields and InMemory tables.
frdata7.app , frinspect7.app , frmod7.app , frexe7.app , fra7.dct	C7.3 – ABC	TPS	Multi .DLL , with BLOB fields and MEMO fields.
frl7.app , frl7.dct	C7.3 –	TPS	With BLOB fields and MEMO fields.

	Legacy		
frldata7.app , frlinspect7.app , frlmod7.app , frlexe7.app , frl7.dct	C7.3 – Legacy	TPS	With BLOB fields and MEMO fields.
rsa7.app , fra7.dct	C7.3 – ABC	TPS	With BLOB fields and MEMO fields. + NeatMessage template

Folder: FullRecord\MSSQL\

Example	Clarion version	File driver	Comments
FR6ABSQL.app , FR6SqlB.dct	C6.3 – ABC	MSSQL	With BLOB field.
FR6LBSQL.app , FR6SqlB.dct	C6.3 – Legacy	MSSQL	With BLOB fields.
FR6LSQL.app , FR6Sql.dct	C6.3 – Legacy	MSSQL	Without BLOB fields.
FR6SQL.app , FR6Sql.dct	C6.3 – ABC	MSSQL	Without BLOB fields.
FR7ABSQL.app , FR7SqlB.dct	C7.3 – ABC	MSSQL	With BLOB fields.
FR7LBSQL.app , FR7SqlB.dct	C7.3 – Legacy	MSSQL	With BLOB fields.
FR7LSQL.app , FR7Sql.dct	C7.3 – Legacy	MSSQL	Without BLOB fields.
FR7SQL.app , FR7Sql.dct	C7.3 – ABC	MSSQL	Without BLOB fields.

Folder: FullRecord\ ODBC-FIREBIRD\DSN-Less\

Example	Clarion version	File driver	Comments
FRFBA6.app , FR6.dct	C6.3 – ABC	ODBC / FireBird	Without BLOB fields.
FRFBA7.app , FR7.dct	C7.3 – ABC	ODBC / FireBird	Without BLOB fields.

Folder: FullRecord\ ODBC-FIREBIRD\SQL-BLOB\

Example	Clarion version	File driver	Comments
FR6ABSQL.app , FR6SqlB.dct	C6.3 – ABC	ODBC / FireBird	With BLOB fields.
FR6LBSQL.app , FR6SqlB.dct	C6.3 – Legacy	ODBC / FireBird	With BLOB fields.
FR7ABSQL.app , FR7SqlB.dct	C7.3 – ABC	ODBC / FireBird	With BLOB fields.
FR7LBSQL.app , FR7SqlB.dct	C7.3 – Legacy	ODBC / FireBird	With BLOB fields.

Folder: FullRecord\ ODBC-FIREBIRD\ SQL-WithOutBLOB\

Example	Clarion version	File driver	Comments
FR6LSQL.app , FR6SQL.dct	C6.3 – Legacy	ODBC / FireBird	Without BLOB fields.
FR6SQL.app , FR6SQL.dct	C6.3 – ABC	ODBC / FireBird	Without BLOB fields.
FR7LSQL.app , FR7SQL.dct	C7.3 – Legacy	ODBC / FireBird	Without BLOB fields.
FR7SQL.app , FR7SQL.dct	C7.3 – ABC	ODBC / FireBird	Without BLOB fields.

Folder: FullRecord\ ODBC-GUID\

Example	Clarion version	File driver	Comments
FRPGA6GUID.app , FR6GUID.dct	C6.3 – ABC	ODBC / PostgreSQL	Without BLOB fields. Use GUID method for autonumbering in Audit files .
FRPGL6GUID.app , FR6GUID.dct	C6.3 – Legacy	ODBC / PostgreSQL	Without BLOB fields. Use GUID method for autonumbering in Audit files .
FRPGA7GUID.app , FR7GUID.dct	C7.3 – ABC	ODBC / PostgreSQL	Without BLOB fields. Use GUID method for autonumbering in Audit files .
FRPGL7GUID.app , FR7GUID.dct	C7.3 – Legacy	ODBC / PostgreSQL	Without BLOB fields. Use GUID method for autonumbering in Audit files .

Folder: FullRecord\ ODBC-PgSQL\DSN-Less\

Example	Clarion	File driver	Comments

	version		
FRPBA6.app , FR6.dct	C6.3 – ABC	ODBC / PostgreSQL	Without BLOB fields.
FRPBA7.app , FR7.dct	C7.3 – ABC	ODBC / PostgreSQL	Without BLOB fields.

Folder: FullRecord\ ODBC-PgSQL\SQL-BLOB\

Example	Clarion version	File driver	Comments
FR6ABSQL.app , FR6SqlB.dct	C6.3 – ABC	ODBC / PostgreSQL	With BLOB fields.
FR6LBSQL.app , FR6SqlB.dct	C6.3 – Legacy	ODBC / PostgreSQL	With BLOB fields.
FR7ABSQL.app , FR7SqlB.dct	C7.3 – ABC	ODBC / PostgreSQL	With BLOB fields.
FR7LBSQL.app , FR7SqlB.dct	C7.3 – Legacy	ODBC / PostgreSQL	With BLOB fields.

Folder: FullRecord\ ODBC-PgSQL\ SQL-WithOutBLOB\

Example	Clarion version	File driver	Comments
FR6LSQL.app , FR6SQL.dct	C6.3 – Legacy	ODBC / PostgreSQL	Without BLOB fields.
FR6SQL.app , FR6SQL.dct	C6.3 – ABC	ODBC / PostgreSQL	Without BLOB fields.
FR7LSQL.app , FR7SQL.dct	C7.3 – Legacy	ODBC / PostgreSQL	Without BLOB fields.
FR7SQL.app , FR7SQL.dct	C7.3 – ABC	ODBC / PostgreSQL	Without BLOB fields.

Folder: FullRecord\ SQLAnywhere\

Example	Clarion version	File driver	Comments
FR6ABSQL.app , FR6SqlB.dct	C6.3 – ABC	SQL Anywhere	With BLOB fields.

FR6LBSQL.app , FR6SqlB.dct	C6.3 – Legacy	SQL Anywhere	With BLOB fields.
FR6LSQL.app , FR6Sql.dct	C6.3 – Legacy	SQL Anywhere	Without BLOB fields.
FR6SQL.app , FR6Sql.dct	C6.3 – ABC	SQL Anywhere	Without BLOB fields.
FR7ABSQL.app , FR7SqlB.dct	C7.3 – ABC	SQL Anywhere	With BLOB fields.
FR7LBSQL.app , FR7SqlB.dct	C7.3 – Legacy	SQL Anywhere	With BLOB fields.
FR7LSQL.app , FR7Sql.dct	C7.3 – Legacy	SQL Anywhere	Without BLOB fields.
FR7SQL.app , FR7Sql.dct	C7.3 – ABC	SQL Anywhere	Without BLOB fields.

Notes on the examples:
RSA6.APP
This example use the NeatMessage template to record the messages to messages database and check for the "don't ask again" feature.

FRFBA6.APP
This example uses Firebird files (via ODBC) and is based on ABC. This example app was created with Clarion 6.3, and it should work with any previous version of Clarion 6. You need Firebird installed on your computer to run this example. See the ABC - FullRecord (Without NeatMessage) chapter for more details.
This example shows the recovering of records using ODBC and SQL syntax.

FR6SQL.APP
This example uses MS-SQL files (via the Clarion MS-SQL driver) and is based on ABC. This example app was created with Clarion 6.3, and it should work with any previous version of Clarion 6. You need MS-SQL Server installed on your computer (any version) to run this example. See the ABC - FullRecord (Without NeatMessage) chapter for more details.
This example shows the use of server-side autonumbering of the Audit file.

FR6LSQL.APP
This example is the same as the example above except for the fact that it is based on the Legacy (Clarion) template chain.

FRAIM6.APP
This example uses Clarion’s IMDD files and is based on ABC. This example was created with Clarion 6.3, and it should work with any previous version of Clarion 6. You need the IMDD driver installed on your computer to run this example.

Part 5

5 Installation over previous versions

Installation over versions prior to 3.00 (Upgrade from v2.x to 3.x)

There are a lot of internal changes in the new version 3.00, but we tried to keep the upgrade the easiest possible.
1) It is better if you totally uninstall and delete any traces of the previous versions.
2) The templates changed names, so you have to unregister the previous version and register the new one. Open your template registry, and unregister any FullRecord 2.x template. If you still have the old templates on the templates folder, delete them. Then register the new ones (FullRecord.tpl and FullRecordLgcy.tpl).
3) check the changes in the documentation, there are new information in the Using TXD Analyzer section, General Tab, Definitions 2 Tab, Local Extension and others.
4) Change your Audit files in the DCT. There are several new fields for the Audit file, and a new file. We recommend you take the Audit.DCT dictionary and copy the changes and new files from that DCT to yours.
5) After change the dictionary, make sure there are no empty fields in the global extension (As noted specially in Definitions 2 Tab).
6) Don't forget to convert your current audit files if you want to keep accessing them.

Installation over versions prior to 2.00 (Changes in v2.00)

Note that you cannot use FullRecord 2.x with Clarion 5.5 or earlier. FullRecord 2.x will work only with Clarion 6 and later.
If you added manual code to audit embedded legacy code, you may leave it or remove it, that code will be simply ignored in this version (Otherwise you would be auditing the same code twice).
The information stored in the Audit file is slightly different:
For Inserts: if the file has an autonumber key, now you will see the real behaviour of the program, which is and ADD operation with almost no data, and a CHANGE operation with the added data.
For Changes: the "Read" operation is no longer always necessary and it is not stored anymore if not, unless the ID field actually changed or there is no previous information (like when you "clean up" the audit file). The "old" read information can be left in the Audit file or it can be erased for shrink the database.

Installation over versions prior to 1.82 (Changes in v1.82)

If you are upgrading from a version older than 1.80, then you need to import the new BrowseAuditField procedure into your app and the rest of the section don't apply.
The "BrowseAuditField" you already imported in version 1.8x into your app requires a new parameter. You need to change it from what you have now:
(*? AField,STRING SearchKey,STRING FileName,LONG MemoNumber)
To what is needed:
(*? AField,STRING SearchKey,STRING FileName,LONG MemoNumber,STRING ScreenPicture)
Add the new STRING ScreenPicture parameter both in "Prototype" and in "Parameters".
If you don't do this, you will get an error at compile time:
Syntax error: No matching prototype available

Due to the parameter change, a full recompile will be needed.

We are going to use a new field "FileVersion" in the audit files (Check the supplied DCT).

The audit files will need to be converted to include the new field.

Installation over versions prior to 1.80 (Changes in v1.80)

For implement the new "Field Inspect" feature a key in the Audit file needs to change. You need to change and convert your current Audit file in order to use the new feature.

The old key
bySearchKey (KEY(AUD:SearchKey,AUD:InternalNumber),DUP)
need to be changed to
bySearchKey (KEY(AUD:SearchKey,AUD:FileName,AUD:InternalNumber),DUP).
The FileName field need to be added after the search key.

Installation over versions prior to 1.60 (Changes in v1.60)

The recording of procedures names was improved in this version. A global threaded queue is necessary for this change. You have to change the dictionary and point to the new fields in the General Tab. Instead of the single "procedure name" field in previous versions, now exists a queue with a single field (the procedure name inside the queue), with the following structure:

```
FRGLO:ProcNameQueue  QUEUE,PRE(),THREAD
PName                  STRING(14)
                     END
```

The "Inspect" procedures are no longer generated as local procedures. If you have the " View Record" control template populated in the Audit Browse, and you don't have the "Generate Global Procedures" option in the Inspect Tab active, you have to active this option for the Inspect Window to work. Previously if you don't had this option active, local code were generated instead of global code; now, no code is generated, thus you won't be able to inspect the stored data, if you don't activate this option.

The "View memo" new feature (enabled by default) for the inspection window requires a new button to be added to the window structure. If you don't want to use this new feature, you can disable it from the Inspect Tab.

Installation over versions prior to 1.50 (Changes in v1.50)

There are many new features in version 1.50. However, they are not enabled by default, therefore after installation your programs should continue to work the same way without problems. You can enable the new features later as needed.
To enable the global compression feature, you need to convert your existing Audit files.
To enable the recording of the procedure name and workstation, you need to add new variables to the dct. You may copy & paste them from the example audit.dct file and convert your audit file as well.
Important: As the TPS example files changed, you should erase (or manually convert) your old ones prior to test the new examples, otherwise you will get and error 47.

Installation over versions prior to 1.33 (Changes in v1.33)

As the global procedures now generate several procedures per module, delete the old .clw and .obj files prior to recompile to clean up the garbage. If you don't, you won't get any problem with your program but you will have some (probably a lot of) no longer used files in your folders.

Installation over versions prior to 1.31 (Changes in v1.31)

If you have a custom TRN file, note that there are two new entries in it, they are:

JALWantToRenumber and JALRenumberTitle.

Installation over versions prior to 1.20 (Changes in v1.20)

An embed changed its internal name; if you had code in this embed, it will be as "orphaned". In that case you have to manually move your code to the embed again. The embed description is: 'FullRecord: After Open(window)'

As the global procedures now generate with full long names, delete the old .clw and .obj files prior to recompile to clean up the garbage. If you don't, you won't get any problem with your program but you will have some (may be a lot of) no longer used files in your folders.

Installation over versions prior to 1.18 (Changes in v1.18)

There are new variables that you have to define in the global template extension. Make sure to enter the Global extension and fill them before recompile your app.

Installation over versions prior to 1.10 (Changes in v1.10)

Some DCT variables have been changed from spanish to english.
If you are upgrading from version 1.0x you should change these variables in your own DCT as well:

Previous name	New name
FRGLO:COLA_Buffer	FRGLO:Queue_Buffer
FRGLO:Archivo	FRGLO:FileName
FRGLO:Registro	FRGLO:TheRecord

If you were using the "big dct" control and extension templates, you will see some errors upon opening your app with the new template (See "Generator: Unknown template type"). You have to delete the remains of those controls and repopulate the new ones.

Installation over beta version

If you install this version over a beta, it is recommended to uninstall the previous version first, before you install this one. Due to the big changes in beta versions and in version 1.00, if you already started to call the procedures from source code, please read the present manual before start to work with the current version.

Part 6

6 Quick Start

6.1 Single EXE

Import the definitions into your DCT.

1. Open your DCT
2. Export it to TXD (by using the "File","Export text" option inside the Clarion Dictionary Editor).
3. From the FullRecord example folder (Inside your Clarion folder, look for 3rdParty\Examples\FullRecord), run the TXDAnalyzer program against the TXD just generated.
4. While you are in your Dictionary, import the audit.txd file into your Dictionary. The Audit.TXD file is also located in the FullRecord example folder.
5. Review the file definitions. The example txd was created with two memo and no blob fields. If you have files that will be audited that contain more than two memo fields or blob fields, you should alter the audit file to include those fields. See the Definitions 1 Tab for a more detailed explanation. Use the results obtained from the TXDAnalyzer program to adjust the memo and record length size settings in the audit files your DCT.
6. Save your DCT, close it and open your APP.

Define the global settings in your APP.

7. Press the "Global" button to go to the global area of the APP.
8. Press the "Extensions" button to open the global extensions area.
9. Press the "Insert" button.
10. Select the "FullRecord" global extension from the list of extensions.
11. If you want to record the recovered records (recommended), select the "Definitions 3" tab and check the "Record Recover Information" checkbox.
12. Select the "Files to Audit" tab and select the files you want to audit. If you want to do a quick test of FullRecord's functionality, just click the "Select All" button. You may also import the file list generated by the TXDAnalyzer program.
13. Press "Ok"
14. Press "Ok" again.
15. Legacy only: Enable the "Enable the use of ABC classes" in the "Classes" Tab, and enable "Enable triggers support" in the "File Control Flags" Tab.

Import the Inspect Procedures into your APP.

16. Choose the FRA6.app example (you will find it in the example folder) for ABC or FRL6. app for Legacy.
17. Delete the global extension and do NOT save the app, unless you made a backup.

Do a selective Export to TXA of the procedures you want to import (For example, the BrowseAudit procedure). You may select one or more of the following procedures:
CleanAudit
BrowseAudit
BrowseAuditBck
BrowseAuditField
BrowseAuditRecord, and
BrowseAuditExtended

18. Close the example app without saving it.
19. Open your app and import the TXA you just created.

20. Add a menu item or items in your main menu to call the procedures CleanAudit, BrowseAudit, BrowseAuditBck and BrowseAuditExtended if you are going to use them. The other procedures will be called automatically by the template.
21. Compile and go.

MDI WARNING.

If your program has NON-MDI windows or menus, you will need to remove the "MDI Child" attribute from the FullRecord's procedures to avoid a run-time error. You can do that either when creating the TXA or after importing to the new app.

6.2 Multi-DLL

Import the definitions into your DCT.

1. Open your DCT
2. Export it to TXD (by using the "File","Export text" option inside the Clarion Dictionary Editor).
3. From the FullRecord example folder (Inside your Clarion folder, look for 3rdParty\Examples\FullRecord), run the TXDAnalyzer program against the TXD just generated.
4. While you are in your Dictionary import the audit.txd file into your Dictionary. The Audit. TXD file is also located in the FullRecord example folder.
5. Review the file definitions. The example txd was created with two memo and no blob fields. If you have files that will be audited that contain more than two memo fields or blob fields, you should alter the audit file to include those fields. See the Definitions 1 Tab for a more detailed explanation. Use the results obtained from the TXDAnalyzer program to adjust the memo and record length size settings in the audit files your DCT.
6. Save your DCT, close it and open your APP.

Define the global settings in your Data-APP.

7. Press the "Global" button to go to the global area of the data-APP (The data-app is the one where you export all the files and variables definitions).
8. Press the "Extensions" button to open the global extensions area.
9. Press the "Insert" button.
10. Select the "FullRecord" global extension from the list of extensions.
11. If you want to record the recovered records (recommended), select the "Definitions 3" tab and check the "Record Recover Information" checkbox.
12. Select the "Files to Audit" tab and select the files you want to audit. If you want to do a quick test of FullRecord's functionality, just click the "Select All" button. You may also import the file list generated by the TXDAnalyzer program.
13. Press "Ok"
14. Press "Ok" again.
15. Legacy only: Enable the "Enable the use of ABC classes" in the "Classes" Tab, and enable "Enable triggers support" in the "File Control Flags" Tab.

Import the Inspect Procedure into one of your APP.

16. Choose the FRA6.app example (you will find it in the example folder) for ABC, or FRL6. app for Legacy.
17. Delete the global extension and do NOT save the app.
18. Do a selective Export to TXA of the procedures you want to copy (For example, the

BrowseAudit procedure).
You may select one or more of the following procedures:
CleanAudit
BrowseAudit
BrowseAuditBck
BrowseAuditField
BrowseAuditRecord, and
BrowseAuditExtended
19.Close the example app without saving it.
20.Open your app and import the TXA you just created.
21.Press the "Global" button to go to the global area of each other APP (including the main app).
22.Press the "Extensions" button to go to the global extensions area.
23.Select the "FullRecord" extension and activate the "use local procedures in this app" option.
24.Press "Ok" and Enter again into the extension, you will have most of the options visible now.
25.Select the "Files to Audit" tab and select the files you want to inspect. You should pick the same files you selected in point 12.

Define the global settings in each other APP.

26.Press the "Global" button to go to the global area of each other APP (including the main app).
27.Press the "Extensions" button to go to the global extensions area.
28.Press the "Insert" button.
29.Select the "FullRecord" global extension from the extension list.
30.Select the "Files to Audit" tab and select the files you want to inspect. Normally you would pick the same files you selected in point 12 (You can use the TXD Analyzer generated file).
31.Press "Ok"
32.Press "Ok" again.
Steps 26 through 32 are necessary to enable the "Record Inspect" and "Field Inspect" features calls from all the procedures.

Add menu item to your Main app.
33.Add a menu item or items in your main menu to call the procedures CleanAudit, BrowseAudit, BrowseAuditBck and BrowseAuditExtended if you are going to use them. The other procedures will be called automatically by the template.
34.Compile everything and go.

MDI WARNING.

If your program has NON-MDI windows or menus, you will need to remove the "MDI Child" attribute from the FullRecord's procedures to avoid a run-time error. You can do that either when creating the TXA or after importing to the new app.

Part 7

7 Using TXD Analyzer

A companion program is supplied with FullRecord. This program will let you analyze your dictionary and easily pick the correct values for FullRecord's variables. It will also generate a "file list" for you to import it into the FullRecord Global Template Settings.

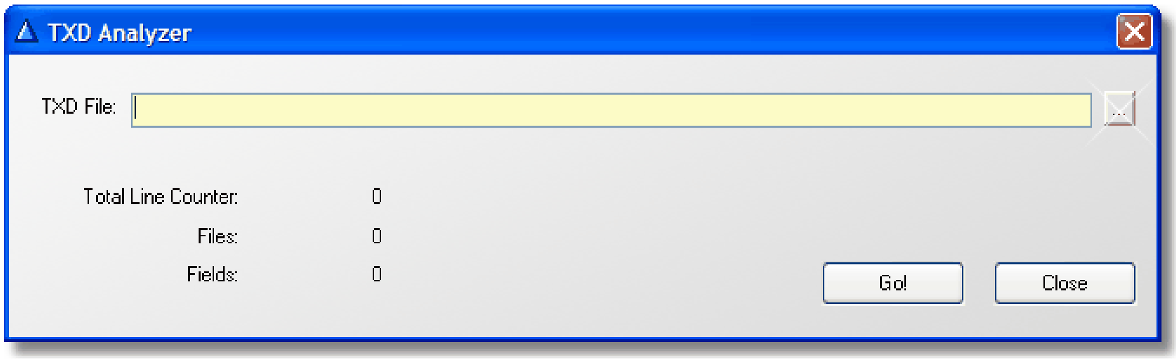
Before using the Analyzer, you have to export your DCT to a TXD file.
For Clarion 6.x: From within your Clarion Dictionary Editor, select the "file" menu item and then the "export text" option.
For Clarion 7.x or Clarion 8.x: From the DCT Explorer toolbar, you should see a button whose tool tip says "Import/Export". On that button, select "Export Dictionary to Text" When the dialog to export pops up, you will see the DCTX extension by default. Select "All Files" from the File of Type Drop List, and then enter "anyname.TXD".
When you give the file name a TXD extension, the export process will save as TXD instead of DCTX.

Launch the TXDAnalyzer.exe program using the shortcut in your FullRecord menu (or look for it with Windows Explorer). The "Actions" menu has two options, "Pick TXD" and "Browse Files".

The "Pick TXD" option let you start a new project, while the "Browse Files" option let you continue working with an existing project.

7.1 Pick TXD

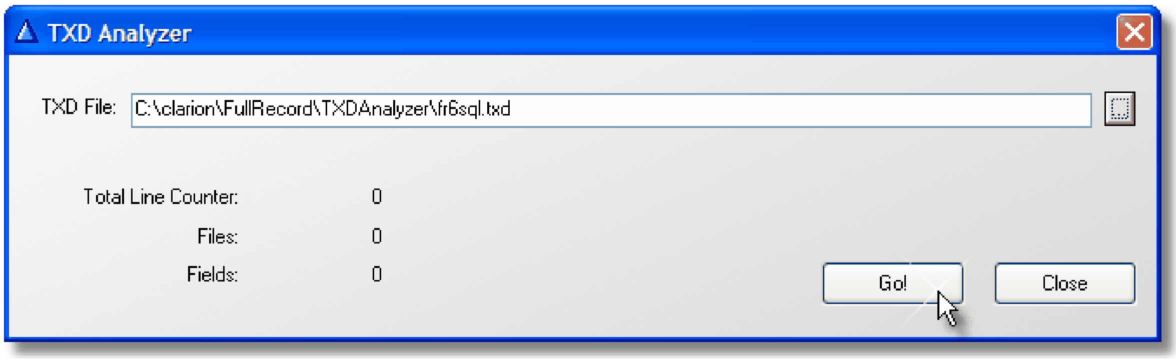
The "Pick TXD" option will show the following window:



You can use the lookup button (the ... button to the right) to search for a TXD file, or you can type it manually in the "TXD File" field (if you are really brave). If you use the lookup button, a standard Windows File Open dialog window will be displayed to allow you to pick a TXD file.

Once you selected the TXD file, you can press the "Go!" button to process it. When you process a new TXD, all your previous workings will be erased.

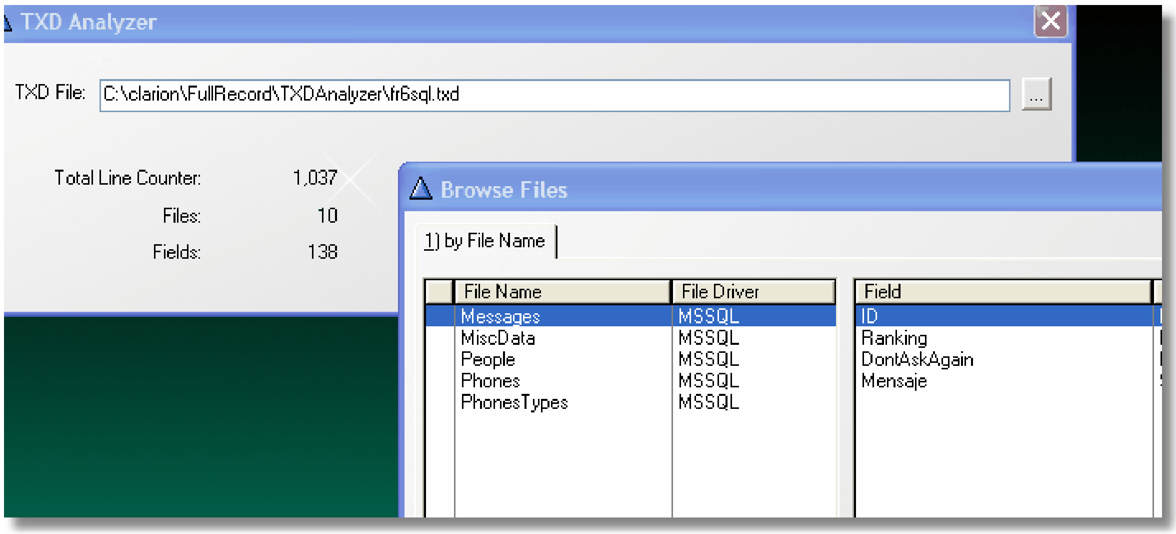
TIP: If you want to keep different jobs associated with different TXDs, copy the TXDAnalyzer program to the TXD folders and execute the program in that folder. The individual working files will be created in the current folder.



TXD File: C:\clarion\FullRecord\TXDAnalyzer\fr6sql.txd

Total Line Counter: 0
Files: 0
Fields: 0

When the analyze process is complete the Browse Files window will open immediately, and you will see the processed totals in the background, showing how many Files and Fields were found in your TXD.



TXD File: C:\clarion\FullRecord\TXDAnalyzer\fr6sql.txd

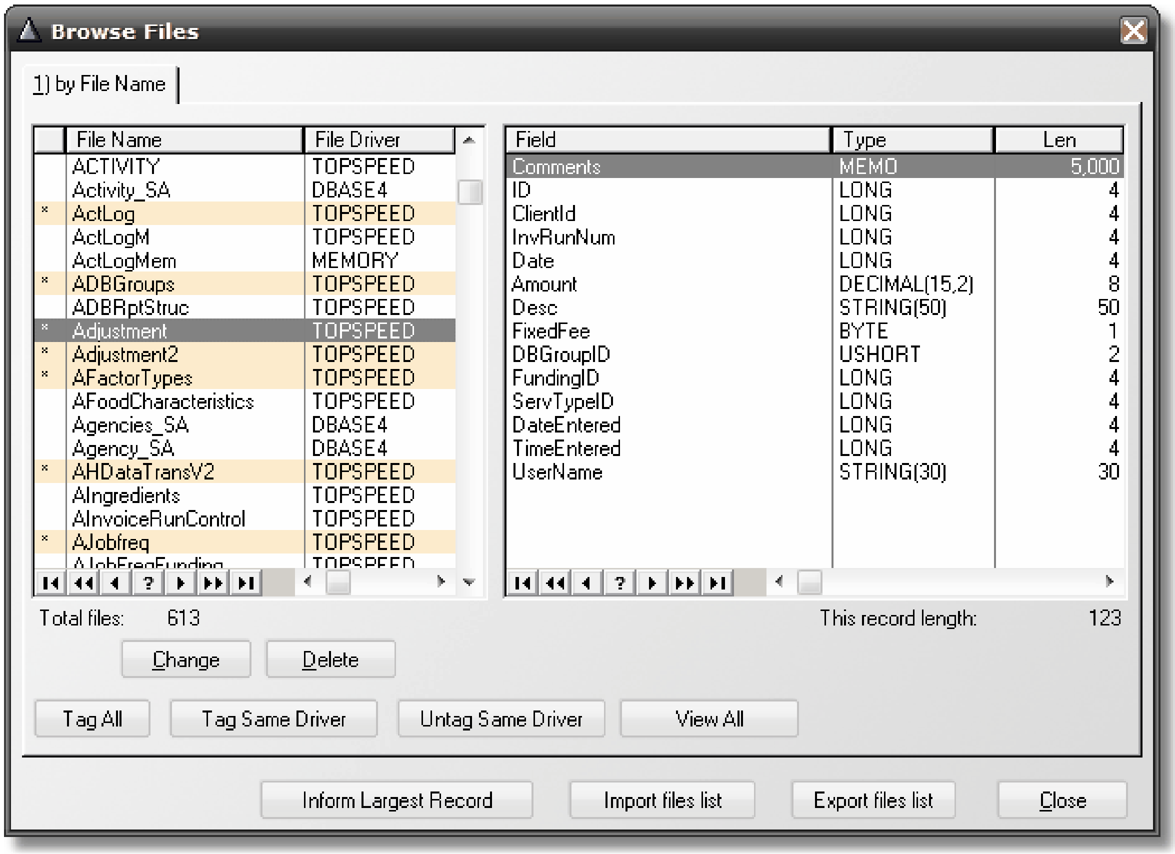
Total Line Counter: 1,037
Files: 10
Fields: 138

File Name	File Driver
Messages	MSSQL
MiscData	MSSQL
People	MSSQL
Phones	MSSQL
PhonesTypes	MSSQL

Field
ID
Ranking
DontAskAgain
Mensaje

7.2 Browse Files

When you enter this procedure, you will see the following window:



	File Name	File Driver
	ACTIVITY	TOPSPEED
	Activity_SA	DBASE4
*	ActLog	TOPSPEED
	ActLogM	TOPSPEED
	ActLogMem	MEMORY
*	ADBGroups	TOPSPEED
	ADBRptStruc	TOPSPEED
*	Adjustment	TOPSPEED
*	Adjustment2	TOPSPEED
*	AFactorTypes	TOPSPEED
	AFoodCharacteristics	TOPSPEED
	Agencies_SA	DBASE4
	Agency_SA	DBASE4
*	AHDataTransV2	TOPSPEED
	AIngredients	TOPSPEED
	AInvoiceRunControl	TOPSPEED
*	AJobfreq	TOPSPEED

Field	Type	Len
Comments	MEMO	5,000
ID	LONG	4
ClientId	LONG	4
InvRunNum	LONG	4
Date	LONG	4
Amount	DECIMAL(15,2)	8
Desc	STRING(50)	50
FixedFee	BYTE	1
DBGroupID	USHORT	2
FundingID	LONG	4
ServTypeID	LONG	4
DateEntered	LONG	4
TimeEntered	LONG	4
UserName	STRING(30)	30

Total files: 613

This record length: 123

This window shows the list of Files found in your TXD file, in alphabetic order. In the list on the right, you will see a list of Fields belonging to the selected File, in the same order as in your DCT.

You can use the "Change" button to change the name of a file, if the name of the file was imported incorrectly from the TXD. You can use the "Delete" button to permanently erase a file from the list.

Tag All button: With this button you can select all the Files in the list at once. When you press this button, all the Files are selected, and the Tag All button changes to "Untag All". So, if you press the button again, you will unselect all the Files.

Manually tagging files: If you select a file in the list with a mouse click, it will change its tagged status; if it is not selected, it will be selected and if it is selected, it will unselected.

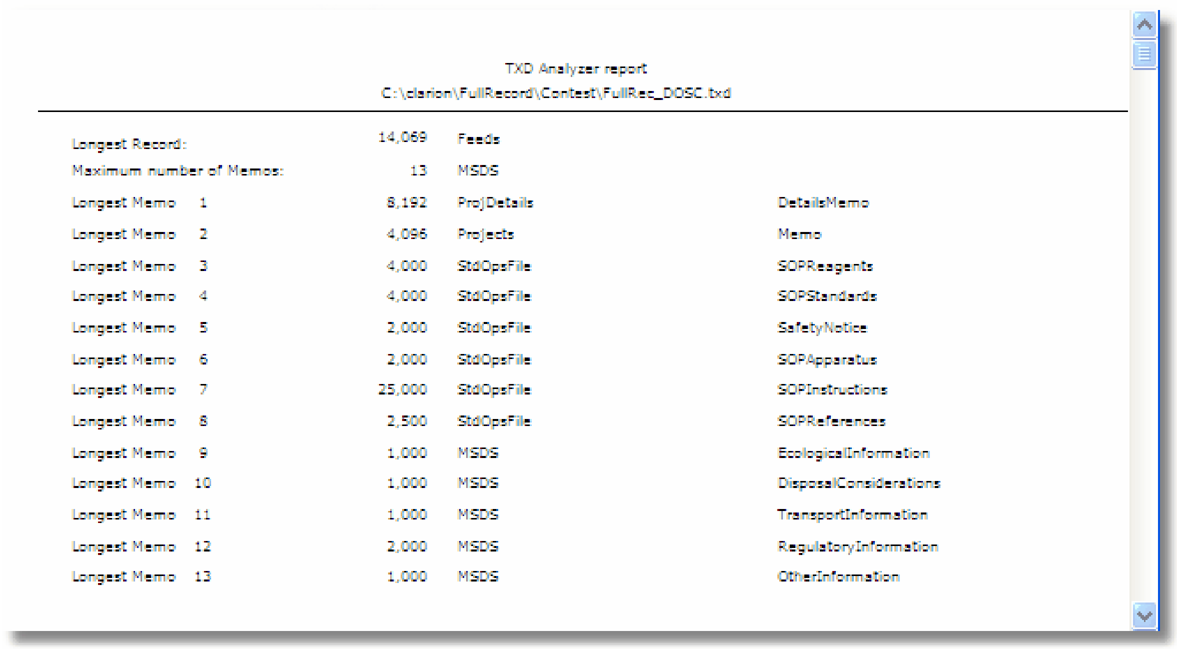
Tag Same Driver: This button will select all the files with the same driver than the selected one, including itself.

Untag Same Driver: This button will unselect all the files with the same driver than the selected one, including itself.

View All: By default, the FullRecord "Audit" files are excluded from the list. Pressing this button you will see them, and you can even select them (but it doesn't make sense to select the audit files because that will cause FullRecord to audit the Audit files... that's why they are disabled by default). When you press this button it changes to "Filter Audit Files". Pressing this button again will once again hide the Audit files.

7.2.1 Inform Largest Record

Inform Largest Record: Once you have selected the files you want to be audited, you can press this button to get a report that will assist you to configure your FullRecord Global variables and files.
The values shown in the report below are just an example, yours will be different.

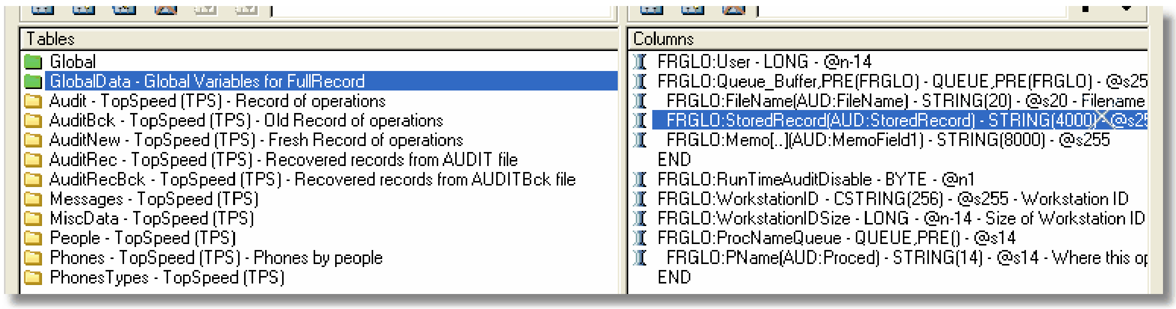


```
                         TXD Analyzer report
           C:\clarion\FullRecord\Contest\FullRec_DOSC.txd

Longest Record:                  14,069   Feeds
Maximum number of Memos:             13   MSDS
Longest Memo    1                 8,192   ProjDetails      DetailsMemo
Longest Memo    2                 4,096   Projects         Memo
Longest Memo    3                 4,000   StdOpsFile       SOPReagents
Longest Memo    4                 4,000   StdOpsFile       SOPStandards
Longest Memo    5                 2,000   StdOpsFile       SafetyNotice
Longest Memo    6                 2,000   StdOpsFile       SOPApparatus
Longest Memo    7                25,000   StdOpsFile       SOPInstructions
Longest Memo    8                 2,500   StdOpsFile       SOPReferences
Longest Memo    9                 1,000   MSDS             EcologicalInformation
Longest Memo   10                 1,000   MSDS             DisposalConsiderations
Longest Memo   11                 1,000   MSDS             TransportInformation
Longest Memo   12                 2,000   MSDS             RegulatoryInformation
Longest Memo   13                 1,000   MSDS             OtherInformation
```

After you have imported the corresponding Audit TXD (As described in the Template Usage chapter) into your DCT, you will use the data from this report to change your Audit files and global variables declared in your DCT, as follows:

Longest Record: You have to change the StoredRecord field in each area of your DCT, that's the GlobalData area, the Audit File, the AuditNew file and the AuditBck file.



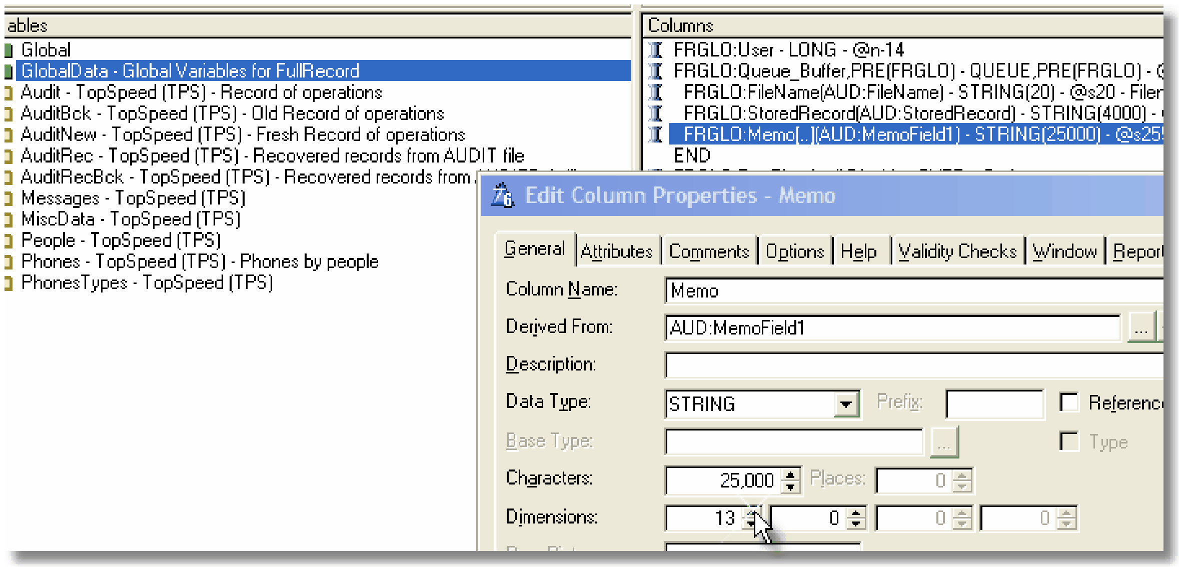
```
Tables
Global
GlobalData - Global Variables for FullRecord
Audit - TopSpeed (TPS) - Record of operations
AuditBck - TopSpeed (TPS) - Old Record of operations
AuditNew - TopSpeed (TPS) - Fresh Record of operations
AuditRec - TopSpeed (TPS) - Recovered records from AUDIT file
AuditRecBck - TopSpeed (TPS) - Recovered records from AUDITBck file
Messages - TopSpeed (TPS)
MiscData - TopSpeed (TPS)
People - TopSpeed (TPS)
Phones - TopSpeed (TPS) - Phones by people
PhonesTypes - TopSpeed (TPS)
```

```
Columns
FRGLO:User - LONG - @n-14
FRGLO:Queue_Buffer,PRE(FRGLO) - QUEUE,PRE(FRGLO) - @s25
  FRGLO:FileName(AUD:FileName) - STRING(20) - @s20 - Filename
  FRGLO:StoredRecord(AUD:StoredRecord) - STRING(4000) - @s2
  FRGLO:Memo[..](AUD:MemoField1) - STRING(8000) - @s255
END
FRGLO:RunTimeAuditDisable - BYTE - @n1
FRGLO:WorkstationID - CSTRING(256) - @s255 - Workstation ID
FRGLO:WorkstationIDSize - LONG - @n-14 - Size of Workstation ID
FRGLO:ProcNameQueue - QUEUE,PRE() - @s14
  FRGLO:PName(AUD:Proced) - STRING(14) - @s14 - Where this op
END
```

Change the default length of 4000 to the size indicated in the "Longest Record" line. You can leave a small margin if you want, just in case. If you change the StoredRecord's field type to Blob this step is not required (like for example if you use

MSSQL as the Audit file driver).

Maximum number of memos: This number becomes the DIM number of the named "Memo" field in FullRecord's global data area.



ables
- Global
- GlobalData - Global Variables for FullRecord
- Audit - TopSpeed (TPS) - Record of operations
- AuditBck - TopSpeed (TPS) - Old Record of operations
- AuditNew - TopSpeed (TPS) - Fresh Record of operations
- AuditRec - TopSpeed (TPS) - Recovered records from AUDIT file
- AuditRecBck - TopSpeed (TPS) - Recovered records from
- Messages - TopSpeed (TPS)
- MiscData - TopSpeed (TPS)
- People - TopSpeed (TPS)
- Phones - TopSpeed (TPS) - Phones by people
- PhonesTypes - TopSpeed (TPS)

Columns
```
FRGLO:User - LONG - @n-14
FRGLO:Queue_Buffer,PRE(FRGLO) - QUEUE,PRE(FRGLO) - @
  FRGLO:FileName(AUD:FileName) - STRING(20) - @s20 - Filer
  FRGLO:StoredRecord(AUD:StoredRecord) - STRING(4000) -
  FRGLO:Memo[..](AUD:MemoField1) - STRING(25000) - @s25
END
```

Edit Column Properties - Memo

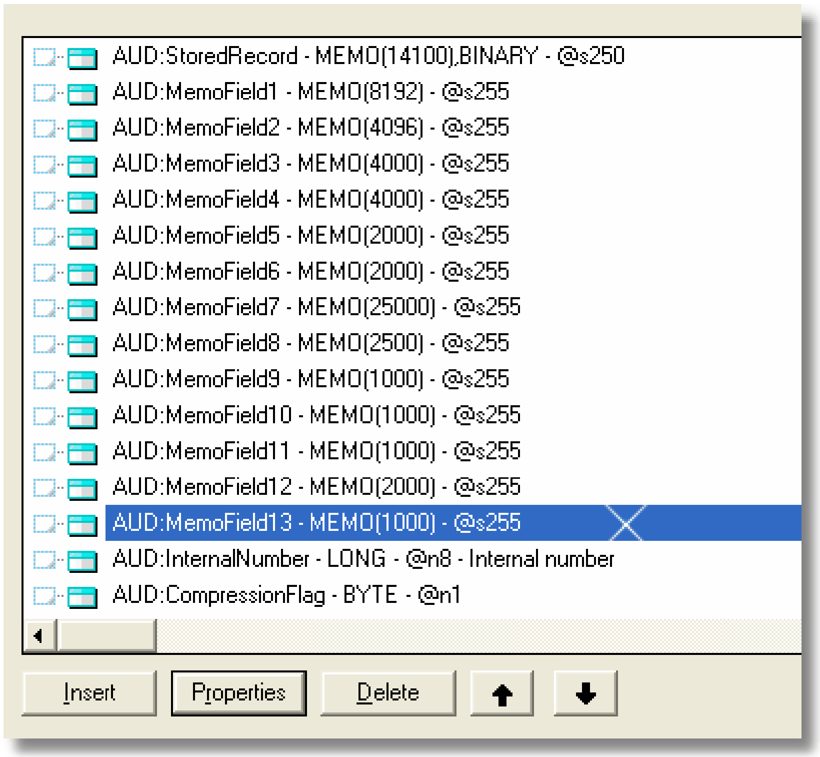
General | Attributes | Comments | Options | Help | Validity Checks | Window | Report

Column Name:	Memo
Derived From:	AUD:MemoField1
Description:	
Data Type:	STRING
Base Type:	
Characters:	25,000
Dimensions:	13

You have to change the first "Dimensions" value to the number indicated in the "Maximum number of memos" field.
You also have to change the "Characters" value to the longest of all the listed memos. This is also mentioned in the General Tab explanation.

Longest Memo: For the Audit, AuditBck and AuditNew files, you have create memo fields for the number of memos reported here, and with the same length as in this report.

Here is an example screen:

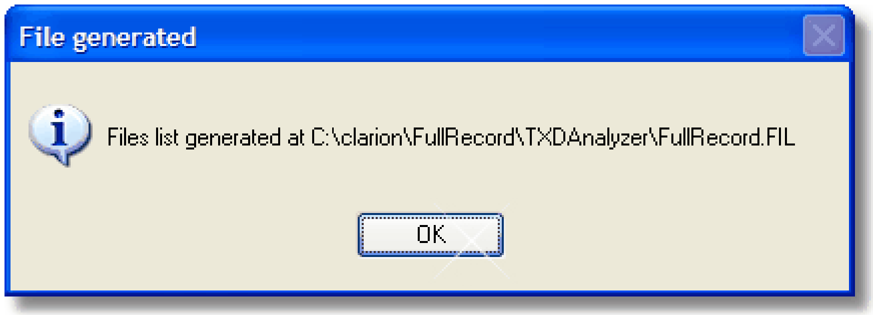


AUD:StoredRecord - MEMO(14100),BINARY - @s250
AUD:MemoField1 - MEMO(8192) - @s255
AUD:MemoField2 - MEMO(4096) - @s255
AUD:MemoField3 - MEMO(4000) - @s255
AUD:MemoField4 - MEMO(4000) - @s255
AUD:MemoField5 - MEMO(2000) - @s255
AUD:MemoField6 - MEMO(2000) - @s255
AUD:MemoField7 - MEMO(25000) - @s255
AUD:MemoField8 - MEMO(2500) - @s255
AUD:MemoField9 - MEMO(1000) - @s255
AUD:MemoField10 - MEMO(1000) - @s255
AUD:MemoField11 - MEMO(1000) - @s255
AUD:MemoField12 - MEMO(2000) - @s255
AUD:MemoField13 - MEMO(1000) - @s255
AUD:InternalNumber - LONG - @n8 - Internal number
AUD:CompressionFlag - BYTE - @n1

7.2.2 Export Files List

Export Files List: When you press this button a text file named FullRecord.FIL will be created in the working folder. This file will contain a list of all the current selected files.

The following message will pop up after the operation is completed:



File generated

Files list generated at C:\clarion\FullRecord\TXDAnalyzer\FullRecord.FIL

As can be seen in the message above a file named FullRecord.FIL will be created in the path as indicated by the message. If the folder is not the one where your app resides, you can manually move this file into your app's folder.

The FullRecord.FIL file is a plain text file, and you can manually edit or create this file if you like.

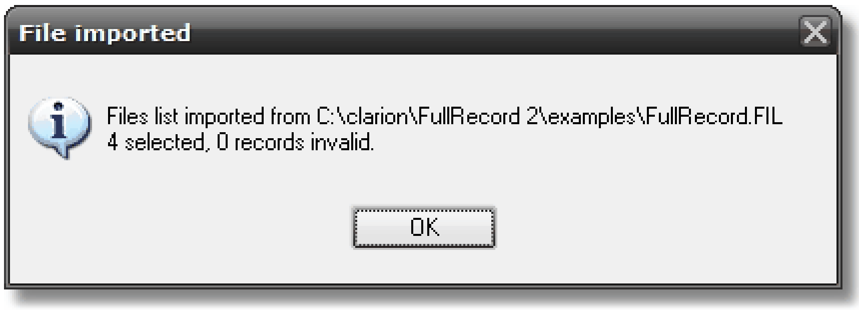
The benefit of creating this file is that it can be imported directly into the FulllRecord's Global Extension template (See the Files to Audit Tab). This will save you the time that it takes to select all the files that you want to be audited inside the template and in the case of a multi-dll app it will save you the selection of these file in each and every app that makes up the multi-dll system.

You can also import this file at a later stage into TXD Analyzer to activate your current selection if you change your DCT and have to analyze it again (for example adding or removing files) (See Import Files List).

7.2.3 Import Files List

Import Files List: When you press this button the text file FullRecord.FIL found in the working path, will be examined. Each file found in the text file will be compared against the current opened file list, and if found the file will be selected in the current opened file list.

The following message will pop up after the operation is completed:



Files list imported from C:\clarion\FullRecord 2\examples\FullRecord.FIL
4 selected, 0 records invalid.

This option is handy in the following situation: Suppose you made a complex file selection in a big TXD. Now, you change (for example add or remove) files in the DCT and after that you have to generate a new TXD to analyze. The moment you do that you will lose all your selections.
If you import your previously created FullRecord.FIL text file you can reactivate your previous selections so that you don't have to select individual files again. You will only have to check the new files you added to your dictionary and select the ones you need to be audited.

Part 8

8 Template Usage

As a first step import the AUDIT.TXD file into your dictionary. The sample Audit.TXD file can be found in the example folder (See the Guide for Examples). If you have any problem importing it, please let us know about your problem. The alternative option available to you is to "copy & paste" the data from the example dictionary AUDIT.DCT into your dictionary.

You have to import the following into your dictionary:

- The global data for FullRecord (GlobalData)
- The AUDIT file, to record the movements of your files.
- The Messages file, to record the messages that you use in your system.

Optionally:

- If you plan to keep track of the undeletion of records, you have to import the AUDITRec file.
- If you plan to clean the audit file at times, you need to import the AUDITBck and AUDITNew files, and then the AUDITBckRec file for cleaning of the undeletion track as well.

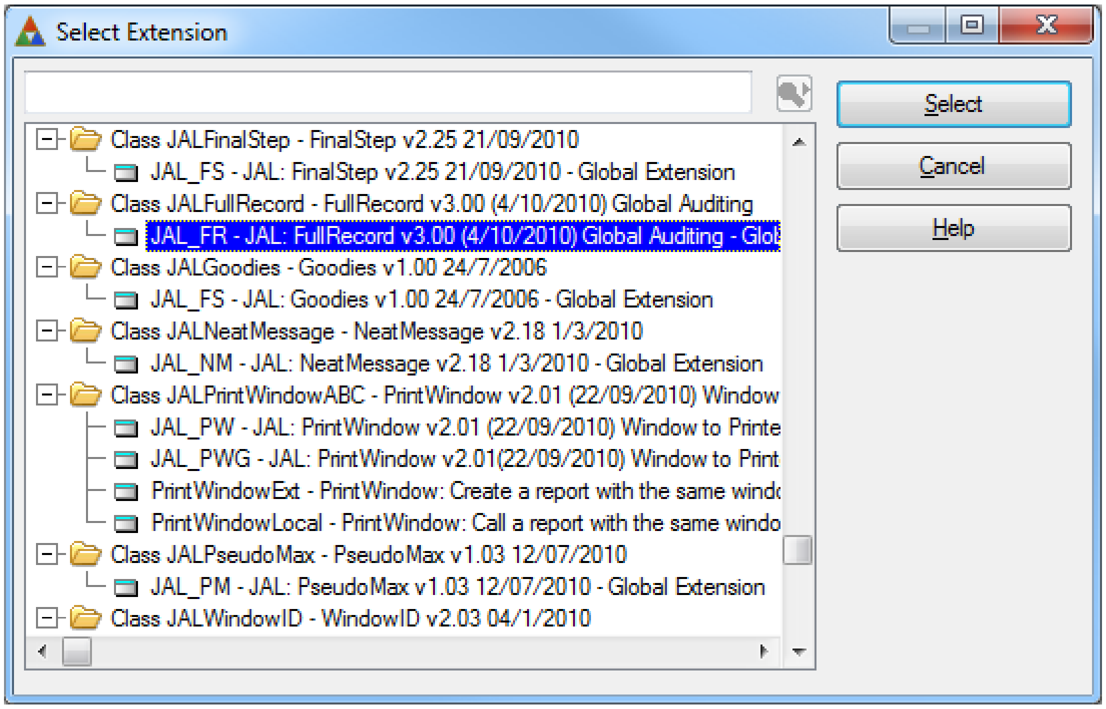
All the files and global variables are present in audit.txd. You may import it, and later delete what you don't need.

Clarion 7 warning: If you import the TXD into the Clarion DCT several times, you will end with several copies of the same files (unlike Clarion 6.x, Clarion 7.x renames the files so they are not overwritten). So if you need or want to import the files more than once, you have to revert to a previous copy of the DCT, or you have to delete the audit files, prior to import the TXD.

8.1 Global extension

You have to insert the FullRecord's Global Extension in every app of your system, but only after you have prepared your DCT (See the Quick Start and Template Usage).

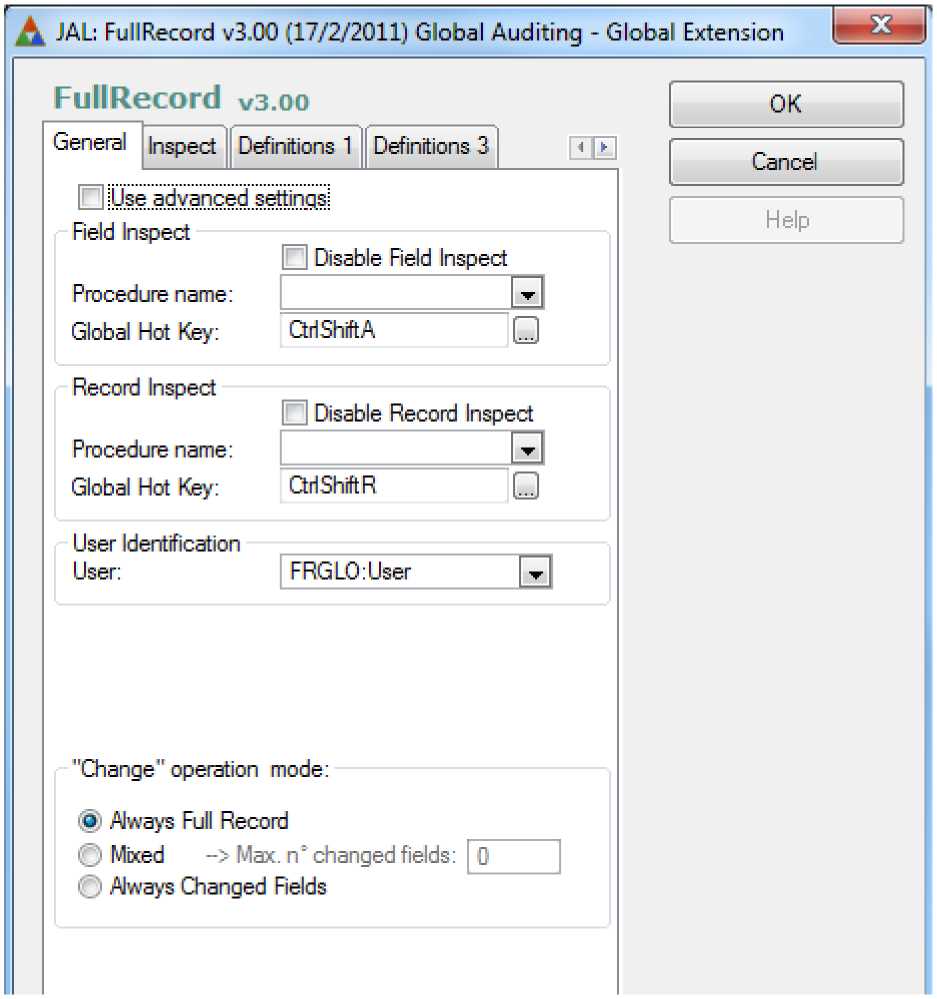
To insert FullRecord's Global Extension, click on "Global", "Extensions" and select the global extension "JAL: FullRecord v.... Global Extension".



Class JALFinalStep - FinalStep v2.25 21/09/2010
JAL_FS - JAL: FinalStep v2.25 21/09/2010 - Global Extension
Class JALFullRecord - FullRecord v3.00 (4/10/2010) Global Auditing
JAL_FR - JAL: FullRecord v3.00 (4/10/2010) Global Auditing - Glob
Class JALGoodies - Goodies v1.00 24/7/2006
JAL_FS - JAL: Goodies v1.00 24/7/2006 - Global Extension
Class JALNeatMessage - NeatMessage v2.18 1/3/2010
JAL_NM - JAL: NeatMessage v2.18 1/3/2010 - Global Extension
Class JALPrintWindowABC - PrintWindow v2.01 (22/09/2010) Window
JAL_PW - JAL: PrintWindow v2.01 (22/09/2010) Window to Printe
JAL_PWG - JAL: PrintWindow v2.01(22/09/2010) Window to Print
PrintWindowExt - PrintWindow: Create a report with the same windc
PrintWindowLocal - PrintWindow: Call a report with the same windo
Class JALPseudoMax - PseudoMax v1.03 12/07/2010
JAL_PM - JAL: PseudoMax v1.03 12/07/2010 - Global Extension
Class JALWindowID - WindowID v2.03 04/1/2010

Note: the current version may be different than the one shown above (The current one is 3.10).

If you installed the FullRecord's global extension correctly, you should see this window:


JAL: FullRecord v3.00 (17/2/2011) Global Auditing - Global Extension
FullRecord v3.00
General
Inspect
Definitions 1
Definitions 3
OK
Cancel
Help

Use advanced settings

Field Inspect

Disable Field Inspect

Procedure name:

Global Hot Key: CtrlShiftA

Record Inspect

Disable Record Inspect

Procedure name:

Global Hot Key: CtrlShiftR

User Identification

User: FRGLO:User

"Change" operation mode:

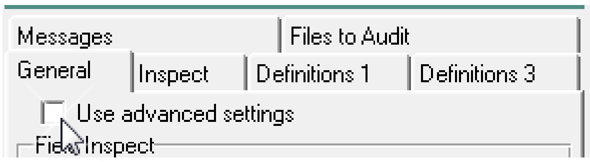
[✓ Always Full Record]

Mixed --> Max. n° changed fields: 0

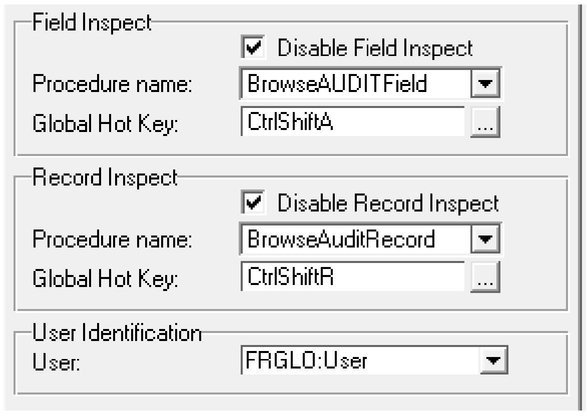
Always Changed Fields

8.1.1 General Tab

"General" Tab - Basic Settings



The Use advanced settings checkbox allows you to toggle between the Basic and the Advanced interface. The Basic interface has everything you need to use the FullRecord template to its full extent without any clutter, while the Advanced interface gives you the chance to customize the template to the full.
Some tabs like the Global Tab will be available only in advanced mode.



Field Inspect
- [✓] Disable Field Inspect
- Procedure name: BrowseAUDITField
- Global Hot Key: CtrlShiftA

Record Inspect
- [✓] Disable Record Inspect
- Procedure name: BrowseAuditRecord
- Global Hot Key: CtrlShiftR

User Identification
- User: FRGLO:User

Disable Field Inspect: Check this option if you want to disable this feature. Uncheck it to enable this feature.

Field Inspect Procedure name: Select the procedure from the drop down list which will allow you to inspect the changes made to a particular *field*. If you leave this field blank, it will default to BrowseAuditField. If you want to *disable* this feature, you need to tick the "Disable Field Inspect" checkbox above this option.
In a multi-dll system, this Inspect procedure needs to be in the data-dll app to make use of the Template defaults. If you place it in another app, you will have to change several things manually which is possible, although not advised.

Global Hot Key: This is the key you will press when a field on any of your application's window is selected and you want to see the change history for that field. By default it is set to Ctrl+Shift+A but you can change it to whatever you want, or leave it blank to disable this feature.
Note: when you disable this feature by leaving the Hot Key blank, no code related to the field inspect feature is actually generated.

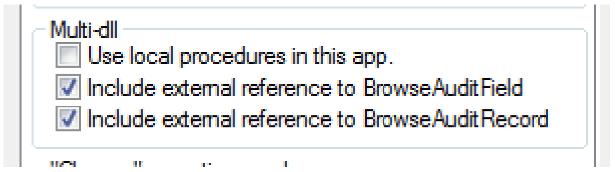
Disable Record Inspect: Check this option if you want to disable this feature. Uncheck it to enable this feature.

Record Inspect Procedure name: Select the procedure from the drop down list which will allow you to inspect the changes made to a particular *record*. If you leave this field blank, it will default to BrowseAuditRecord. If you want to *disable* this feature, you need to tick the "Disable Record Inspect" checkbox above this option.
In a multi-dll system, this Inspect procedure needs to be in the data-dll app to make use of the Template defaults. If you place it in another app, you will have to change several things manually which is possible, although not advised.

Global Hot Key: This is the key you will press in any Form or Browse window to see the change history for the selected record. By default it is set to Ctrl+Shift+R but you can change it to whatever you want, or leave it blank to disable this feature.
Note: when you disable this feature by leaving the Hot Key blank, no code related to the record inspect feature is actually generated.

User identification: The "User" global variable is the variable that will receive the active user code or name for your system. This variable should be populated by your login security system, and may be any data type you want (either for a code or to store a full name). In the FullRecord examples our variable makes use of a numeric code.

Options available for multi-dll systems:

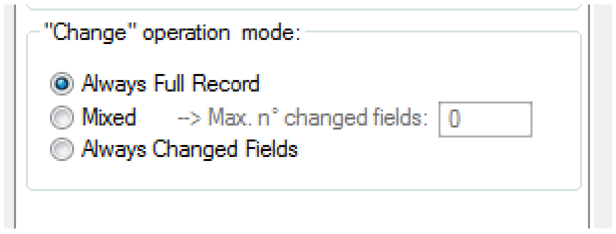

Multi-dll
[ ] Use local procedures in this app.
[✓] Include external reference to BrowseAuditField
[✓] Include external reference to BrowseAuditRecord

Use local procedures in this app: This option is only visible if your app is a dll and it is not the data dll, it will also be hidden if your app's target is “exe”. When you tick this checkbox you will be able to activate all the Template tabs, for using the control templates in this particular app.
You can see this option being used in a multi-dll example, check FullRecord's Global Extension in FRInspect55.app or FRInspect66.app.
Note: You may need to close FullRecord's Global Extension and reopen it to make all the tabs visible after activating this option.

Include external reference to BrowseAuditField: In a multi-dll system, you'd normally have to include an "external" reference to the BrowseAuditField procedure for each app that has a reference to the data app. If you leave the default procedure name as is, you can tick this checkbox and the external reference will automatically be included for you.

Include external reference to BrowseAuditRecord: In a multi-dll system, you'd normally have to include an "external" reference to the BrowseAuditRecord procedure for each app that has a reference to the data app. If you leave the default procedure name as is, you can tick this checkbox and the external reference will automatically be included for you.

"Change" operation mode:


"Change" operation mode:
● Always Full Record
○ Mixed --> Max. n° changed fields: 0
○ Always Changed Fields

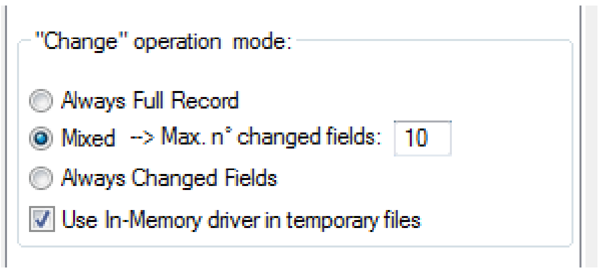
This part allow you to select how the audit will work for you in this program. There are three modes available, that will allow you to select how the CHANGES to your bases are saved. INSERTS and DELETES are always saved as Full Records.

Always Full Record
This is the same as FullRecord 2.x. In this mode, no matter how many fields are changed at once, the program will save the full record for the file changed. This is convenient if you changed many fields at once, but it can be a space waster if you change only a field or two each time.
If you pick this mode you will work pretty much the same as previous versions of FullRecord.

Always Changed Fields

In this new mode FullRecord will save to the audit file one record per changed field, containing the previous value of the field, and the new (current) value of the field. If you often change only one or two fields, this should save you a lot of space. However, if you change most of the fields of the record, there are two disadvantages, one is you will end with twice as many information saved as in the single-whole record mode, and two, as this mode saves one record per field changed, it can be deadly slow. For example, if you change 20 fields, this will save 20 records to the audit file (as opposite to a single record -or at worst two- for the FullRecord mode).

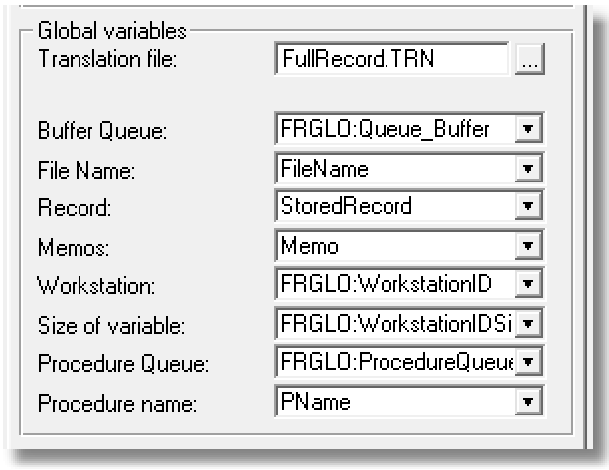
Mixed
Trying to look for the best of both worlds, we developed this mode where if the number of changed fields are up to so many, the changed fields only are saved, and beyond that number, the whole record is saved. You have to set up the nnumber in the "Max. no. changed fields" entry that is enabled when you pick this mode.
For example, if you enter "6" here, when you change the value of one, two, three or up to 6 fields, the system will save one record per field changed. If you change 7 fields or more, the system will save only one "full" record to the audit base.
So this mode will try to balance speed with file size convenience.


"Change" operation mode:
- ○ Always Full Record
- ● Mixed --> Max. n° changed fields: 10
- ○ Always Changed Fields
- ☑ Use In-Memory driver in temporary files

When you use **Mixed** or **Always Changed Fields** mode, the **Use In-Memory driver in temporary files** option becomes enabled.
If you select the **Use In-Memory driver in temporary files** checkbox, a temporary file needed to store some blob fields will be create in memory, using SoftVelocity's **In Memory Data Driver** (purchased separately, directly from Softvelocity). If you enable this option and you don't have the driver, you'll get compilation errors.
If you don't select this option the temporary files will be created on disk, in the OS's temporal folder for the current user.

8.1.2 Global Tab (Advanced)

"Global" Tab - Advanced Settings



Global variables	
Translation file:	FullRecord.TRN
Buffer Queue:	FRGLO:Queue_Buffer
File Name:	FileName
Record:	StoredRecord
Memos:	Memo
Workstation:	FRGLO:WorkstationID
Size of variable:	FRGLO:WorkstationIDSi
Procedure Queue:	FRGLO:ProcedureQueue
Procedure name:	PName

The "Global variables" area refers to the global data area in your DCT and APP.

Translation file: The "translation file" is a text file where you store all the translatable messages to the user (See Translation options). If you don't include a path (like in this example), Clarion's RED file is used to locate the file.

Buffer Queue: The "Buffer Queue" global variable is the global queue which is going to store the buffer for saving the last record read for each file. As such, the queue have to contain one entry per file, with the filename, the record itself and the "memo" fields. These variables have to be declared in your DCT.

Memos: The "Memos" field will record the last memos accessed. As it is not possible to define a global variable as type memo, this field needs to be defined as a STRING. Due to the fact that you may have several memo fields in any file, this field should be Dimensioned. The dimension setting needs to be equal to the maximum number of memos that you may have in a single file. This may be difficult to calculate if you have a great number of Files. To simplify this task, we provide you with the TXD Analyzer companion program (See Using TXD Analyzer and Inform Largest Record), this program calculates the number you have to put into the dimension field for you.

If you still want to calculate the number manually, follow these guidelines.
For example: say you have the following files;
A: contains no memo fields
B: contains memo1 of 5000 bytes, and memo2 of 3000 bytes.
C: contains memo1 of 8000 bytes.
D: contains memo1 of 800 bytes, memo2 of 3600 bytes and memo3 of 1200 bytes.
From the example above, the "memos" variable should be Dimensioned to a value of 3 (file D contains the highest number of memo fields) and 8000 bytes long (the biggest memo field as in file C).

Workstation: The "Workstation" variable will hold the computer name where the audit operation took place. This variable will be populated by an API call and as such, it has to be declared as a CSTRING(256) in your DCT. If you don't want to store the workstation name where the audit operation took place, leave this entry blank.
Size of variable: The "Size of variable" variable will hold the size of the Workstation variable for the API call. It should be a type LONG variable. If you completed a Workstation variable, you should define this variable as well.

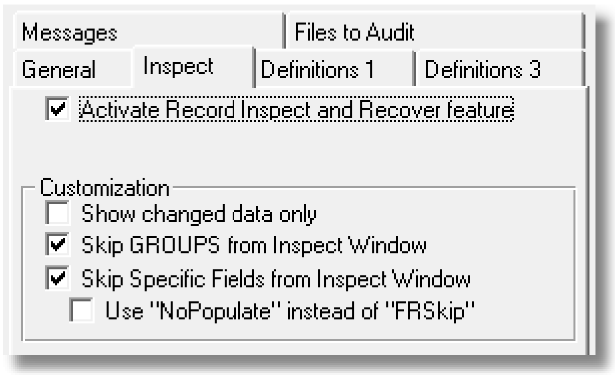
Procedure Queue and Procedure name: The "Procedure Queue" and "Procedure name" variables will be used to store the name of the current procedure in memory. The "Procedure name" variable should be a STRING and it should be large enough to store your longest procedure name. The "Procedure Queue" has to be declared as THREADED , otherwise it will fail to correctly register the name of the procedure when you open several windows at the same time.

You can change these names to any valid name you desire, or you can accept the defaults.

8.1.3 Inspect Tab

"Inspect" Tab - Basic Settings

In this window you define some values related to the View Record control template and the "inspection window".



- [✓] Activate Record Inspect and Recover feature
- [ ] Show changed data only
- [✓] Skip GROUPS from Inspect Window
- [✓] Skip Specific Fields from Inspect Window
- [ ] Use "NoPopulate" instead of "FRSkip"

Activate Record Inspect and Recover feature: If your app uses the ViewRecord control template then some internal procedures are needed. These procedures will be generated as global procedures in your program. To generate them as global in a standalone exe project you just enable the global option as shown above.
The "ViewRecord" control template is used in the procedures BrowseAudit, BrowseAuditBck and BrowseAuditRecord in FullRecord's examples.

In a multi-dll project you have to activate this option in the main data dll and in the app where you will use the control template. See the multi-dll example, where this option is activated in FRData66.app and in FRInspect66.app. To avoid the unnecessary generation of references to these procedures, don't activate this option in other apps or if you are not going to use the control template.

Beware: If you use the ViewRecord control template and you don't activate this option in the same app, no inspection code will be generated. There won't be any compiler errors, but as the inspection procedures are not there, you won't be able to see the stored record information. i.e. you will press the "View" button and the contents will be empty.

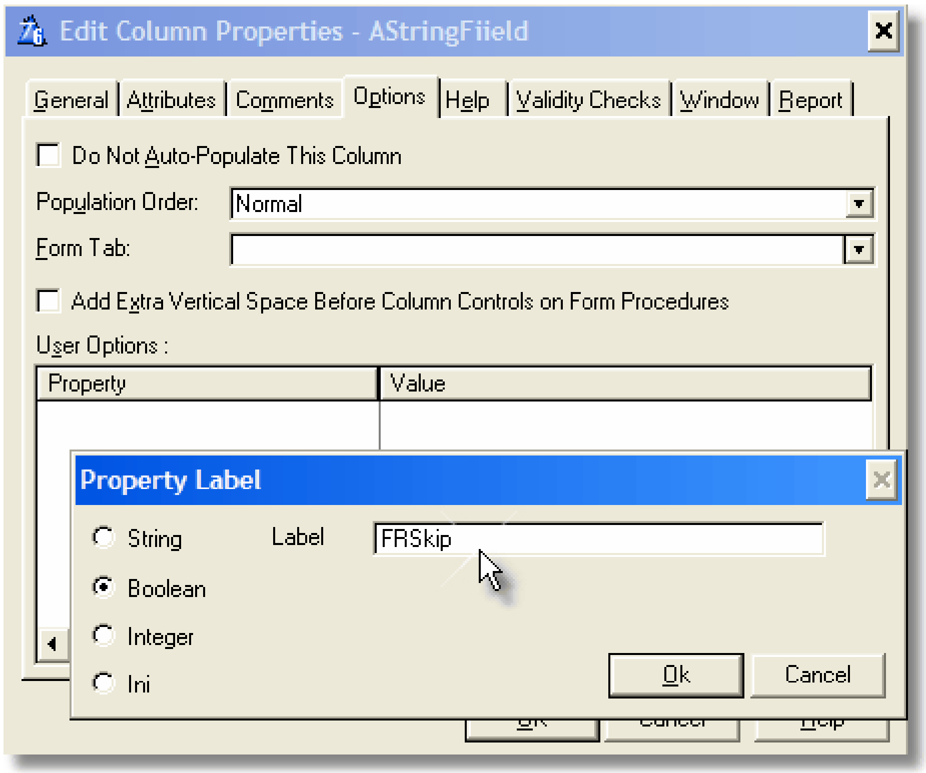
Customization: The inspection window has several customization options. The window itself is in an external text file that you can customize to your exact needs (See *External "window" file* below).

Show changed data only: If you activate this option, the inspection window that shows changed data will only show the changed fields, and not all the fields, with the changed ones highlighted.

Skip GROUPS from Inspect Window: Normally all fields are shown in the inspection window, but GROUPS often show meaningless information (as they may be composed of "packed" type fields). If you activate this option, GROUPed fields won't show in the inspection window.

Skip Specific Fields from Inspect Window: You may have fields that you don't want to be seen in the Inspect window (for example, password fields). You have two methods to force the Inspect procedure to skip them;
The first and default method is to add a boolean variable called FRSkip to the Field's User "Options" in the DCT, and set its value to TRUE or 1.
Steps to do this:

1. Select the field you want to exclude from the Inspect Window in the DCT and click on the "Options" tab.
2. Right click in the "User Options" listbox.
3. Choose "Insert", and enter a Label FRSkip as can be seen in the example screen below:



Do Not Auto-Populate This Column

Population Order: Normal

Form Tab:

Add Extra Vertical Space Before Column Controls on Form Procedures

User Options :

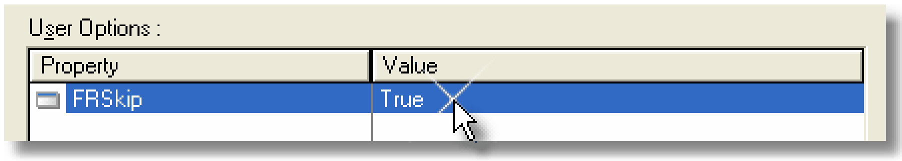
Property	Value

Property Label

String ( ) Boolean [✓] Integer ( ) Ini ( )

Label FRSkip

4. Select the label you have just created and set the variable to TRUE as can be seen below:



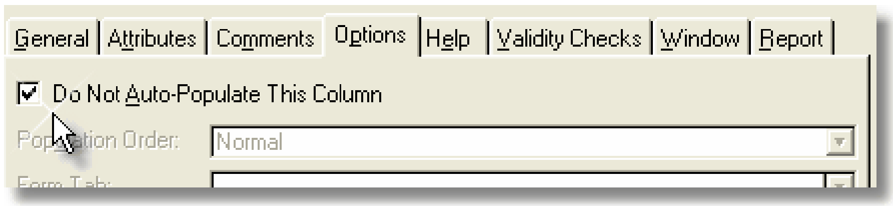
User Options :

Property	Value
FRSkip	True

If you don't want to skip this field at a later stage, you can delete the FRSKip label, or simply set it to FALSE.

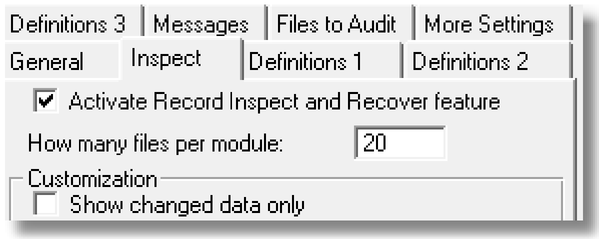
Note: The "skipped" fields won't show in the Inspect Window, but they are saved in the audit trail anyway. If you want to enable them at a later stage, you will see this field was saved in the old records.

Use "NoPopulate" instead of "FRSkip": This is dependent on the option above. An alternative for using the FRSkip variable is to simply use the "Do Not Auto-Populate This Column" option on the Field’s "Options" tab in the DCT. If you check this checkbox, the field will be skipped by the Inspect Window.



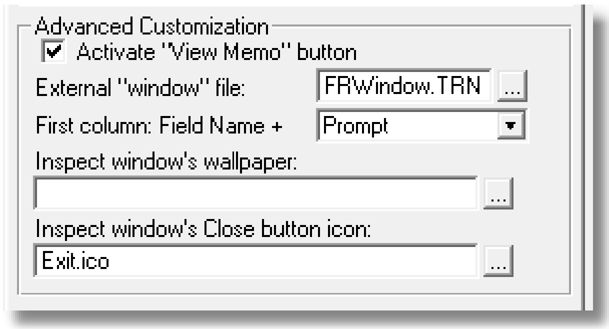
☑ Do Not Auto-Populate This Column
Population Order: Normal
Form Tab:

"Inspect" Tab - Advanced Settings



☑ Activate Record Inspect and Recover feature
How many files per module: 20
Customization
☐ Show changed data only

How many files per module: The "How many files per module" option allows you to specify how many procedures will be generated per module. In versions of FullRecord prior to version 1.33 only one procedure were generated per module, thus the generation and compilation process was very slow when the dictionary had quite a few files. Now you can specify a value of 20 or more procedures per module. This will speed the generation and compilation process up by a big margin. The higher you make this value the faster the generation and compilation will be, but you may get a "segdef" linker error. The default value of 20 should be more than adequate for most apps, thus this setting is considered to be an advanced setting.



Advanced Customization
☑ Activate "View Memo" button
External "window" file: FRWindow.TRN
First column: Field Name + Prompt
Inspect window's wallpaper:
Inspect window's Close button icon: Exit.ico

Activate "View memo" button (in the Inspect Window): When you activate this option the "View memo" button inside the inspection window will be enabled when you select a memo field (and disabled for other fields), and when you click on it you will see a window that contains all of the content of the memo field.
If you don't activate this option, you can remove the memo button from the Inspect window structure.

External "window" file: The "External "window" file" option allows you to specify where the text file is that contains the windows definitions. The default is FRWindow.TRN. *Do not confuse this file with the FullRecord.TRN file.* If you don't include a path (like in the example above), Clarion's RED file is used to locate the file.

If you change this file path and/or name, you have to ensure that a future upgrade of FullRecord doesn't override your changes.
All the Window's prompts and button prompts are in the FullRecord.TRN file so you can translate or change them as needed. See Translation options for further explanation.

Note: If you are upgrading from a version older than version 1.60 see Installation over previous versions as a new button needs to be added to the window structure to take advantage of the new "View memo" feature.

First column: Field name +...: You can customize the contents of the first column of the inspection window. You will see the Field name plus whatever you selected from the "First column" option:

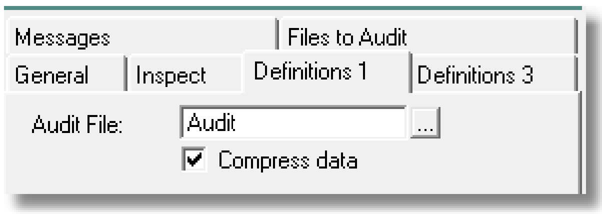
If you select "prompt", you will see the field name plus the prompt, while the other option will show the field name plus the short description (as entered in the DCT).

Inspect window's wallpaper and Close button icon: You can also select the window wallpaper and the "close" button's icon. These resources are automatically included in the global project of the app.

8.1.4 Definitions 1 Tab

"Definitions 1" Tab - Basic Settings

You have to specify the following in the main app, or in a multi-dll system, you have to specify it in the data dll app and you have to specify it in the app where you will use the ViewRecord control template (these can be the same app).

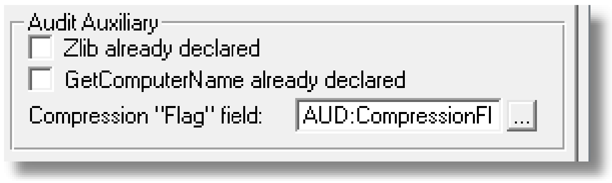


Audit File: Audit
[✓] Compress data

Audit File: First, you have to specify the name of your Audit file. This is the file where the audit operations are going to be recorded. In our example DCT it is the file called Audit.

Compress data: Below the Audit file name, you have to specify whether the audit data will be compressed. If you don't activate this option, the audit data will be stored in uncompressed format (as it was up to version 1.33 of FullRecord), and the template will work exactly as in versions before version 1.50.
If you choose to activate this option, all the audit data will be compressed before it is saved to file, and it will be decompressed after reading it from file.
Note: To activate this option, if all your audited *records* are very small (less than 40 bytes), may in fact produce bigger files due to the compression overheads. Compression is only a viable option when the record is bigger than 40 bytes. The bigger the record is, the “better” the compression result will be.

"Definitions 1" Tab - Advanced Settings



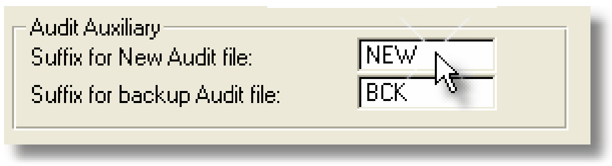
Audit Auxiliary
- [ ] Zlib already declared
- [ ] GetComputerName already declared

Compression "Flag" field: AUD:CompressionFl

Zlib already declared: If you own another product that uses zlib as compression library, you might get duplicate declarations in your global map and project, so to avoid that tick this check box or disable the zlib library declaration in the other 3rd party product.

GetComputerName already declared: If you own another product that define the GetComputerName API, you might get duplicate declarations in your global map. To avoid such a situation tick this checkbox.

Compression Flag: After the "Compress data" option you will see a "Compression Flag" variable. We strongly advise you to define this variable in your Audit file if you are going to use compression, although is not mandatory. If you leave this field empty while the "compress data" option is activated, the audit data will always be decompressed after reading and compressed prior to saving.
Note: Omitting this variable could lead to severe problems if you decide to change the status of the "Compress data" at some stage, as you might end up with compressed and uncompressed data mixed in the Audit file, and you won't be able to inspect data that isn’t in the same state as the global option. Using the "Compression flag", each record in the audit file will be identified as compressed or not, and the template will read them as such. By using this flag, the template will also be able to change the compression status of the records as and when required.



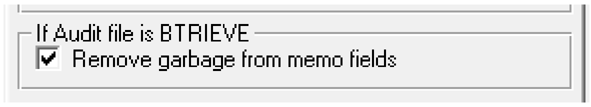
Audit Auxiliary

Suffix for New Audit file: NEW

Suffix for backup Audit file: BCK

Suffixes for New and Backup Audit files: You have to specify suffixes to create two files that are identical to the audit file. You should take extra care making sure that these two files stay in sync with the Audit file. The name of these files must be the name of the

Audit file plus a suffix. In FullRecord’s example, the Audit file is named AUDIT and the other two files should be named AUDITNEW and AUDITBCK. These three files have to be identical, although in a SQL environment the External Key names should be different to avoid problems with the SQL engine.
You can define these files as an ALIAS of the Audit file, although this is not automatically supported by FullRecord. While this is perfectly valid, it adds the complexity of having to manage the file names just before opening them, on each procedure. This exceeds the scope of FullRecord, which doesn't deal with that.



If Audit file is BTRIEVE, remove garbage from memo fields: Due to a bug in Clarion, if you use Btrieve files to store the Audit information the "memo" fields may be saved with some some “garbage” characters at the start of the memo field. When you tick this checkbox, FullRecord will attempt to clean the memo information before being saved to file.

8.1.5 Notes on Audit Data Compression

When you enable data compression for the Audit file, the record to be stored and the memo fields are compressed before being saved to disk and it is decompressed after being read from the disk. As the compression is done in memory, it is very fast, and as it reads and writes less data from and to the disk, it is possible to experience no performance delay during the auditing process. However, depending on your selected file driver storage, you may note a dramatic saving in the audit file size. You will have to experiment with your own configuration to see if this feature is a viable option or not.

The compression and decompression operations are performed by a open-source library called "zlib". You can download the latest version from www.zlib.net. For your convenience, FullRecord's setup installs the necessary libraries, which are zlib.dll, zlib.lib and zlibwapi.dll.
Note that when you use compression, even if you compile your project as local (Everything in the exe), zlib.dll will still be called by your exe, and it should be distributed with your program as an additional external dynamic link library.

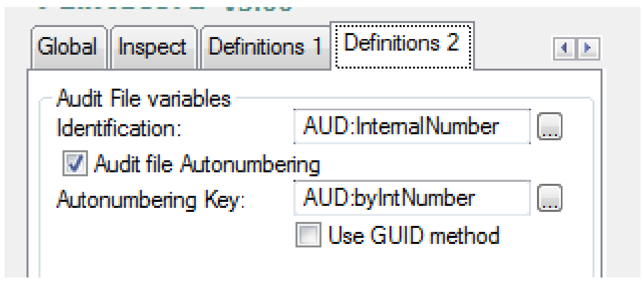
For very small records the compression may produce undesired results, due to some header overhead information required by the zlib library. Our tests have shown that compressing records smaller than 40 bytes produced records that were actually bigger than the original size. However, in records over 200 bytes in length the compression ratio was over 50%. This is highly dependent on the type of data you are saving and results may vary from test to test.

8.1.6 Definitions 2 Tab

"Definitions 2" Tab - Advanced Settings

This whole Tab is part of advanced settings and won't be visible if the basic interface is set.

In the box "Audit File variables", you can specify names for the fields that serve a purpose in the Audit file. This gives you the flexibility to use your own field and key names in the Audit files declared in the dictionary, and it gives you the option to add additional fields not included in the original Audit file.

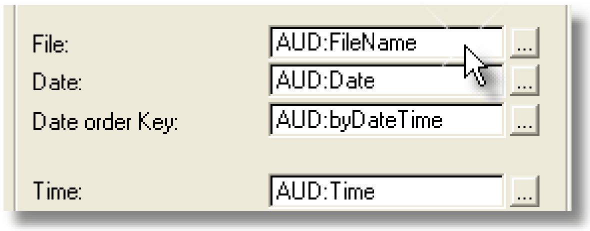


Audit File variables
Identification: AUD:InternalNumber
[✓] Audit file Autonumbering
Autonumbering Key: AUD:byIntNumber
[ ] Use GUID method

- Identification: This is the field that contains the unique ID for the audit records. Normally it is a type LONG field.
- Audit file auto numbering: If you use an ISAM file, you should always check this option. The audit file is normally auto numbered. This means that when a new record is added to the audit file, the system finds the last record stored, add 1 to the identification field and will store the new record with the new unique number. This operation is not done transaction framing, so it in a multi user environment it is possible that in the time between reading the last file and the recording of the new one, someone else might have used your number. The auditing system will detect a situation like this and will try to deal with it by retrying the operation.
  But if you use an SQL file and you decide that the server will take care of the auto numbering process, you may uncheck this mark so that the template doesn't try to auto number the record. If you decide to go this route you have to define the field as autonumbered (identity) in the SQL engine. If you don't activate this option and you forget to define this field as an identity field in the SQL database, you will get an error as the moment you add the second record to the audit file.

  Use GUID method: if you store the audit file in a SQL database, you may want to use this option to automatically generate a GUID identification. If that is the case, the "Identification" field has to be adecuate to store a GUID, not a LONG but a STRING(16) or CSTRING(17). A GUID value will be automatically generated by the template (instead of auto numbering the key).

- Auto numbering key: Specify which key in your Audit file is the key that has the auto numbering option on. Normally it is the primary key and it should only contain one component, the identification field.



File: AUD:FileName
Date: AUD:Date
Date order Key: AUD:byDateTime
Time: AUD:Time

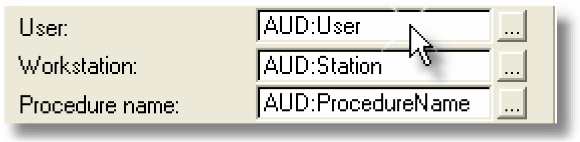
- File: Specify the field that will contain the name of the file that is being audited. It has to be a STRING variable that is big enough to store the longest name of all the files you have

in your dictionary. If you have files with names that are longer than 20 characters, you have to increase this field's size.
Note: If you change this field's length, you have to update the Audit file as well as the other 2 audit files (New and Bck). You also need to update length of the FileName in the Queue_Buffer in the global declaration called FRGlobalData.

- Date: Specify which field in the audit file will store the date that the record has changed. This field will be populated with the computer system date where the program is running. If the user tampers with the computer's machine date, this will be reflected in the audit file. You can use the identification field to sort the operations in their proper order and that way detect the out of sequence date. This field would normally be a LONG.

  Note: For SQL files, dates usually go into groups. If you "pick" a field inside a group with the lookup button, it won't be accepted because it lacks the prefix. Instead, you have to type it manually.

- Date order Key: Specify which key is the key where the first field in the key is the date field. This key will be used for the "Audit Cleanup" process.
- Time: Similar to Date except for the fact that it is using the system clock and not the system date..



User: AUD:User ...
Workstation: AUD:Station ...
Procedure name: AUD:ProcedureName ...

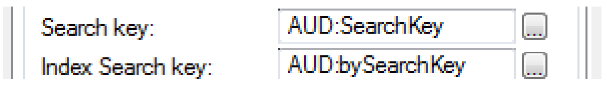
- User: This is the field in the audit file that stores the code or name of the user that is logged in to the system. Normally it should be the same as the global user variable. This variable is not mandatory: if you leave it blank no user information will be recorded. We choose to store the logged in user's numeric code, but if you prefer to store the name, you should change this field to STRING and make it big enough to hold the value.
- Workstation: This field in the audit file will store the computer's windows name where the operation took place. This may be a STRING as long as you need to store the longest machine name you may have. You may leave this field blank if you don't want to record the workstation name.
- Procedure name: This is the field in the audit file that stores the name of the procedure in your system that generated the operation. It is very useful to track the origin of this operation in your system. This may be a STRING as need to be big enough to store the longest procedure name you may have among all your apps. You may leave this field blank if you don't want to record the procedure name.

Operation: AUD:Operation ...

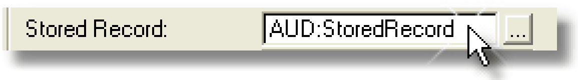
- Operation: This is the field in the audit file that stores the operation type performed by the system. This value is set in the Translation .TRN file. By default, it may have the following values:
  - AD (Add)
  - DE (Delete)
  - RE (Read; like in GET, NEXT, PREVIOUS, etc.)
  - CH (Change)

- ME (User message). A message presented to the user (from NeatMessage).

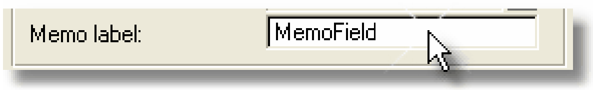
Beware!! Only the first letter is used by some internal processes, therefore if you translate this default values take care that the first letter of each option is always unique. Also, keep the length of these equates consistent with the length of the audit file "operation" field, or some comparisons may fail and the template might work incorrectly.


Search key: AUD:SearchKey
Index Search key: AUD:bySearchKey

- Search Key: Here you will store, in a STRING format, significant data that might be examined in a query or in the inspector program. Keeping in mind that depending on the data type the audit process is storing the records in their binary form, if you use fields (Like LONG or REAL) the data will be unreadable. Furthermore, if you compress the data, it will be completely unreadable. If you want to do a query or a search with SQL, this "search key" field will let you see some information from the record.
  Index Search key: identifies the key in the Audit file that indexes the file by the Search Key field mentioned above.

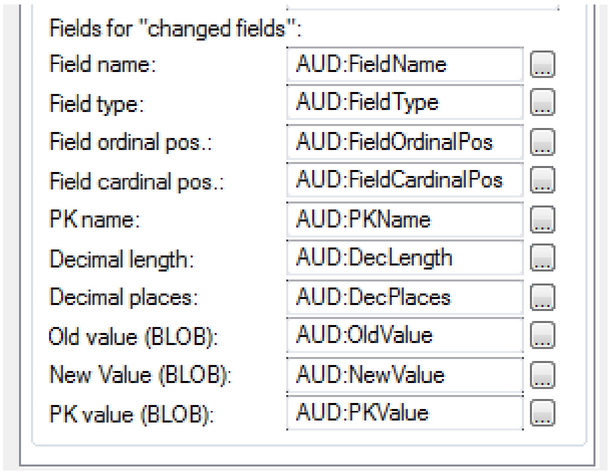

Stored Record: AUD:StoredRecord

- Stored Record: This field stores the audited record. It may be a STRING or a MEMO (binary) field, but it has to be big enough to store the longest record (excluding memos) of all the files you have in your dictionary. You can obtain this value using the TXD Analyzer supplied program. If you need to increase the size of this field, then you have to change this field in the Audit file as well as the other two audit fields (New and Bck). You also need to update the length of the StoredRecord in the Queue_Buffer in the global declaration called FRGlobalData. If you fail to make all of them the same size, you may lose audit data or it will be the cause of unpredictable results.
  - If you choose a Memo type field to store the record, then it has to be defined as BINARY.
  - IF you use a MSSQL file to store the audit data, this field should be of datatype Binary or VarBinary in the SQL definition.
  - If you use a FIREBIRD file to store the audit data (connecting via ODBC), this field should be a BLOB.
  - IMPORTANT: Do not use a CTRING type of field for the stored record, as the stored information might have binary zeroes that will "cut" the information and you will lose audit data.


Memo label: MemoField

Memo label: Is the *label* of the memo fields in the audit file, without prefix and without the final numbers. For example, if you have two memo fields in the audit file, named AUD: MemoField1 and AUD:MemoField2, here you have to type MemoField. The length of each memo field should be as long as the longest of all the memos in all your files. You must change this field in the Audit file as well as the other two audit fields (New and Bck). You also need to update the length of the MEMO in the Queue_Buffer in the global declaration called FRGlobalData. (See the global "memos" field in General Tab). The example dictionary has two memos, if your system has more than two in any file, you should add Memofield3, MemoField4, etc. to the audit files and increase the DIM value in the "MEMO" field in the global data area. See Using TXD Analyzer for obtaining this information automatically.

Fields for "changed fields"
If you select the "Changed fields" or the "Mixed" operation mode for the template, you will need the following fields in your audit file:



Fields for "changed fields":

Field name:	AUD:FieldName
Field type:	AUD:FieldType
Field ordinal pos.:	AUD:FieldOrdinalPos
Field cardinal pos.:	AUD:FieldCardinalPos
PK name:	AUD:PKName
Decimal length:	AUD:DecLength
Decimal places:	AUD:DecPlaces
Old value (BLOB):	AUD:OldValue
New Value (BLOB):	AUD:NewValue
PK value (BLOB):	AUD:PKValue

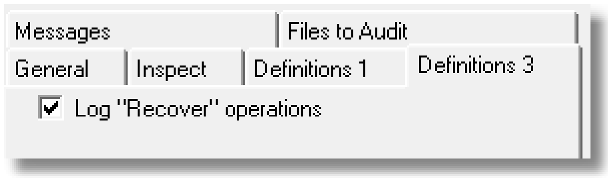
Field name: This one will store the name of the changed field. By default is a STRING(40), you may change it in the DCT if you have larger (or far shorter) field names.
Field Type: The system will store here the type of the field, this information is necessary to retrieve it.
Field ordinal pos: This is the ordinal location of the field inside the whole record structure. It has to be a signed field in case it is a Memo or Blob, as their Ordinals are negative numbers (A SHORT will normally work). This field is used as well to identify if the stored information is a Full Record (0 value) or a Changed Field (Any value not zero).
Field cardinal pos: If the field is part of an array, this would be the element inside the array. For example, if you have a field dimensioned as DIM(3), and you changed the second element, this would contain the value 2. If you have a DIM(3,3), and you change the element (2,1), this field will take the value of 4, as in the record structure the fields are repeated one after the other in order, so you will have the field with the same name as (1,1), (1,2), (1,3), (2,1), etc.
PK name: Is the name of the file primary key that allows to identify the record in unique way - it is used to restore the changed value if you want to recover from the audited data.
Decimal length: The number of numeric places if the stored field is a decimal type field.
Decimal places: The number of decimal places if the stored field is a decimal type field.
Old value (BLOB): A BLOB field to store the "old" value of a changed field.
New Value (BLOB): A BLOB field to store the "new" value of a changed field.
PK value (BLOB): Is the content of the file primary key that allows to retrieve the record using the primary key of the audited file - it is used to restore the changed value if you want to recover from the audited data.
Important: if you are **upgrading** from FullRecord 2.x, you won't have these fields: you have to add them to your audit file and convert it with any way you like (FM3 is ok, for example).
Note: if you will use the "FullRecord" mode and will never use the "mixed" or "change fields" modes, you can remove the extra three BLOB fields from the audit file, however the rest of the fields must be added anyway. In this case you may left the "blob" fields entries here empty.

Review the fields from the AUDIT file and from the global data area (FRGlobalData) in your dictionary, to verify that they fit your needs. You should not modify the audit fields that are not mentioned here (if you find any) or the template may stop working accurately.

8.1.7 Definitions 3 Tab

"Definitions 3" Tab - Basic Settings

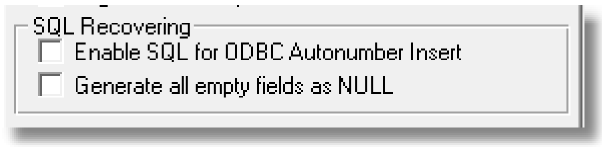
Log "Recover" operations: One of the features of the auditing system is that you may recover (undelete) any single record of any file from the audit file. You may define a secondary audit file to store the information about these recovered records, including which record, who and when was retrieved.



☑ Log "Recover" operations

In the main app, or for a multi-dll system in your data dll app and in the app where you will use the control template (these can be the same app), you should activate this option if you want to use the recover recording system.

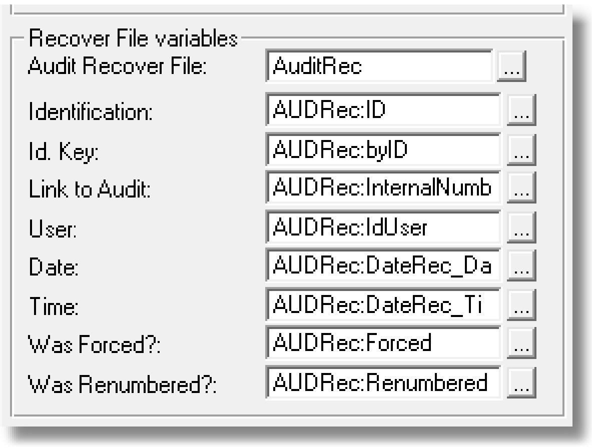
"Definitions 3" Tab - Advanced Settings


SQL Recovering
☐ Enable SQL for ODBC Autonumber Insert
☐ Generate all empty fields as NULL

Enable SQL for ODBC Autonumber Insert: In some SQL systems like MySQL, when you use an autonumber key, the recovered record will always take a new number, unless you recover it using SQL syntax. You may then activate the "Enable SQL for ODBC Autonumber insert" option in this control to change the insert syntax from Clarion code (ADD...) to SQL code (INSERT INTO...), when the file is SQL (Currently ODBC and Pervasive SQL files are supported).

Generate all empty fields as NULL: If your fields are Nullable, you may also check the "Generate empty fields as NULL" option. This will generate a NULL clause instead of '' or 0 to fill the fields. Keep in mind that if you have any no nullable fields, this option may cause an SQL backend error.

Then, you may change the "Recover File variables".



Recover File variables

Audit Recover File:	AuditRec
Identification:	AUDRec:ID
Id. Key:	AUDRec:byID
Link to Audit:	AUDRec:InternalNumb
User:	AUDRec:IdUser
Date:	AUDRec:DateRec_Da
Time:	AUDRec:DateRec_Ti
Was Forced?:	AUDRec:Forced
Was Renumbered?:	AUDRec:Renumbered

- <u>Audit Recover File:</u> First, in "Audit Recover File", you have to specify the name for your Audit recover file. This is the file where the recover operations are going to be logged. In our example DCT is the file AuditRec.

- <u>Identification</u>: Is the field that contains the auto numbering for the information about the recovered audit records. Normally it would be a LONG or ULONG type of field.

- <u>Id. Key</u>: Among the Audit Recover file keys, you have to specify which one is the one that carries the auto numbering field. Normally it would have one single component, the identification field.

- <u>Link to audit</u>: Is the field that contains the link for the audit records, so the recovered information could be show in a browse box. This field has to be identical to the identification field from the audit file.

- <u>User:</u> This field will contain the identification of the user that recovers the deleted record.

- <u>Date:</u> This will be the date when the record was recovered.

- <u>Time:</u> This will contain the time when the record was recovered.

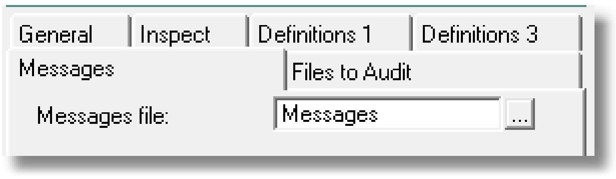
- <u>Was Forced?:</u> When you recover a record, it will be placed in the same position where it was deleted from. If this position is already occupied with a new record, you will have the option to delete the existing record to recover the old one. This case is indicated with a value of TRUE in this field, as "Forced" means that you replaced the existing record by force.

- <u>Was Renumbered?:</u> In the "Was Forced?" case, if you choose not to delete the new record to recover the old one, you still may recover the old record giving a new ID number to it. This case is indicated with a value of TRUE in this field, as "Renumbered" means that you change the original ID to a new valid number. This is only possible when you have an autonumber main key, and the other keys allow duplicated values.

8.1.8 Messages Tab

In this Tab you will specify some options that only work when you have the global extension for NeatMessage incorporated in the same application (Purchased separately).

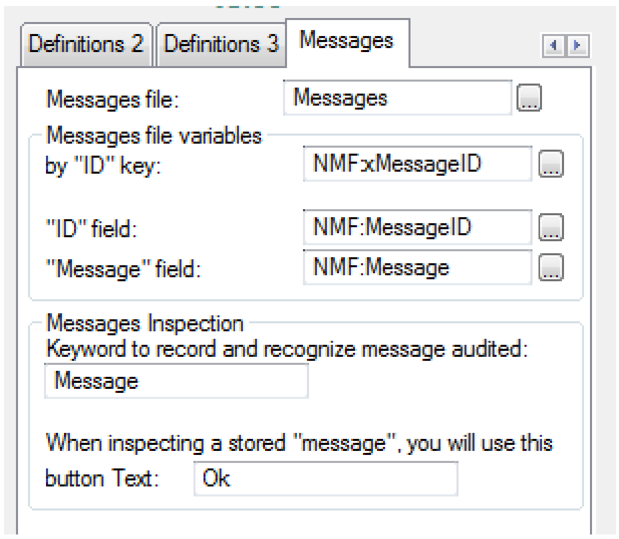
"Messages" Tab - Basic Settings



Messages file: Messages

Messages file: In combination with NeatMessage, FullRecord can record all the messages that are presented to the user, and their response to them. If that is the case, you have to specify the name of the file where the messages are going to be stored ("Messages" by default).

"Messages" Tab - Advanced Settings



Messages file: Messages

Messages file variables

by "ID" key: NMF:xMessageID

"ID" field: NMF:MessageID

"Message" field: NMF:Message

Messages Inspection

Keyword to record and recognize message audited: Message

When inspecting a stored "message", you will use this button Text: Ok

by "ID" key: This is the key of the Messages file that will identify each message by an unique number.

"ID" field: This is the field inside the file structure that will identify each message by a unique number. This should be a LONG.

"Message" field: This is the field to store the actual message.

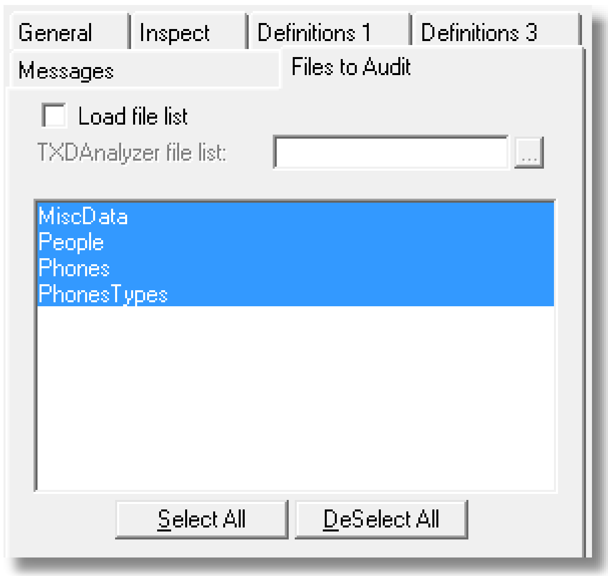
Keyword to record and recognize message audited: The messages are stored in the Audit file using one record by message. As they are not really "file" records, the system needs a keyword to identify that this is not a "record" but a message. Enter the keyword in the entry field, just make sure that this keyword is *not* a real file name.

When inspecting a stored "message", you will use this button Text: From the "Inspect" window, when you press the "view record" button over a "message" type record, a message box will show with the content of the original message. That message box have one single button to continue, the button will take this text.

8.1.9 Files to Audit Tab

In the main app, or in your main-data dll app for a multi-dll system, you have to specify which files you are going to audit.

"Files to Audit" Tab - Basic Settings

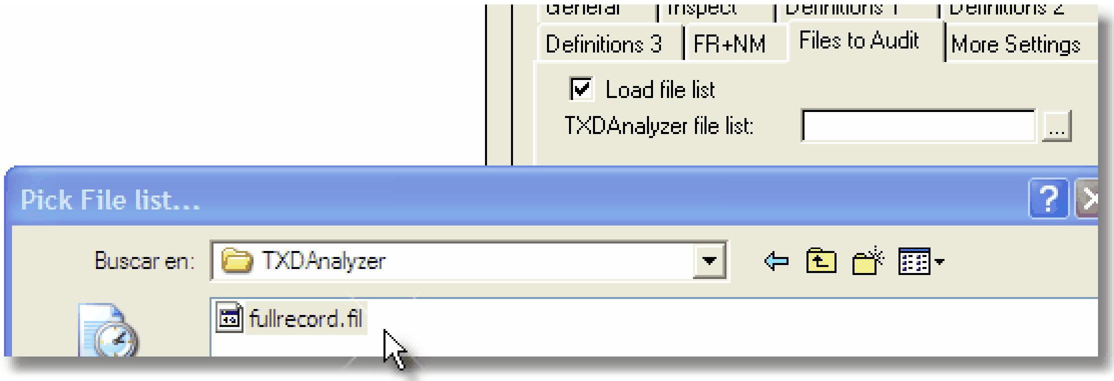


MiscData
People
Phones
PhonesTypes

The fastest is just to press the "Select all" button to select all the files. The Audit file (with Bck, New and Recover secondary files) and the Messages file defined before are automatically excluded.

If you don't use the TXD Analyzer companion program, in a multi-dll system you have to select the files to inspect in this tab in the same way as you did in the main-data dll, i.e. you have to select the files twice; in the main-data dll and in the dll or exe where the inspect procedure will be. If the inspect procedure is in the same main-data dll (not advisable), then you just need to select the files in that place.

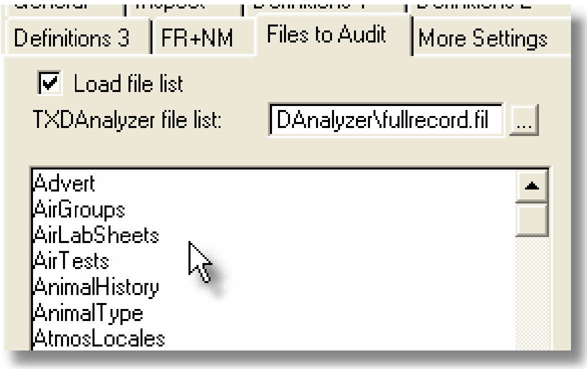
A quick and better alternative way to do this is to use the TXD Analyzer program and Export the Files List. Then you can select here the "Load file list" option, and pick up the file generated by the TXD Analyzer program.



Load file list: ✓

TXDAnalyzer file list:

After you select a FIL file here, your files list will be imported into the global extension of the template. Then you can "Select all" the files you want to audit or inspect.



Load file list: ✓

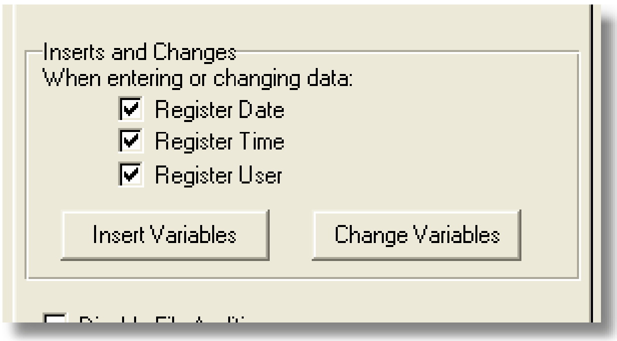
TXDAnalyzer file list: DAnalyzer\fullrecord.fil

Beware!: This procedure is automatic and will save you a lot of time, but be careful, the "manual" way of selecting files will pick files from a list made from your current DCT, while this automatic import of the list is made blindly; i.e. if you pick the wrong FullRecord.FIL file, you will end with a list of files that don't belong to the current app. The remedy is simple, though; just pick the correct FIL file and the list will be corrected automatically.

When you use the "Load file list" method, the file list is imported into your app each time the app is opened or the global extension is accessed. Your actual file selection inside this list is preserved, but the FullRecord.FIL file referenced here (Generated by the Export Files List button in the TXD Analyzer program) have to externally exists in the indicated path, and it will be needed for compiling the program.
Note also that if you make changes to the external file list, like adding files, those files will not be automatically selected in the global extension, so you have to select them entering the global extension in the app needed.

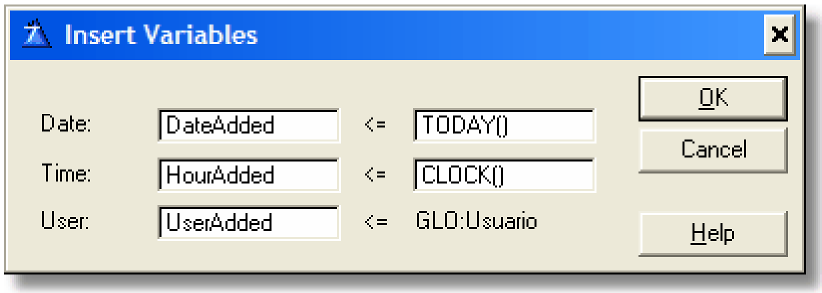
"Files to Audit" Tab - Advanced Settings

Below that, you can see the following prompts:



Inserts and Changes
When entering or changing data:
- [✓] Register Date
- [✓] Register Time
- [✓] Register User

These allow you to record the "insert" and "last change" audit data to the original file record (NOT to the audit file). This way you can quickly check in a regular browse for any file when and who inserted the record, and when and who last touched it.
You may select if want to record the date, the time and/or the user code.
You should add in your files 2 fields for each checkbox, for each option one for "inserts" and the other for "changes" (if you mark all the options that would be 6 fields). You can name them whatever you want and put them in your files in any position and order you like.
When you press the "Insert variables" button you'll see:

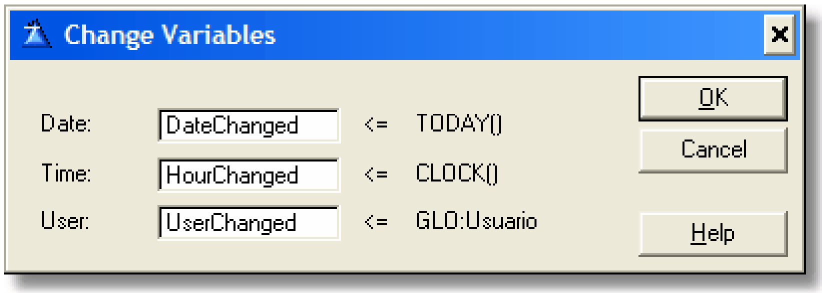


Insert Variables

Date:	DateAdded	<=	TODAY()
Time:	HourAdded	<=	CLOCK()
User:	UserAdded	<=	GLO:Usuario

Here you specify the name of the fields for recording the date, time and user (without the prefix), when the user inserts a new record to any of the audited files. To the right, you specify the value that fields should take (The user variable was already specified above). As for this example, if you have a PEOPLE file with PEO as prefix, the template will look at three variables named PEO:DateAdded, PEO:HourAdded and PEO:UserAdded (case insensitive, in any order and position within the PEOPLE file definition) and if it found any of them, it'll fill them when appropriate. If the template is instructed to fill these variables, but any of them are not found in the file definition, the unfound variable is just ignored.

When you press the "Change variables" button you'll see:



Change Variables

Date:	DateChanged	<=	TODAY()
Time:	HourChanged	<=	CLOCK()
User:	UserChanged	<=	GLO:Usuario

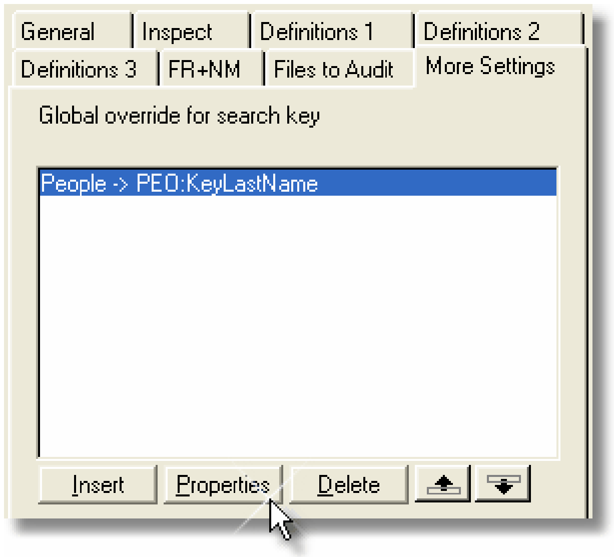
Here you specify the name of the fields for recording the date, time and user (without the prefix), when the user changes an existing record on any of the audited files. To the right, you can see the values that the fields are going to take. These are shared with the insert variables.

8.1.10 More Settings Tab

"More Settings" Tab - Advanced Settings

This whole Tab is part of advanced settings and won't be visible if the basic interface is set.

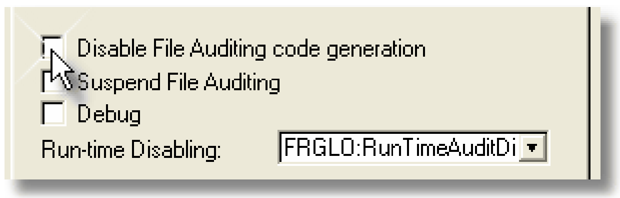
The "search key" field (AUD:SearchKey by default) generation inside the audit file it is made automatically by the template, which takes the first key of the file.
If the information of this particular key were of no significant for you, you might want to replace this key by another one. Within this Tab, you choose the file for which you want to override the key selection (press insert), and then select the key for what you want the fields to be taken from. The order and pictures will be taken from the data dictionary.



Global override for search key

People -> PEO:KeyLastName

If you don't want the search key to be generated either with the primary key nor other key indicated here, you can still specify a manual criteria using embed code. Entering the global properties of the app, via the "Embeds" button, look for "FullRecord: After assign search key", and you will see a tree with a branch for each file. Entering here in the branch for the wanted file, simply write
JAL:CLAVE = ... (expression)
Where "..." is the expression you want to make, and the search key generation will be replaced by your code.
To *add* your expression to the existing code, try
JAL:CLAVE = CLIP(JAL:CLAVE) & ... (expression)
If you want that the original routine code were not even generated, you must surrounded it with an OMIT group, using the mentioned embed and the previous one "FullRecord: Before assign search key". It is very useful to see the generated source code when doing this.



Run-time Disabling: FRGLO:RunTimeAuditDi

Disable File Auditing code generation: This option allows you to completely skip all code generation, disabling in fact the template, while maintaining all your extension settings. This might be handy for testing purposes. Although, as no code will be generated, if you have any "manual" call, it will issue a compiler error as it won't be recognized any longer. If you have manual calls and want to retain them while disabling the template, use the next option instead.

Suspend File Auditing: This option will generate all the template code as usual, but none will be executed. The final effect will be as if the template wasn't there, but if you have " manual" calls to the template, they won't issue any compiler error while they won't work any more (they will simply do nothing).

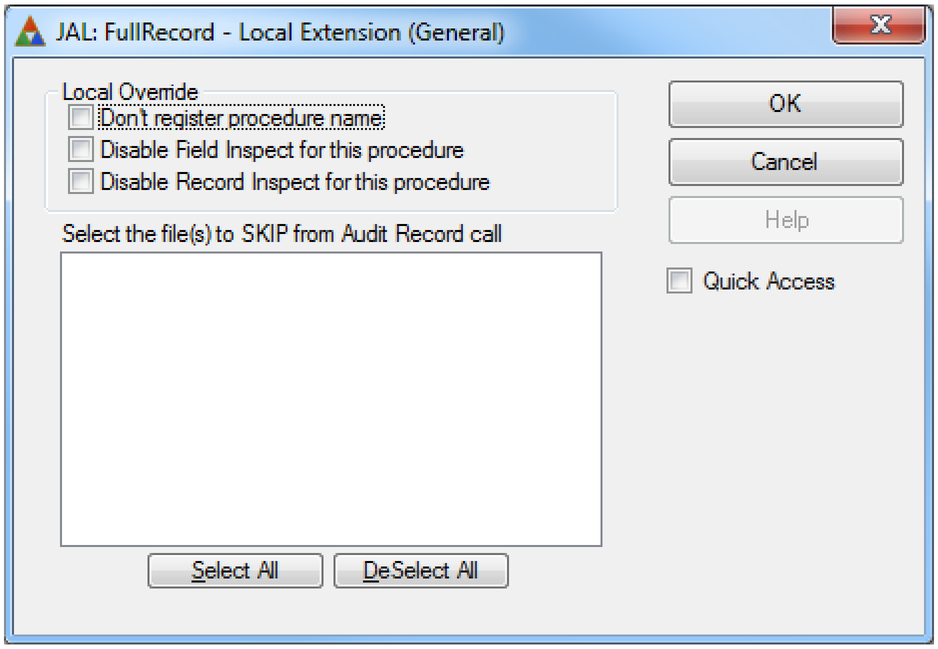
Run-time Disabling: You may choose to disable the auditing in run-time. Use the "Run-time Disabling" field to specify which variable will receive the value. It should be at least a BYTE (boolean) variable as it will have a value of TRUE or FALSE. If this variable is TRUE, then the auditing won't take place. You may activate and deactivate this value at run-time on a need-to basis, if there is some specific procedure where you don't want to audit the files.

Debug: The "debug" option should be always unchecked unless requested by Laro Group support. This option may generate a log file which will make some operations work slower.

8.2 Local Extension

When you add the Global Extension in your program, a Local Extension is automatically added for every procedure.
Note: due to a Clarion problem, if you are going to export a procedure to import it from another program, you must first remove the global extension, else the local extension will be duplicated in the target procedure, and will produce compilation errors.

If you go to the local Extensions, you will see the JAL: FullRecord - Local Extension (General). Go to its properties and you will see this window:


JAL: FullRecord - Local Extension (General)
Local Override
Don't register procedure name
Disable Field Inspect for this procedure
Disable Record Inspect for this procedure
Select the file(s) to SKIP from Audit Record call
Select All
DeSelect All
OK
Cancel
Help
Quick Access

Don't register procedure name: A couple of routines are included in each procedure to handle a queue that stores the procedure name, so it can be saved to the audit file. These are **JAL_InitProcName** and **JAL_KillProcName**. They are called automatically by the template code, except in source type procedures (See Manual Calls). If you tick this option, the routines won't be generated, the call to them will be skipped and no procedure name information will be saved (unless you manually handle it).

Disable Field Inspect for this procedure: If you tick this checkbox, you will disable the automatic calls to the Field Inspection window made from the fields of the window in this procedure.

Disable Record Inspect for this procedure: If you tick this checkbox, you will disable the automatic call to the Record Inspection Window made from this procedure. This will disable all the calls for the Record Inspection Window.

Select the file(s) to SKIP from Audit Record call: When you have several main files in the Browse or Form, the Record Inspect window will be called for each one of them in turn (when you press the hot key). If you want to skip any (or all) of those files, select them in this local list.

Note: This will only skip the calls to the inspect window, none of these options will skip the files from being audited.

8.3 Manual Calls

FullRecord audits every change to your files no matter if you are using legacy code, therefore manual calls are never necessary.
You may however save custom information to the Audit file, like warning messages or local variables.

8.3.1 Manual save of custom information

You can manually add information to your Audit file, using the SaveAuditMessage procedure.

```
SaveAuditMessage(Message)
```

- "Message" is a string with whatever you want to register.

For example, suppose you need to save an audit entry when your user press a button. In the event:accepted for the button, add the following:

```
SaveAuditMessage('The forbidden Button was pressed')
```

If you want to audit some local or global variables, you can simply add them to the saved string:

```
SaveAuditMessage('Warning: Amount approved: '&FORMAT(LOC:AMOUNT,@N_12.2))
```

You will see the message with all the related information (User, date, time, calling procedure, etc.) in the Audit Browse. Also, when you press the "View record" button in the Audit browse you will see a Message containing the full message you saved.

See also: The AboutWindow (Manual Event audit) example.

8.3.2 Calling procedure name generator from Source template

Some code needs to be executed on each procedure so FullRecord knows the name of the procedure. On "source" type procedures the template generator cannot "execute" the code as there is only one embed for code execution and it's yours. So if you are going to audit files in a source procedure, you may want to call the "procedure name" arm and disarm routines, as follows:

```
DO JAL_InitProcName  !Arm the procedure name generator
!Your code
DO JAL_KillProcName  !Disarm the procedure name generator
```

For example, the following code is valid and audited in a source procedure:

```
DO OpenFiles
DO JAL_InitProcName
Clear(PEO:Record)
PEO:FirstName = 'Jorge'
PEO:LastName = 'Lavera'
If Access:People.PrimeRecord(1)
Else
  PEO:CivilState = 'M'
  PEO:Gender = 'M'
  If Access:People.Insert()
     Access:People.CancelAutoInc()
  End
End
DO JAL_KillProcName
DO CloseFiles
```

Note: If you forget to add these "arm" and "disarm" calls, the auditing will occur anyway, but the procedure name will be wrong in the audit file, for the operations made in this source procedure.

You can avoid the code generation of these routines using the checkbox in the local extension.

8.4 Control Templates

8.4.1 View Record (ConsultaAudit)

You may populate this Control Templates on a Browse over the audit file. It adds all the code needed to inspect all the fields of the file you are selecting in the Browse. When you press the button at run-time, you will see the window described at the inspection window.

When you populate this button in the window, you have to select the "Tables" button to enter the file Schematic and select a primary Audit file for the control. At run-time, when you press the button, you will see the information stored in that file (Which may be the same of the Browse, or not; in case you need to combine or replace audit files). You need to activate the global generation option mentioned in the Inspect Tab chapter.

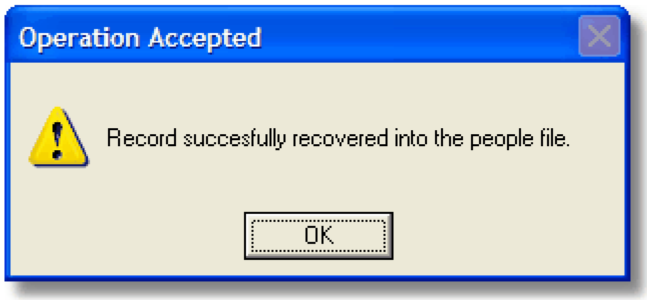
When you select a modify operation (stored as "CH" by default) you will see highlighted in red the fields that changed from the previous reading to the saving, and an additional column to the right showing the previous (original) value.

A special case are the operations recorded as "ME" (user message by default – only available when using NeatMessage). When you choose one of that, you will see the message that the user got.

8.4.2 Recover (RecuperarAudit)

You may populate this Control Templates on a Browse over the audit file. It adds all the code needed to recover the full record of the file you are selecting in the Browse. The recovering doesn't need to be from a delete operation.

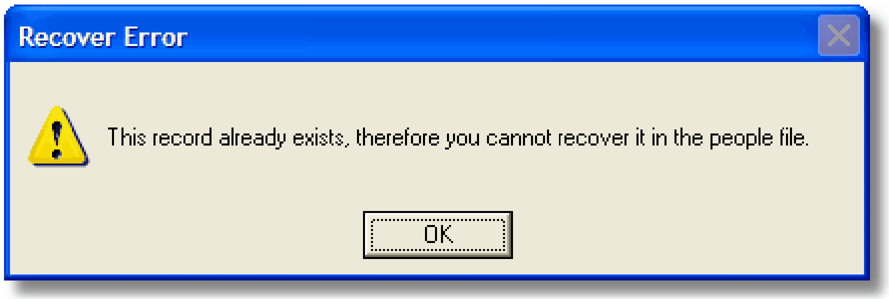
When you select a proper candidate, and press the recover button, you will see the following message (with the correct file name) if all goes well:


Operation Accepted

Record succesfully recovered into the people file.

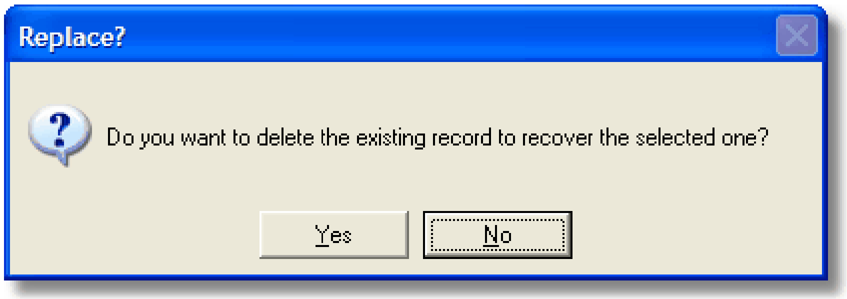
If you select to recover a record that already exists in the original file, you have several options.

(a) First of all, the template issues a warning:


Recover Error

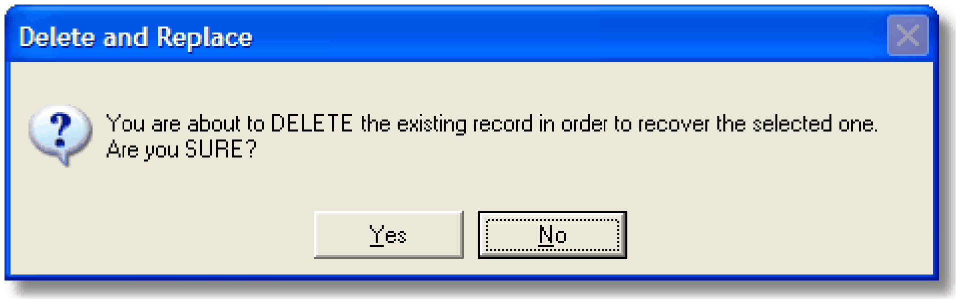
This record already exists, therefore you cannot recover it in the people file.

After you press OK, you can go forward. The first question you have to answer is if you want to overwrite the existing record:


Replace?

Do you want to delete the existing record to recover the selected one?

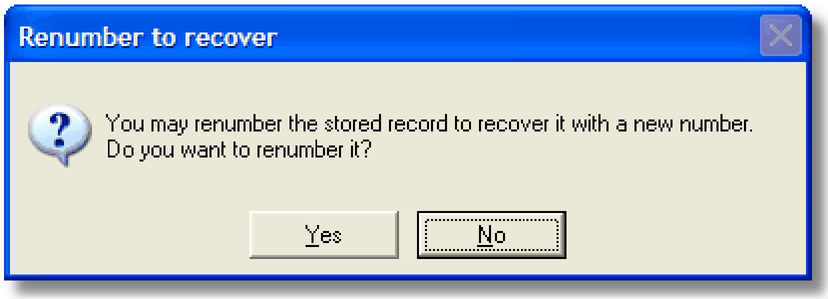
If you answer No, you have the option (b):
If you answer Yes, you have to confirm that you want to delete the current data in favour of the old one:


Delete and Replace

You are about to DELETE the existing record in order to recover the selected one.
Are you SURE?

If you answer Yes again, you will replace the current record with the audited record.
If you answer No, you exit to the Audit browse window.

(b) If you answered No to the "delete the current record" option, you will see this question:

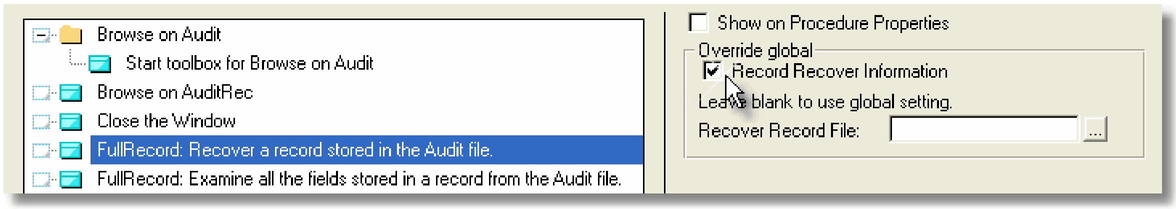

Renumber to recover

You may renumber the stored record to recover it with a new number.
Do you want to renumber it?

If you answer No, you exit to the Audit browse window.
If you answer Yes, the current record is preserved, and the audited record is recovered to the file with a new primary key number (equal to the last available number plus one).
Note: This will work if there is a primary key, and if the recover doesn't creates any duplicate key on the secondary keys. If the primary key is a compound key, the field renumbered will be the last field of the key.

* * *

The correct recover of the record depends that the Primary key be unique. If not, the recovered record will not be in the same order as originally was. Additionally, when you use an autonumber key, if you delete the last added record and then insert another one, this last one takes the number of the previously deleted one, therefore you can only "recover" the erased one deleting the new one or renumbering it. If you need to maintain both and the old one with the original number, a trick to do this is that in your system, you change the key field in the last record added so you make a "hole" in the numbering sequence. The hole should be coincident with the deleted record. Then, you can recover it safely.


Browse on Audit
Start toolbox for Browse on Audit
Browse on AuditRec
Close the Window
FullRecord: Recover a record stored in the Audit file.
FullRecord: Examine all the fields stored in a record from the Audit file.

☐ Show on Procedure Properties
Override global
☑ Record Recover Information
Leave blank to use global setting.
Recover Record File:

If you are not going to record the track of recovered information from the audit file, you should disable the "Record Recover Information" at the local extension.
If you use a different "recover record file" than the global one, you may specify it here.

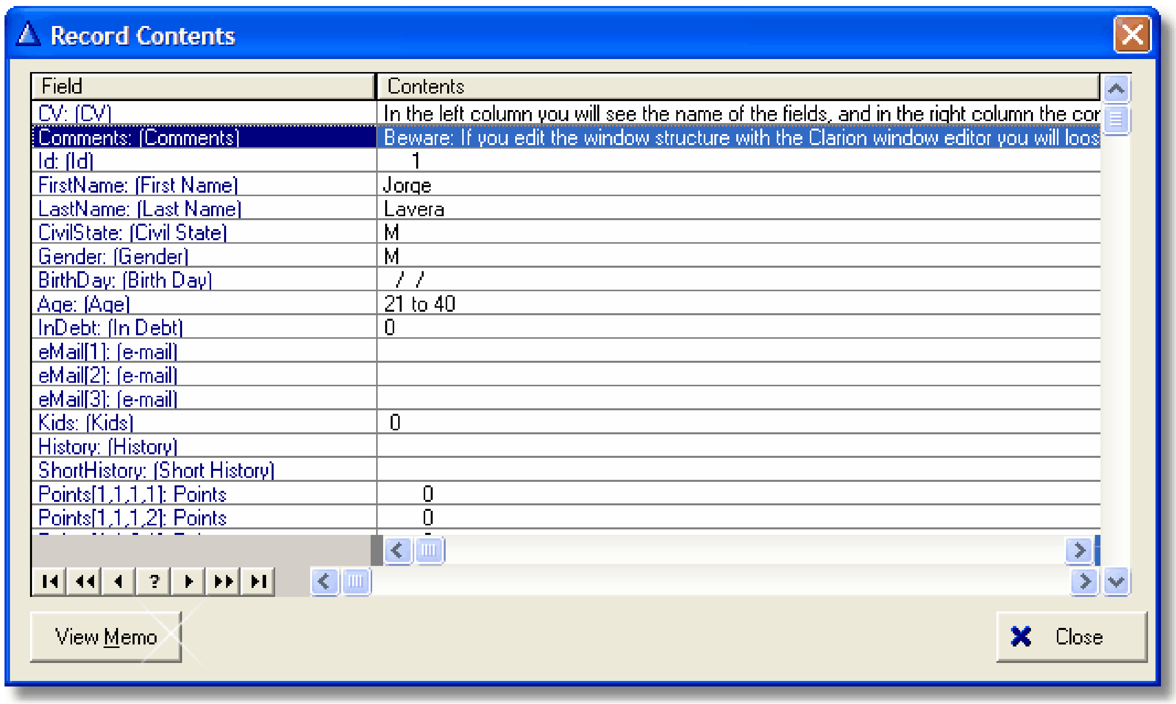
In some SQL systems like MySQL, when you use an autonumber key, the recovered record will always take a new number, unless you recover it using SQL syntax.
You may then activate the "Enable SQL for ODBC Autonumber insert" global option in this control to change the insert syntax from Clarion code (ADD...) to SQL code (INSERT INTO...), when the file is SQL (Currently ODBC and Pervasive SQL files are supported).

If your fields are Nullable, you may also check the "Generate empty fields as NULL" global option. This will generate a NULL clause instead of '' or 0 to fill the fields. Keep in mind that if you have any no nullable fields, this option will cause an SQL backend error.

8.5 The inspection window

This is a template generated window. However, it can be completely customized via external options and source TRN files.
There are two versions of this window. Both versions are included in the "FRWindow. TRN" file (by default) so you can easily customize them (See Inspect Tab).

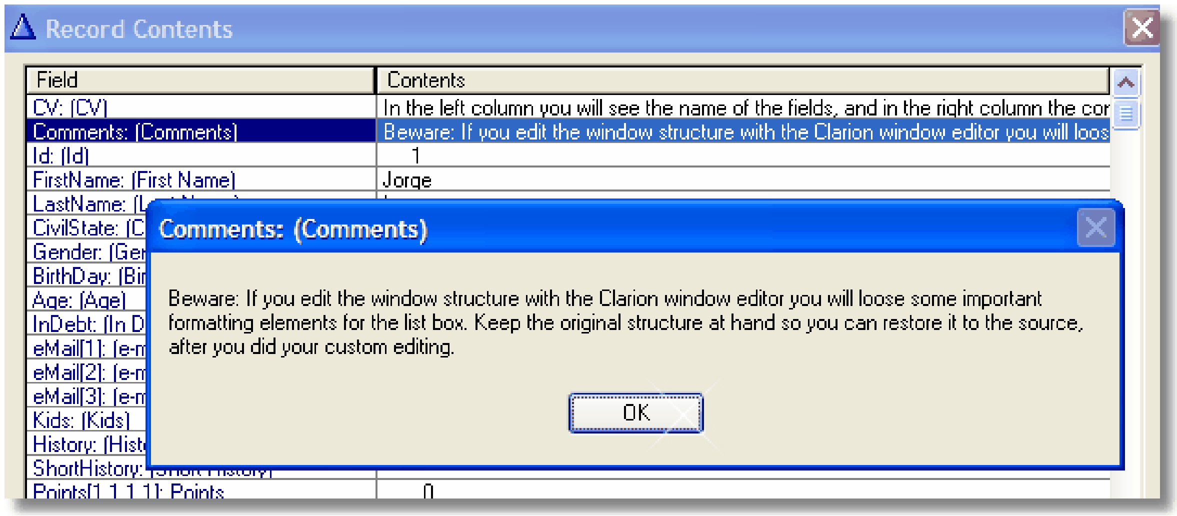
When you press the "View Record" button over a read, add or delete option (LW, AW or BW by default) you will see the window below:



Field	Contents
CV: [CV]	In the left column you will see the name of the fields, and in the right column the cor
Comments: [Comments]	Beware: If you edit the window structure with the Clarion window editor you will loos
Id: [Id]	1
FirstName: [First Name]	Jorge
LastName: [Last Name]	Lavera
CivilState: [Civil State]	M
Gender: [Gender]	M
BirthDay: [Birth Day]	/ /
Age: [Age]	21 to 40
InDebt: [In Debt]	0
eMail[1]: [e-mail]	
eMail[2]: [e-mail]	
eMail[3]: [e-mail]	
Kids: [Kids]	0
History: [History]	
ShortHistory: [Short History]	
Points[1,1,1,1]: Points	0
Points[1,1,1,2]: Points	0

In the left column you will see the name of the fields, and in the right column the content of each field at the moment of the highlighted operation. You can change the title of the columns and window, and the text of the button (See Translation options) and the first column contents and the wallpaper and button's icon (See Inspect Tab).

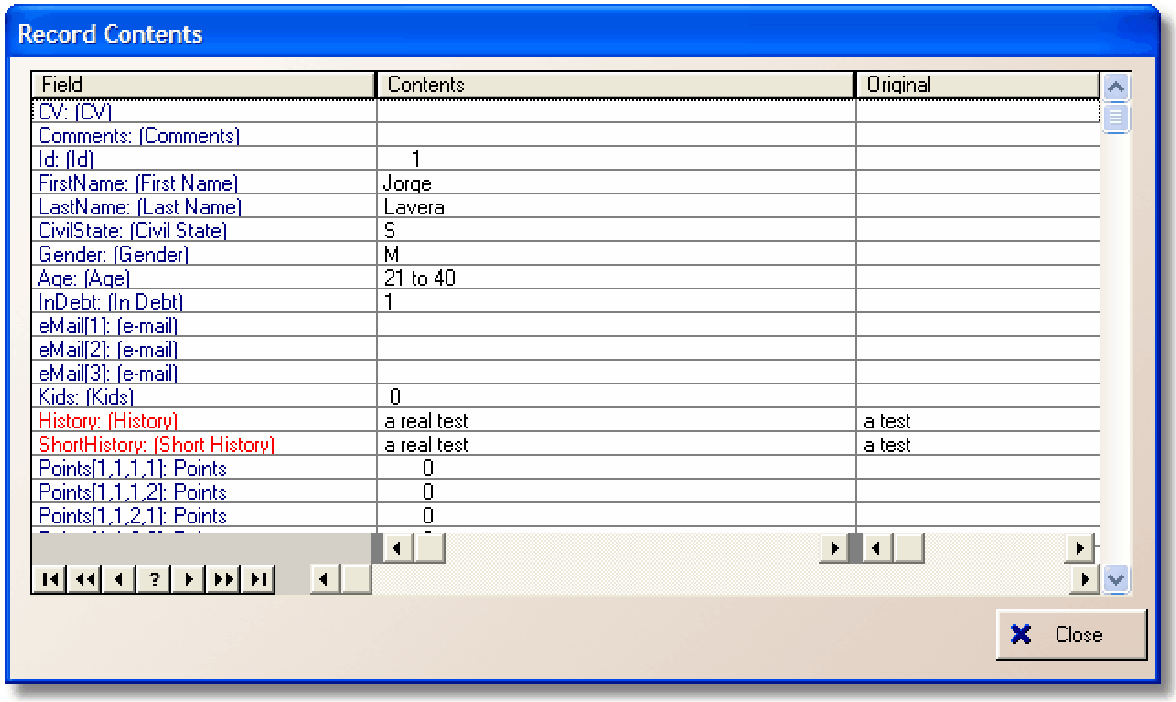
The "View Memo" button will be active only when you select a memo field in the list of fields. When you press the button you will see the full contents of the memo field:



Field	Contents
CV: (CV)	In the left column you will see the name of the fields, and in the right column the cor
Comments: (Comments)	Beware: If you edit the window structure with the Clarion window editor you will loos
Id: (Id)	1
FirstName: (First Name)	Jorge
LastName: (L	
CivilState: (C	
Gender: (Ger	
BirthDay: (Bir	
Age: (Age)	
InDebt: (In D	
eMail[1]: (e-m	
eMail[2]: (e-m	
eMail[3]: (e-m	
Kids: (Kids)	
History: (Hist	
ShortHistory:	
Points[1,1,1,1]: Points	0

Beware: If you edit the window structure with the Clarion window editor you will loose some important formatting elements for the list box. Keep the original structure at hand so you can restore it to the source, after you did your custom editing.

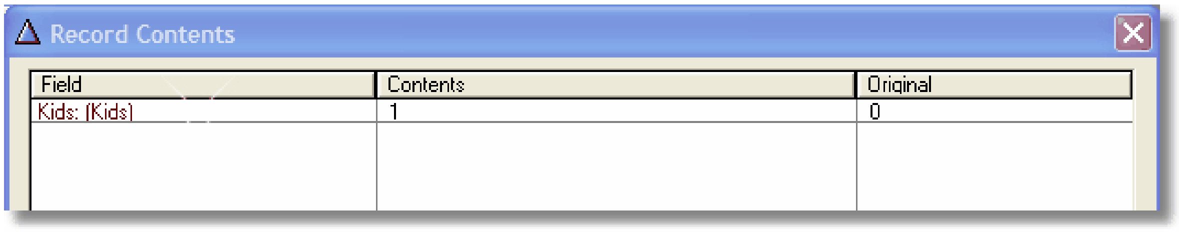
When you press the "View Record" button over a modification option (CH by default) you will see the window below:



Field	Contents	Original
CV: (CV)		
Comments: (Comments)		
Id: (Id)	1	
FirstName: (First Name)	Jorge	
LastName: (Last Name)	Lavera	
CivilState: (Civil State)	S	
Gender: (Gender)	M	
Age: (Age)	21 to 40	
InDebt: (In Debt)	1	
eMail[1]: (e-mail)		
eMail[2]: (e-mail)		
eMail[3]: (e-mail)		
Kids: (Kids)	0	
History: (History)	a real test	a test
ShortHistory: (Short History)	a real test	a test
Points[1,1,1,1]: Points	0	
Points[1,1,1,2]: Points	0	
Points[1,1,2,1]: Points	0	

In this window, you will see three columns. In the left column you will see the name of the fields, in the middle column the final content of each field at the moment of the highlighted operation (after the modification) and in the third column the original content of each field before the modification, only when the value is different.
When you actually have a field modification, you will see the changed field name in red in the first column.
Note that if you don't have an original value the third column will remain empty, and in that case only the red color will indicate that a change occurred.

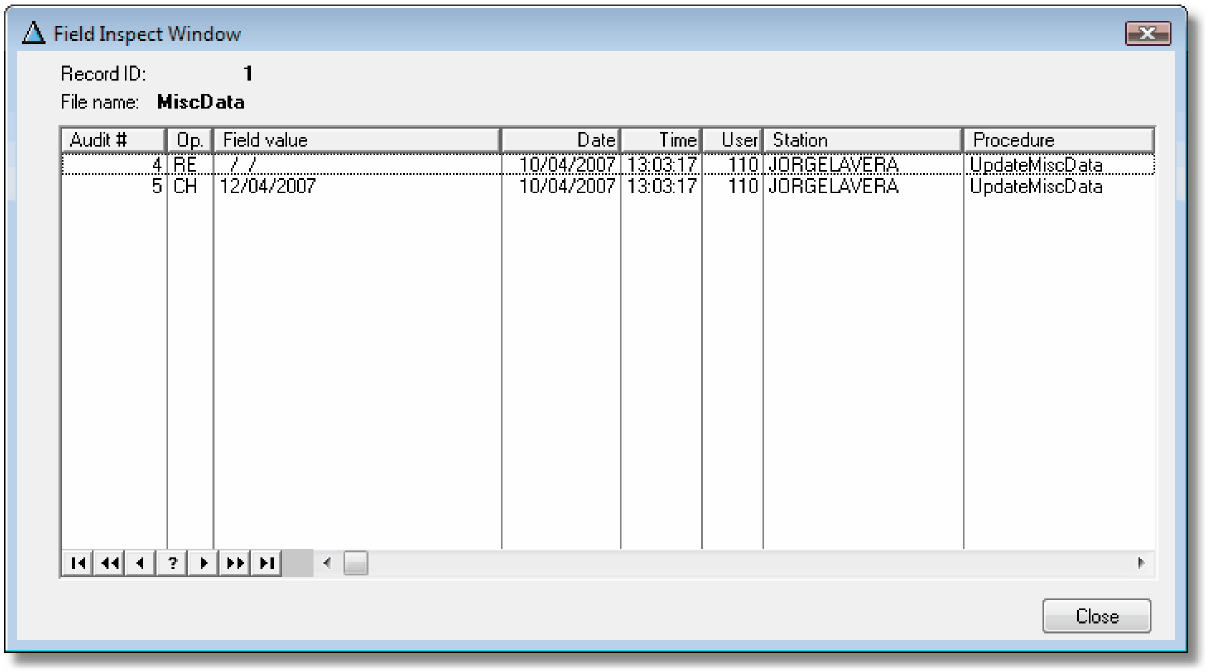
If you choose to "change changed data only" in the Inspect Tab, you will see only the changed fields in the window, as follows:


Record Contents

Field	Contents	Original
Kids: (Kids)	1	0

8.6 The "field" inpection window

This window will allow you to inspect the changes to a specific field of a specific record in a Form. From a form, you select an entry field and hit the Hot Key (By default Ctrl-Shift-A) and this window will be called. You can disable the call to this window using the Local Extension.
This is a regular procedure (template driven) that you import into your app (or data app, in a multi-dll system).


Field Inspect Window

Record ID: 1
File name: MiscData

Audit #	Op.	Field value	Date	Time	User	Station	Procedure
4	RE	/ /	10/04/2007	13:03:17	110	JORGELAVERA	UpdateMiscData
5	CH	12/04/2007	10/04/2007	13:03:17	110	JORGELAVERA	UpdateMiscData

To import it into your app, use the "File" menu, "Import Text" option, and pick the file BrowseAuditFieldA55.txa for ABC, or
BrowseAuditFieldL55.txa for Legacy (For any Clarion version).

This will insert the BrowseAuditField procedure into your app. This is all you need to do.

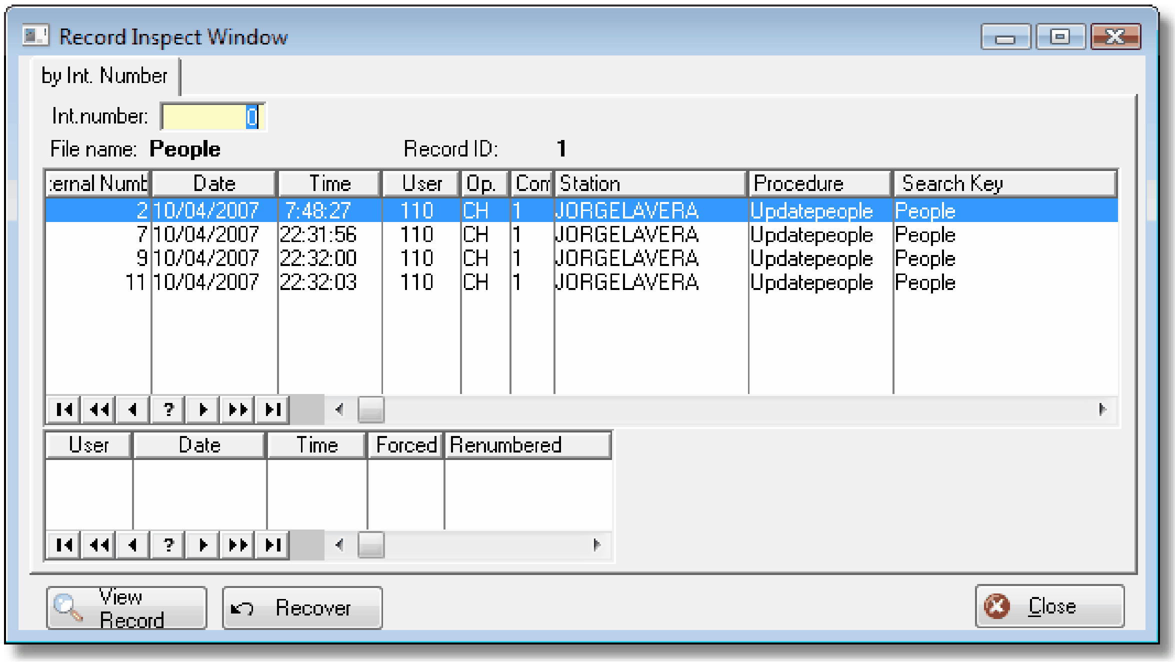
If you want to customize the window to your needs, you may do so.
If you want to change the order of the fields being displayed, you may do it changing the order of the queue FieldQueue in the Data button. You should manually change the Titles

inside the listbox as well.

You cannot simply remove fields from the listbox; if you want to do so, you need to change their order inside the FieldQueue queue past the end of the last visible field. You cannot delete fields from the Queue, nor change its field names.

8.7 The "record" inspection window

This window allow you to see the records audited related with the file in use in a specific Form or Browse.
This procedure is called from any Browse or Form if you activate the global Hotkey (by default Ctrl-Shift-R). You can disable the call using the Local Extension of the caller procedure.
The "record" inspection window is a regular procedure (template driven) that you import into your app (or data app, in a multi-dll system).



Int.number: 0

File name: **People** Record ID: **1**

ernal Numb	Date	Time	User	Op.	Com	Station	Procedure	Search Key
2	10/04/2007	7:48:27	110	CH	1	JORGELAVERA	Updatepeople	People
7	10/04/2007	22:31:56	110	CH	1	JORGELAVERA	Updatepeople	People
9	10/04/2007	22:32:00	110	CH	1	JORGELAVERA	Updatepeople	People
11	10/04/2007	22:32:03	110	CH	1	JORGELAVERA	Updatepeople	People

User	Date	Time	Forced	Renumbered

To import it into your app, use the "File" menu, "Import Text" option, and pick the file BrowseAuditRecordA55.txa for ABC, or
BrowseAuditRecordL55.txa for Legacy (For any Clarion version).

This will insert the BrowseAuditRecord procedure into your app. This is all you need to do.

If you want to customize the window to your needs, you may do so. It is a regular browse, so you can add, delete or move its columns as needed.

8.8 Extension Templates

8.8.1 Regenerate Search Key (JALExtGenerarClave)

You may add this extension to a “Process” type procedure over an audit file, so you can process its records and modify these, so its search key matches the current criteria.

See the “ProcessAudit” procedure in the example program.
Suppose that you determine that certain search key is not useful for your queries, and you decided to change it at the global level (either by key or by embedded code). This will be applied to the new records for that moment on, but the “old” records inside the audit file will be with the old search key specification.
Running this process, the search key is changed to match the current design.
If the global "compress" option is activated, the changed record will be compressed upon saving to the file, no matter its initial state.
The regeneration process will be applied over the primary file specified on the “Process” template (Not necessarily the audit file mentioned in the global extension). This allows you to specify regeneration processes to run over backup audit files.
This process doesn't requiere exclusive access to the audit file, can be run with your system in use and in a network.

8.8.2 Auditing Cleanup (JALExtLimpiarAudit)

You add this extension to a “Process” type procedure over the audit file, so you can process its records and generate two new files from the current data. See the “CleanAudit” procedure in the example program.
You must ask for a “cut date” in the process window, and start the process “paused”.
In the File Schematic, you must add the two auxiliary audit files mentioned in the global extension, identical to the main one. One will be the “new” one, and the other the “backup” one.
All the records from the beginning of the audit file to the cut date (excluded) will be copied to the “backup” audit file, while the records from the cut date on, will be copied to the “new” audit file. When the process finishes it erases the current audit file, and replace it with the “new” one, so you end up with a fresh packed current audit file and a fresh packed one containing old audit data (That can be inspected as well).
The whole copy process occurs inside a transaction.

This process cannot be executed while the system is in use, as exclusive access is required to access the audit files. You have to do it when the system is off-line after hours. Also, keep in mind that depending on the amount of records, it may take a long time to complete the process.

8.8.3 Local Extension for the Process (JALExtFRProc) (ABC only)

This extension is inserted in a “Process” type procedure on any file, so the record read inside the process will be read into the audit buffer. Normally when you use any reading ABC statement like NEXT, PREVIOUS, etc. the buffer is automatically saved so later the system can save the record read before the modified one, but the “Process” template from the ABC chain does not seem to be using any ABC statement to read the records. Therefore, the read must be manually registered. Adding this extension you do not need to write the manual call to fill the buffer, it will be done automatically.

8.8.4 Changing record layout

Make a backup copy of the audit file BEFORE you run this process. You may ruin your audit data if made incorrectly.

Even in the most stable systems, from time to time you change a table design. As FullRecord stores the full record in the audit file, and relies in the current record definition for inspecting its fields, its important to change the stored record to keep it current with the actual definition. Otherwise, the old audit data may become inaccesible (you will see garbled or wrong data).

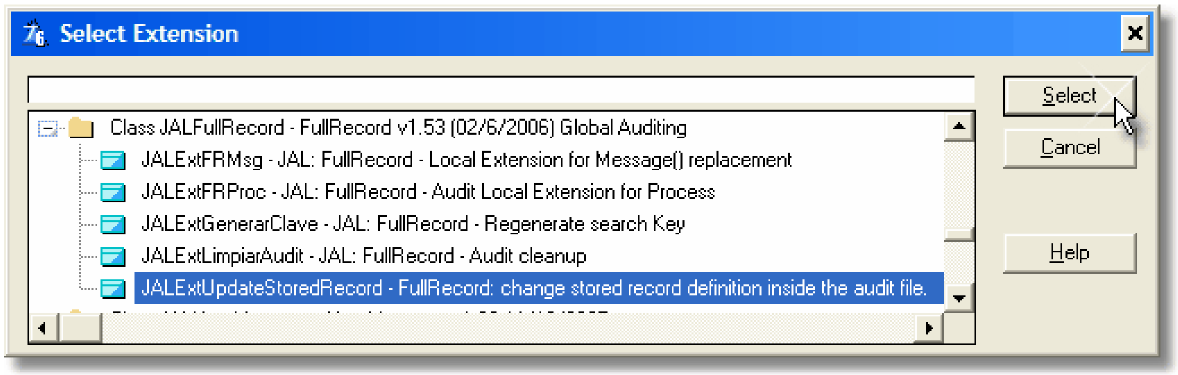
There is one special case; if you just add a field (or more) to the end of the record structure, you don't need to touch the audit file.

However, if you change a record, normally you shall update the stored record in the audit file.
There are many ways of doing this, I prefer one of them as it involves several operational advantages.

First, you need to keep a separate dictionary with a copy of the audit files definitions, and each time you change a record you stores a copy of the old record definition and a copy of the current record definition in that dictionary. I suggest you to append a "date" suffix to each definition (like People_051230 to identify the copy valid until 30th of December of 2005), and number each prefix sequentialy (like Peo1, Peo2, etc.).
Note that if you change the same file several times, the "current" record for the last time becomes the "old" record for the next one.
You might do this in the "work" dictionary, but as you keep adding definitions each time you change a record definition, you fill the dictionary with clutter and I strongly recommend to use a separate dictionary for that matter. Doing this in the working DCT may lead to problems like the "too many segdef" error.

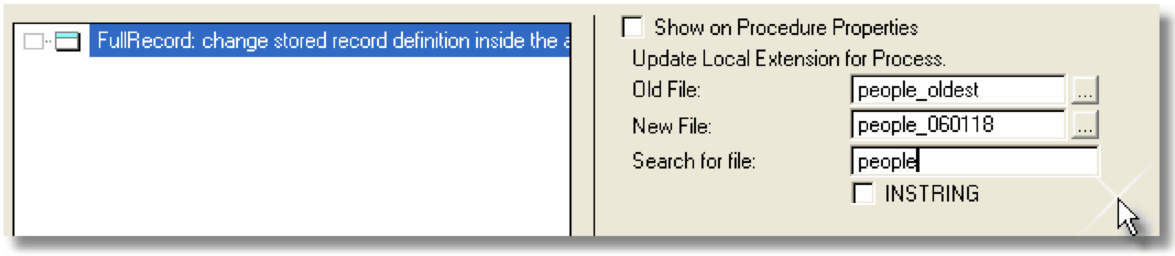
You just make a "process" template on the audit file to change (you might need to make a process for the current audit file and a copy of it for the auditbck historic file).
In the following example, I assume the same files and field names that I shipped in the example. Change them accordingly to your own.

You need to add the JALExtUpdateStoredRecord extension template, as follows:



Class JALFullRecord - FullRecord v1.53 (02/6/2006) Global Auditing
JALExtFRMsg - JAL: FullRecord - Local Extension for Message() replacement
JALExtFRProc - JAL: FullRecord - Audit Local Extension for Process
JALExtGenerarClave - JAL: FullRecord - Regenerate search Key
JALExtLimpiarAudit - JAL: FullRecord - Audit cleanup
JALExtUpdateStoredRecord - FullRecord: change stored record definition inside the audit file.

And you need to fill the prompts for the extension template as shown below:


FullRecord: change stored record definition inside the a

Show on Procedure Properties
Update Local Extension for Process.
Old File: people_oldest
New File: people_060118
Search for file: people
INSTRING

"Old File" is the previous file definition, currently stored in the audit file.
"New File" is the current file definition.
In "Search for file" you have to name the file which record changed, as it is named in the Audit file. This entry will generate a filter embed in the "Validate Record" section, before parent call, priority 2500. This entry is case sensitive.

If you don't mark the "INSTRING" checkbox, a line like this will be generated:
```
IF AUD:DEFILE <> 'people' THEN ReturnValue = Record:Filtered; RETURN ReturnValue.
```

If you do mark the "INSTRING" checkbox, a line like this will be generated:
```
IF INSTRING('people',AUD:DEFILE,1,1) = 0 THEN ReturnValue = Record:Filtered; RETURN
ReturnValue.
```

The latest allows you to select those records in the Audit file that include the word you typed in the "Search for file" entry.

That is all. PLEASE, make a backup copy of the audit file BEFORE you run this process. The process will check the buffers to verify if the procedure run before, therefore it won't ruin the stored record if you run it more than once, but if you type the wrong file in the "Search for file" entry field, you may be altering other file definition than the correct one. It's a good idea to inspect one old entry for the file in the audit file, before and after running the process, to check that all went ok.

You don't need to make the conversion right away after the audit file layout is changed, however, while you don't change the stored record layout, the data provided by the audit file (up to that moment) could be incomplete or unreadable. Nevertheless, the new stored data will be ok.

8.9 RUN-TIME disabling or enabling files to audit

You can now change the audited files in run-time.
You had the option to activate or deactivate the auditing at run-time setting the global variable specified in the More Settings Tab. Now you can also deactivate or reactivate the auditing for specific files changing the value of the fields within the RunTimeGrp group. The fields has the name of the files. For example, if you want to temporary disable the auditing for the Client file, do
RunTimeGrp.Client = False
to disable the auditing over the Client file, then
RunTimeGrp.Client = True
to enable it again.

8.10 Translation options

All the messages, buttons and titles under the template control that the user will see, are stored by default in a text file called FullRecord.TRN in the %ClarionRoot\LIBSRC folder.

The inspection window itself is stored into the FRWindow.TRN file in the same folder.

Beware: If you edit the window structure with the Clarion window editor you will loose some important formatting elements for the list box. Keep the original structure at hand so you can restore it to the source, after you did your custom editing.

In the %ClarionRoot\3RDPARTY\EXAMPLES\FULLRECORD 2 folder you will find several TRN files:
FRENG.TRN English
FRFRA.TRN French (Thanks to Jean-Pierre GUTSATZ)
FRSPA.TRN Spanish
FRPOR.TRN Portuguese (Thanks to Gustavo Pinsard)
FRGER.TRN German (Thanks to Guenter Lamping)

If you want to switch your output language, just select a different file in the "general" tab, "Translation file" setting in the global area (See General Tab).
The file consist in a series of equates. Don't delete any of them, nor change its names. To customize your translation just change the text between single quotes.

For example, this entry from the FRENG.TRN file:

```
JALInspectTitle1   EQUATE('Field')
```

This is the title of the first column of the inspection window.
In the FRSPA.TRN file:

```
JALInspectTitle1   EQUATE('Campo')
```

If you do a translation to another language and care to share with us, we'll include it in the next revision, just Contact us.

Part 9

9 FAQ

We frequently receive the same questions by e-mail. This is a compilation of them.

9.1 Can FullRecord store the Audit-File(s) in MSSQL Tables instead of TPS?

Can FullRecord store the Audit-File(s) in MSSQL Tables instead of TPS?

The answer is YES, the audit file can be a MSSQL table. FullRecord is shipped with several examples, one of them is using MSSQL files. It can be in MySQL, Pervasive SQL and FireBird as well (we tested it with all these databases).

9.2 Will FullRecord work with FileManager?

Will FullRecord work with FileManager2 (or FileManager3) from Capesoft? How does it handle changing file structures?

Yes, it works with FM without problem, as far as compatibility. However FM don't act inside the records stored in the Audit file.
The record structure is not stored inside the audit file, just the data, and the template relies on the current dct structure to recover the information. It is mentioned in "Changing Record Layout" in the "Extension Templates" chapter, but in short, if you add fields to the end of the record structure then no changes are needed, but if you change field types or layout information, you should change the information in the audit file as well, with a process that reads the old information and resave it with the new layout. There is an extension template provided so you don't have to code, and detailed instructions on a simple way to do it in the mentioned chapter.

9.3 I need to remove FullRecord from an APP, how do I do it?

I need to remove FullRecord from an APP, how do I do it?

If it is a temporary removing because you need to solve any problem, just "disable" it globally; that will generate no code at all (the same as not having it), while preserves your settings in case you add it again.

If you simply need to stop the auditing process, but you have many manual calls and you don't want to change the program at this moment, then just "suspend" the template, so it won't audit, but it won't generate compiler errors either.

If the removing is permanent, delete the global extension first, then everything you added to the program (like the browse inspect window or manual calls), then the DCT files). Probably you will need to force a full regeneration (and/or delete the obj/clw/temp folder).

9.4 I am asked for a maintenance plan code. What's going on here?

I have purchased FullRecord a few months ago and I have never installed them. I have downloaded the latest and now I am asked for a maintenance plan code. What's going on here?

To install your FullRecord template you need a serial number and a maintenance plan code. These are e-mailed to you upon purchase.

As FullRecord 2.x is a major upgrade it will need a new serial number and maintenance plan, which will be e-mailed to you when you purchase the upgrade.

9.5 I see a ZLIBWAPI.DLL involved. I thought the product is full source? What is that?

When running the demo (FullRecord and AuditPack) I see a ZLIBWAPI.DLL involved. I thought the product is full source? What is that?

ZLIBWAPI.DLL and ZLIB.DLL are two files from a well known and tested open source compression library. It is used by FullRecord to compress the stored data (so saving a lot of space in the audit file), but it is not mandatory for you to use it; and the full source for that dlls is available at internet, although they are not developed in the Clarion language. See the Notes on Audit Data Compression.

9.6 Can FullRecord log every time a record is accessed, even if it is just viewed (no add/edit/delete)?

Can FullRecord log every time a record is accessed, even if it is just viewed (no add/edit/delete)?

You can make a manual call to store custom information at will, although, are you sure is that what you want? The amount of irrelevant information saved could go sky high. If you are concern about a particular access, FullRecord has an internal "message" save feature that allows you to mark any event you want (See Manual save of custom information); so if want to monitor a particular event (like entering a particular Form, or pressing a button, or selecting a menu or whatever) you just call this function and it will be saved in the audit file (without the user knowing about it), without adding all the records read from every audited file.

9.7 What about file encryption with SQL files?

I noticed that you encrypt the TPS file. With a SQL file, I suspect that the programmer should recur to using a different database user, to prevent ordinary users from selecting/updating/deleting the audit lines... is this your approach?

There are different tastes in the SQL approach, so I left it up to the developers choice. Some prefer to use strong encryption before/after reading the records, others are ok with no encryption at all.

9.8 May I filter records of type READ out from the Inspect Browse?

May I filter records of type READ out from the Inspect Browse?

If you like, it's easy enough to add a filter to the list box browse, although the ideal would be to process the Audit file and delete the old "Read" operations as they are deprecated and no longer necessary (Those from FullRecord 1.x).

9.9 Is there any way to disable FullRecord from recording reads?

FullRecord 2.x stores Read operations only if there is no previous matching operation for a change. If you start with an empty audit file on an already working system, most likely you'll see Read operations often at first, because there is no real tracking of the "history"

of the record. For example, on normal working, you "add" a record, then "change" it, and that's all what is saved (the "changed" record takes the "previous" information from the add). However, if you already have the existing record but no audit file, if you save a "change", there is no previous record already stored in the audit file to gather the "previous fields" information, so a "read" operation is saved, to be able to compare the current data with the old one.
If you change a record for the first time in an empty audit file, you'll see a read-change operation. However, if you change again the same record, you should see only a "change" operation.

9.10 Can I format Date and Time Fields in the "Record Contents" Window?

Can I format Date and Time Fields in the "Record Contents" Window and all Fields in the columns to be left-justified?

The fields inside the "Record Contents" window are formatted according to your DCT settings. So if you want them show a "date" or "time" picture, simply specify such format in the field, inside your DCT. Regarding the fields inside the "Record Contents" window, they -are- formatted left justified. They look moved to the right because of the mask (it could be something like @n8, for example, where if you have a single digit you will see 7 spaces prior to the actual digit).

9.11 Can FullRecord restore a modified record?

Can FullRecord restore a modified record?

Yes it can; the "recover" operation is not limited to just "undeletes", meaning you can recover for a modified record or from an original record. As the template is full source, this can be limited to recovering just deletes very easily, though, if you want that.

9.12 If I have a process running, will FullRecord update the changes done by that process?

If I have a process running, will FullRecord update the changes done by that process?

Yes, it will. You can check that in the demo, the "increase points" option in the "demo" menu is a process.

9.13 Do we have to do any coding?

What effort is required to integrate this Audit Trail in our application? Do we have to do any coding? For e.g.. Say how audit trail will be updated when a new invoice is created / modified/ deleted? Do we have to call some procedures of audit trail from our program and pass values as parameters? If coding required, what would be the volume?

You have to do absolutely no coding. Every ABC or Legacy operation will be automatically recorded.

9.14 What is the licensing model? Is this on royalty free distribution model?

What is the licensing model? Is this on royalty free distribution model?

FullRecord is full source code, there is no royalties or run times required to use it.
Regarding your finished program, you can add FullRecord in as many as you want, and distribute it freely.
Regarding your development team, you have to buy one licence for each machine were FullRecord will be installed.

9.15 Is there any restriction in number of fields or data volume?

Is there any restriction in number of fields / data volume?

The data is stored as a packed record structure inside the Audit file, with several unpacked fields used for indexing and searching. The structure of each record is taken from the current dictionary, it is not stored in the Audit file.
We made a contest to test the template under heavy stress, and we verified that the template can handle more than 900 files without problems. In our internal testings from version 1.7 and later is handling more than 900 files for recovering as well.
There is no limit in the number of fields that we are aware of (we tested with a file of 1700 fields).
The stored data volume is dependant on the database system you choose. If you pick TPS files, there is a 2 Gb size file limit (which still should be several million records, depending on your design). With other file systems (Like SQL or Btrieve 8.x+) the limit is given by your hardware storage.

9.16 Is there a way to store my own fields in the audit file?

Sure there is!
Suppose you want to add the full path and disk file name of each file. Add a "File folder" field to the audit file, name it as you want and make it as long as needed.
In the global embed "FullRecord: Prime fields for audit file" simply add
AUD:FileFolder = TheFile{prop:name}
Being AUD:FileFolder the new field you created.
TheFile{prop:name} will give you the full path and disk file name of the audited file, if you want strictly the path you'll need to take off the name.

9.17 I have an older version of FullRecord. How do I get an upgrade to this product?

I have an older version of FullRecord. How do I get an upgrade to this product?

If your maintenance plan is still current, you can purchase this major upgrade for a very low fee of U$S 49.
If your maintenance plan expired but you own an old version of FullRecord, you still can have a reduced especial price of U$S 99. The full price of this new version is of U$S 149.
Visit www.clariontemplates.com to find links to further information and purchase.

9.18 I set everything but nothing is audited. What's wrong?

I set everything but nothing is audited (and no errors are shown). What's wrong?

There are many reasons because the template might be not working.
Here you have a checklist: You have to check these in the data dll AND on each app.

- Are you using the FullRecord.fil file to load the files?

Make sure the file is in the app folder.
Check that the file list is there and matches your dct.
In the template, "Files to Audit" tab, be sure to actually SELECT the files you want to audit (Press "Select All" if you want to audit the whole list).

- If you are not using the FullRecord.fil file to load the files, double check that you have the files to audit SELECTED in the "Files to Audit" tab, on every app.
- In the "more settings" tab, check that "Disable file Auditing code generation" or "Suspend File Auditing" are NOT checked.
- If you set a Run-time Disabling variable, be sure it is not set to TRUE when your program runs.
- If your program uses the Clarion template chain (a.k.a. Legacy) check that you have active the global switches "Enable Triggers Support" in the "File Control Flags" tab and "Enable de use of ABC classes" in the "Classes" tab.

9.19 When I press the "View Record" button nothing happens

I can see the audit records in the Audit Browse, but when I press the "View Record" button nothing happens. What's wrong?

This means the audited file is not in the list of active files to be inspected.
Check the Activate Record Inspect and Recover feature and that the file you want to inspect is selected in the list of audited files in the app where your Audit Browse procedure is stored.

9.20 Clarion 7 does not generates TXD, how do I use TXD Analyser?

In Clarion 7 you CAN generate TXD files.
From the DCT Explorer toolbar, you should see a button whose tool tip says "Import/Export". On that button, select "Export Dictionary to Text".
When the dialog to export pops up, you will see the DCTX extension by default. Select "All Files" from the File of Type Drop List, and then enter "anyname.TXD".
When you give the file name a TXD extension, the export process will save as TXD instead of DCTX.

9.21 I'm getting compile errors in the PrimeAudit Routine.

Starting with FullRecord 3.00, there are new fields added to the Audit files. Those fields have to be added to the DCT and declared in the Definitions 2 Tab. If you left the fields empty in that Tab, you will get missing declarations like this:

```
PrimeAudit Routine !1
CLEAR()
CLEAR()
= 0
= 0
CLEAR()
= 0
= 0
AUD:Date = TODAY()
AUD:Time = CLOCK()
AUD:User = FRGLO:User
etc.
```

Part 10

10 Guide to examples and tutorials

10.1 ABC - FullRecord (Without NeatMessage)

Topspeed (ISAM):
In your Clarion installation, search in the 3rdParty\Examples\FullRecord2 folder and look for the FRA6.app for ABC. In case you are asked for the dictionary, specify FullRecord.DCT. This example app was created with Clarion 6, if you plan to open it with a later Clarion version you should backup it first with a different name, otherwise you won't be able to open it with Clarion 6.x again. If this happens so, you may reinstall the template to get the original example APP and DCT back.

Firebird:
If you look for the Firebird example, open the FRFBA6.app in the examples folder. This example app was created with Clarion 6.3, and it should work with any version of Clarion 6.x. Entering the global area, there is just one embed, where you may change the datapath and other data for the Firebird database.
You have to manually create the Firebird database externally prior to run the example program. Match the embedded data with your database (path, name, user and password). Once created the database, the files will be created between it as the program runs.
This example was tested with Firebird 1.5.3.4870 and Firebird ODBC 1.2.0.69. It may has problems with other combinations of versions.

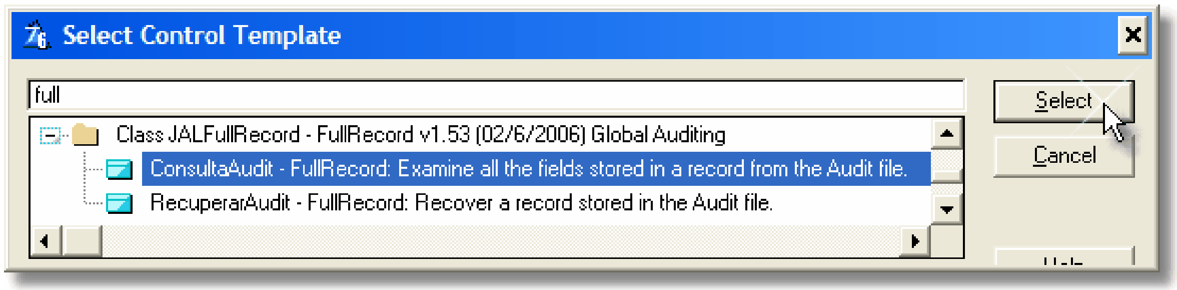
MS-SQL Server:
If you look for the MS-SQL Server example, open the FR6SQL.app in the examples folder. This example app was created with Clarion 6.3, and it should work with any version of Clarion 6.x. Entering the global area, there is just one embed, where you may change the database connection string and other data for the MS-SQL database.
You have to manually create the MS-SQL database externally prior to run the example program. Match the embedded data with your database (path, name, user and password). Once created the database, the files will be created between it as the program runs.
This example was tested with MS-SQL Server Express 2005.
The database can be shared with the Legacy MS-SQL example.

IMDD:
If you look for the IMDD example, open the FRAIM6.app in the examples folder. This example app was created with Clarion 6.3, and it should work with any version of Clarion 6.x. You will need the IMDD driver to test this app, purchased separately from SoftVelocity.
This example was tested with the 2.2 version of the IMDD driver.

All of them:
When you open the example app, first go to the global properties and review the values in the different tabs. Verify and understand them according to what was explained before in the "Template usage" section of this manual.
Following the next sections, you will understand how these procedures were created and you may recreate them in your own applications.
In your application procedure tree, you will find the following:

10.1.1 BrowseAudit (Example of auditing inspector)

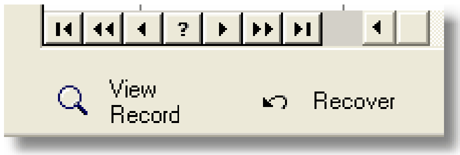
This is a Wizard created Browse over the auditing file. In this Browse you will add the elements that allow you to search and query the audit records. In this example you can locate for all the available keys. You can add your own keys, a Query search (the Clarion 6.x advanced query is highly recommended) and even better, you may enter a SELECT clause if your auditing file is an SQL one.
In this Browse you just simply drop two control templates, "ConsultaAudit" and "RecuperarAudit", so you can inspect the contents of a record and recover it, respectively. When you select the option "Populate control template", you will see the following window, where you should select the mentioned controls:



Class JALFullRecord - FullRecord v1.53 (02/6/2006) Global Auditing
ConsultaAudit - FullRecord: Examine all the fields stored in a record from the Audit file.
RecuperarAudit - FullRecord: Recover a record stored in the Audit file.

Note: As these screen captures are not often updated unless there are significant changes, the date and version of the template may differ from what you currently have. It will also be different according to the chain you are using (ABC or Legacy).

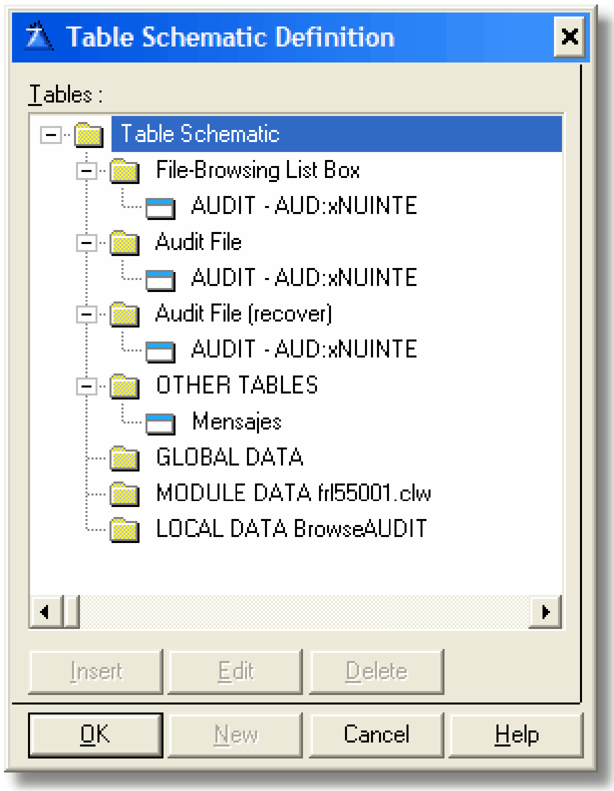
These to controls will look like this:



You may change the icons, text, position, size and attributes of these buttons as fit.

From the "ConsultaAudit" control template properties (View Record) you have to access the "Tables" button to enter the procedures "File Schematic". You will find there an "Audit file" line where you have to specify the name of the file that holds the auditing records.
This may looks like a duplication but this way you have the flexibility to browse a different file from the one which actually holds the auditing records, which be analyzed when you press the button. Normally, it will be the same file.

The same goes to the "Recover" control template.



Tables :

- Table Schematic
  - File-Browsing List Box
    - AUDIT - AUD:xNUINTE
  - Audit File
    - AUDIT - AUD:xNUINTE
  - Audit File (recover)
    - AUDIT - AUD:xNUINTE
  - OTHER TABLES
    - Mensajes
  - GLOBAL DATA
  - MODULE DATA frl55001.clw
  - LOCAL DATA BrowseAUDIT

Important: If you are going to inspect an audit file where messages were recorded (By the NeatMessage message box) you have to specify the "Mensajes" file in OTHER TABLES as shown above, otherwise you won't be able to see the text of the messages.

10.1.2 BrowseAuditBck (Example of historical audit data inspection)

This is just a copy of the Browse “BrowseAudit” with the primary file changed. This Browse browses the historical audit file (AuditBck) create with the previous process (CleanAudit), and allows you to inspect the old data the same way you do to the current one.
You can create it from scratch in the same way as you did “BrowseAudit”, except that you must change the files involved in the “FileSchematic”.

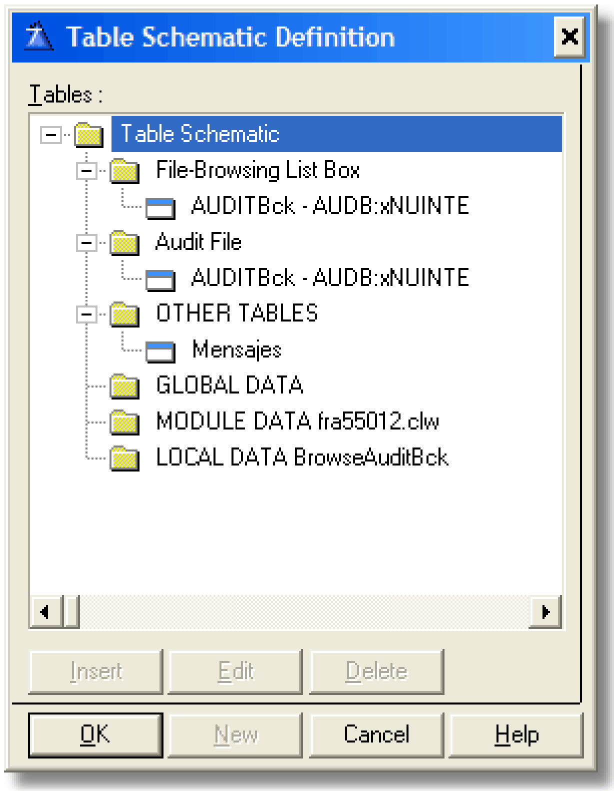


Table Schematic Definition

Tables :

```
Table Schematic
  File-Browsing List Box
    AUDITBck - AUDB:xNUINTE
  Audit File
    AUDITBck - AUDB:xNUINTE
  OTHER TABLES
    Mensajes
  GLOBAL DATA
  MODULE DATA fra55012.clw
  LOCAL DATA BrowseAuditBck
```

This is possible because the control templates from FullRecord works on any audit file, as long as they have the exact same layout as the main Audit file (declared in the global extension).

If you just want to copy the “BrowseAudit” procedure, follow these steps:

1) Copy the procedure with other name (For example, “BrowseAuditBck”)
2) Enter the window source by the ellipsis button (...) and do a search & replace over the file prefix: if the audit file is AUD: and the other file is AUDB:, then search for AUD: and replace it for AUDB:.
3) Accept the window and go to the “File Schematic”, delete the “Audit” file everywhere and replace it for the new one, “AuditBck”.
4) Enter the “Conditional behavior” Tab in the Window properties, and replace the “Key to use” by the corresponding keys for the new file.

That's it, now the copied browse should work with the historical file, without having to redoing the browse from scratch.

10.1.3 CleanAudit (Example of audit cleanup)

This is a Wizard created Process template over the Audit file. This process allow you to quickly and automatically move the old records from the Audit file to a separated historical audit file (We will call it AuditBck) while compacts the current audit file. You probably won't need this process for a while, until the system is working at full capacity. It will depend on the movements per day and the period you want to keep your audit information "alive" for your most common queries.

Actually, we noticed that the queries to the audit file become slow when the audit file is holding over a million records. You should analyze which is the working "window" you should keep in your audit file (normally six months seems to be enough to any kind of investigation).

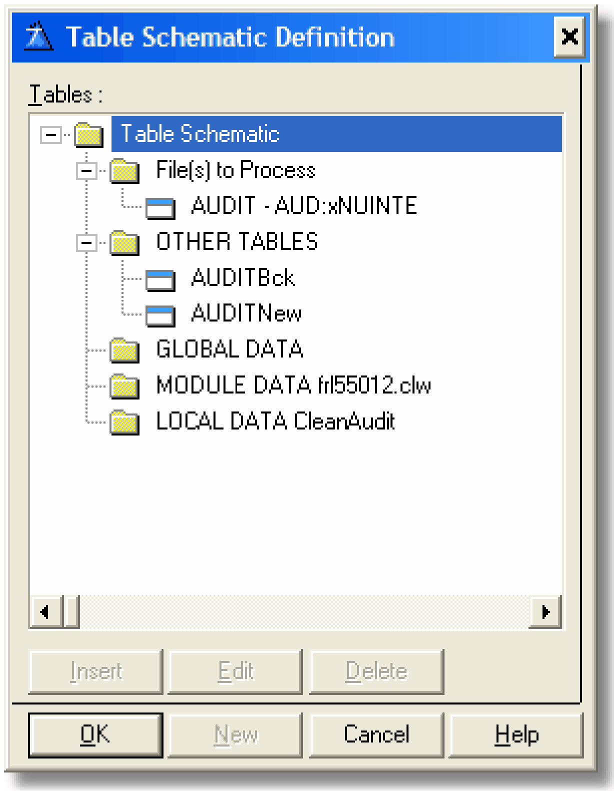
Note: There are no template imposed limits regarding the amount of audit records you store in the audit file, but there are limits imposed by the chosen file driver system and server OS. Different file types (like Topspeed, Btrieve, MS-SQL, etc.) have very different record limits, physical size limits and response times. Which is the best for you depend completely in your particular situation.

This template works "cutting" the audit file in two. You have to specify a "cut date", and all the records from the beginning to before this date will be moved to the historical file (AuditBck), while the records from this date on to the end of the audit file will be moved to a new file (AuditNew). When the process finishes, the current audit file is removed and the new one is renamed as the current one (AuditNew becomes Audit).

Important: Given the way this template works, it is indispensable that

- You make a backup of all your data files before you begin.
- There is nobody using the system while the process run.
- If an error occurs when the process finishes, you must don't do anything else or permit that anyone else use the system until the error is solved.

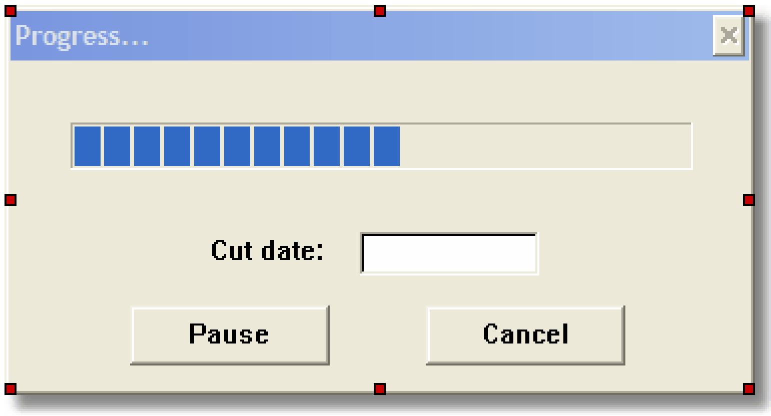
If you want to create this procedure from scratch, you start by use a Wizard to create an empty "Process" over the audit file. Entering the "File Schematic", you must specify the following files:



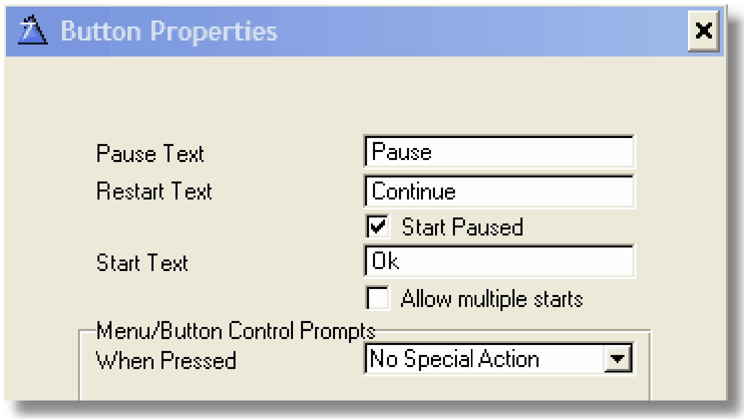
```
Table Schematic
  File(s) to Process
    AUDIT - AUD:xNUINTE
  OTHER TABLES
    AUDITBck
    AUDITNew
  GLOBAL DATA
  MODULE DATA frl55012.clw
  LOCAL DATA CleanAudit
```

Where "AUDIT" is the audit file you want to cleanup, and AUDITBck and AUDITNew have to be defined in the dictionary exactly alike to the audit file. The suffix must be specified in the global extension as shown in "Template usage".

Somewhere (locally or globally) you have to declare a field that will receive the "cut date", and you should ask for it in the process window. You also have to populate Clarion's "Pause" control template, so the process ask the date to you before starting the process. The process window may look like this:



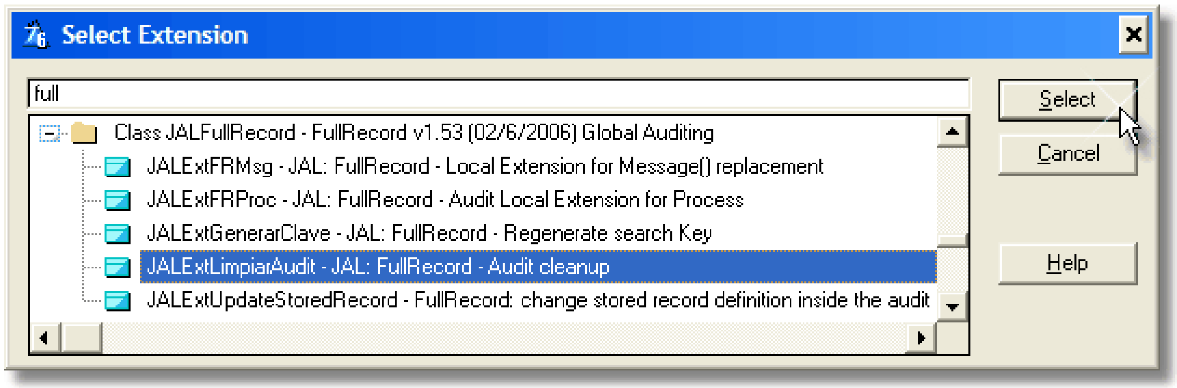
Looking at the "Pause" button properties, you have to define the following characteristics:



Pause Text: Pause
Restart Text: Continue
[✓] Start Paused
Start Text: Ok
[ ] Allow multiple starts
Menu/Button Control Prompts
When Pressed: No Special Action

It is *very* important that you check the "Start Paused" checkbox, so the process don't start until you press the button.

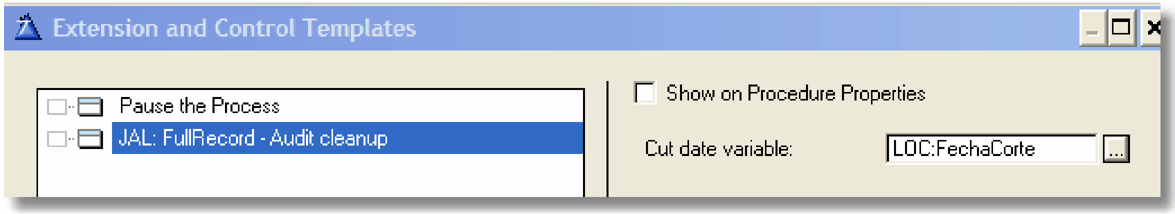
In this example you are going to use an extension to activate the template. Entering by the "Extensions" button, you have to insert a new extension using the "insert" button. The extension template list will show, from which you will pick from the FullRecord extensions the one called "JALExtLimpiarAudit".



full

Class JALFullRecord - FullRecord v1.53 (02/6/2006) Global Auditing
- JALExtFRMsg - JAL: FullRecord - Local Extension for Message() replacement
- JALExtFRProc - JAL: FullRecord - Audit Local Extension for Process
- JALExtGenerarClave - JAL: FullRecord - Regenerate search Key
- JALExtLimpiarAudit - JAL: FullRecord - Audit cleanup
- JALExtUpdateStoredRecord - FullRecord: change stored record definition inside the audit

When you select this extension, you have to specify which variable will receive the cut date.


Extension and Control Templates
Pause the Process
JAL: FullRecord - Audit cleanup
Show on Procedure Properties

Cut date variable: LOC:FechaCorte

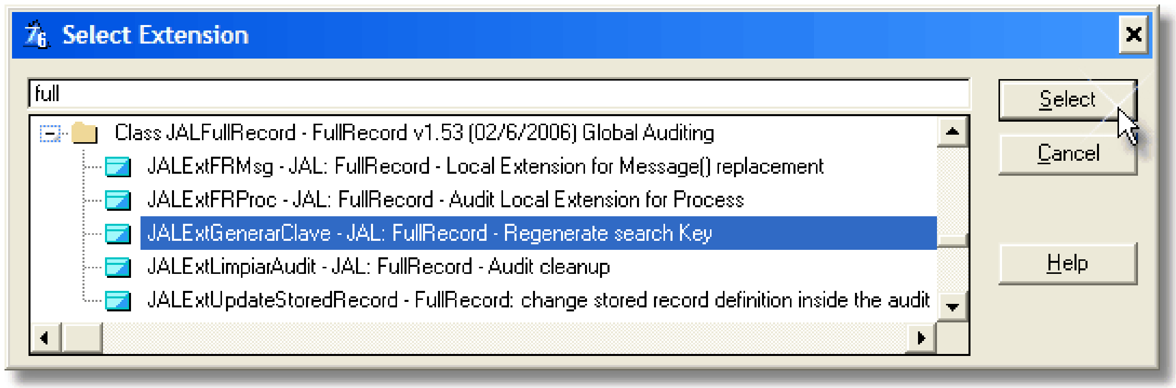
In this case, you have to select the same date variable that you populated in the process window.

<u>Note</u>: Selecting the variable here, allows you to assign the variable in other procedure, using a global variable, or even calculate it. In these cases, you don't need to enter the variable in the process window, and therefore you won't need to populate the "Pause" control template.

10.1.4 ReprocessSearchKey and ReprocessSearchKeyBck (Examples of search key reprocessing)

These are Wizard created Process templates over the audit file (or over the historical one). This process allows you to regenerate the search key for the records where it was changed.
In the “File Schematic” you have to specify the file to process. It could be the audit file (AUDIT in ReprocessSearchKey) or the historical one (For example, AUDITBck in the ReprocessSearchKeyBck process, the one which is generated in the cleanup process).

Then, you have to enter the “Extensions” button and select and insert the “JALExtGenerarClave” extension.



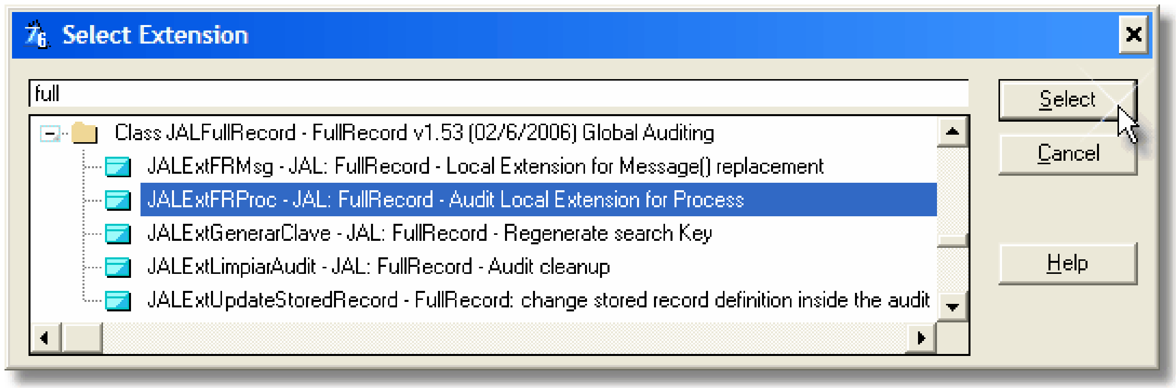
full

Class JALFullRecord - FullRecord v1.53 (02/6/2006) Global Auditing
- JALExtFRMsg - JAL: FullRecord - Local Extension for Message() replacement
- JALExtFRProc - JAL: FullRecord - Audit Local Extension for Process
- JALExtGenerarClave - JAL: FullRecord - Regenerate search Key
- JALExtLimpiarAudit - JAL: FullRecord - Audit cleanup
- JALExtUpdateStoredRecord - FullRecord: change stored record definition inside the audit

Simply with this, the process is ready to use.

10.1.5 ProcessABC (Example of using the "process" extension)

This Wizard created “Process” template processes the records from the “People” file and add a random value from 1 to 5 to a “points” field, which is a 4-dimensioned field in the “People” file.
Here you will see how to add the extension for the process. Simply enter using the “Extensions” button and insert the “JALExtFRProc” extension.



full

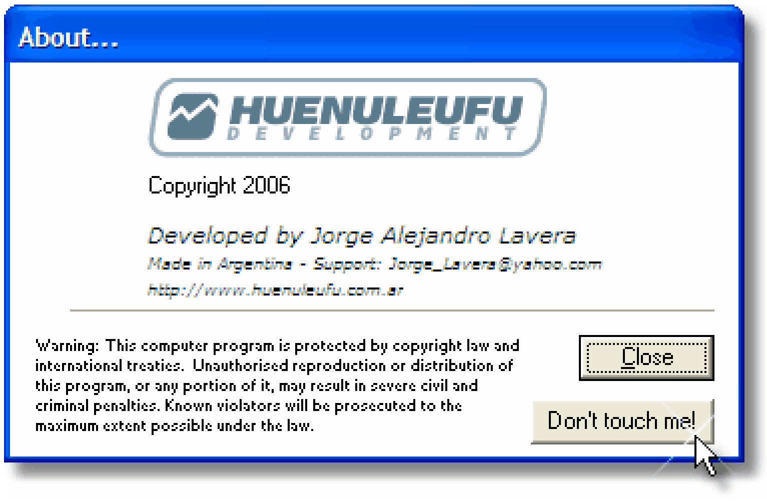
Class JALFullRecord - FullRecord v1.53 (02/6/2006) Global Auditing
- JALExtFRMsg - JAL: FullRecord - Local Extension for Message() replacement
- JALExtFRProc - JAL: FullRecord - Audit Local Extension for Process
- JALExtGenerarClave - JAL: FullRecord - Regenerate search Key
- JALExtLimpiarAudit - JAL: FullRecord - Audit cleanup
- JALExtUpdateStoredRecord - FullRecord: change stored record definition inside the audit

When using TopSpeed files, the speed of the process could be compromised.

The template included by FullRecord does not make use of Logout/Commit, as these commands could be used only once (cannot be nested). If you want to accelerate the process, you can manually include the Audit file in your LOGOUT statement, but keep in mind that LOGOUT locks the file, therefore in a multiuser environment all the others users will be locked until this process finished.
You may use STREAM/FLUSH on the Audit file at the beginning of the process to achieve the same result.

10.1.6 AboutWindow (Manual Event audit)

In the "About" window there is a second button added, labeled "Don't touch me".



Copyright 2006

Developed by Jorge Alejandro Lavera
Made in Argentina - Support: Jorge_Lavera@yahoo.com
http://www.huenuleufu.com.ar

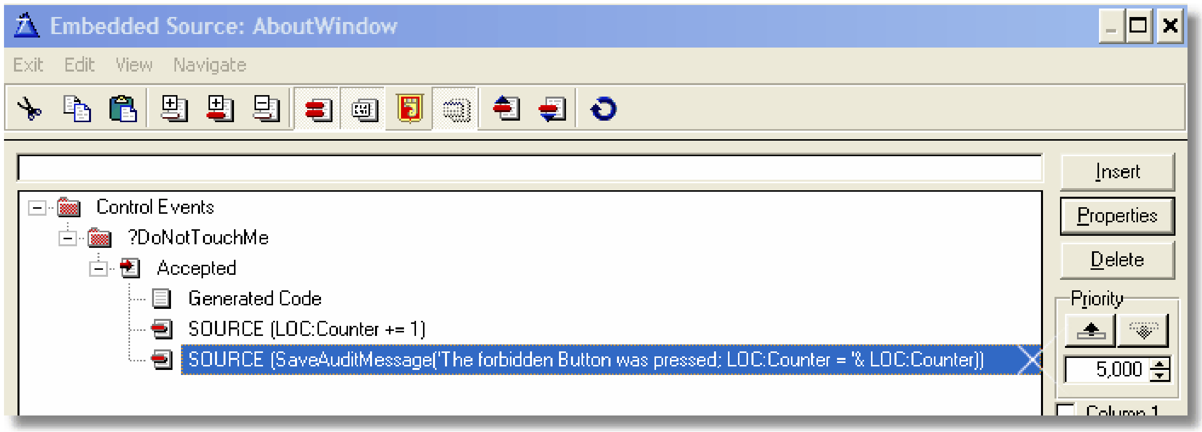
Warning: This computer program is protected by copyright law and international treaties. Unauthorised reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties. Known violators will be prosecuted to the maximum extent possible under the law.

In the event "Accepted" for the "Don't touch me" button there is a piece of manual code;

```
SaveAuditMessage('The forbidden Button was pressed; LOC:Counter = '& LOC:Counter)
```

LOC:Counter is a local static LONG type variable.

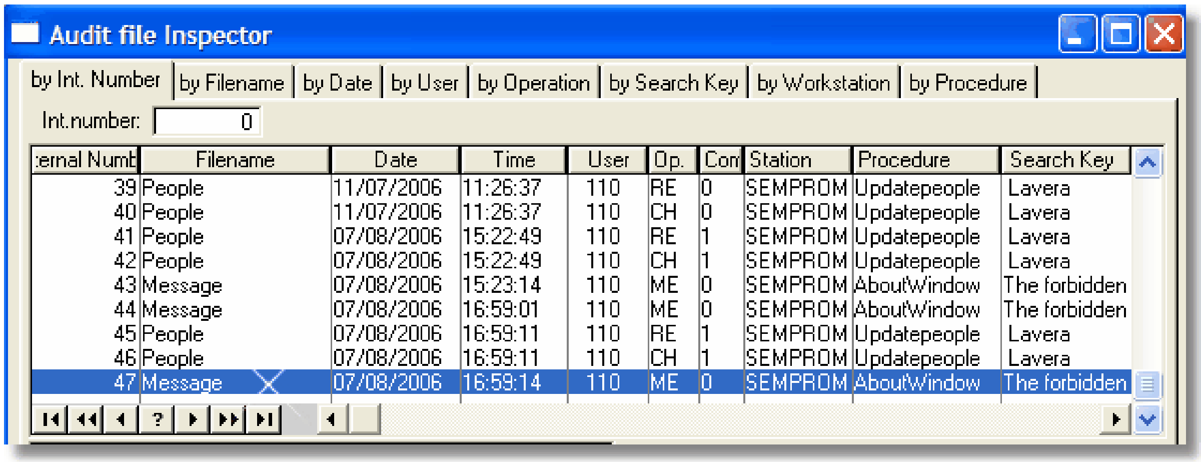
The code is embedded as follows:



```
Control Events
  ?DoNotTouchMe
    Accepted
      Generated Code
      SOURCE (LOC:Counter += 1)
      SOURCE (SaveAuditMessage('The forbidden Button was pressed; LOC:Counter = '& LOC:Counter))
```

With this code, when you press the button, the message "The forbidden Button was pressed" and the actual value of the local variable is saved into the Audit queue.

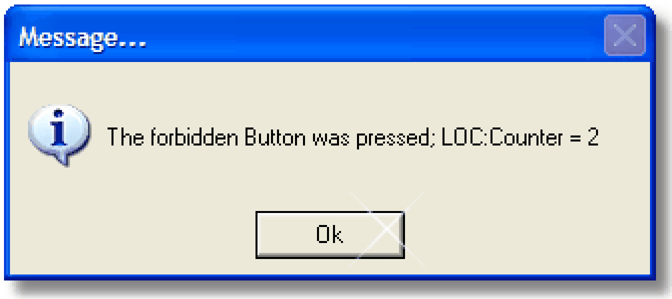
You will see the manual event in the correct order between the file audit operations.



Int.number: 0

ernal Numb	Filename	Date	Time	User	Op.	Con	Station	Procedure	Search Key
39	People	11/07/2006	11:26:37	110	RE	0	SEMPROM	Updatepeople	Lavera
40	People	11/07/2006	11:26:37	110	CH	0	SEMPROM	Updatepeople	Lavera
41	People	07/08/2006	15:22:49	110	RE	1	SEMPROM	Updatepeople	Lavera
42	People	07/08/2006	15:22:49	110	CH	1	SEMPROM	Updatepeople	Lavera
43	Message	07/08/2006	15:23:14	110	ME	0	SEMPROM	AboutWindow	The forbidden
44	Message	07/08/2006	16:59:01	110	ME	0	SEMPROM	AboutWindow	The forbidden
45	People	07/08/2006	16:59:11	110	RE	1	SEMPROM	Updatepeople	Lavera
46	People	07/08/2006	16:59:11	110	CH	1	SEMPROM	Updatepeople	Lavera
47	Message	07/08/2006	16:59:14	110	ME	0	SEMPROM	AboutWindow	The forbidden

If you press the "View Record" button, you will see the whole message saved.



The forbidden Button was pressed; LOC:Counter = 2

10.2 ABC - FullRecord + NeatMessage

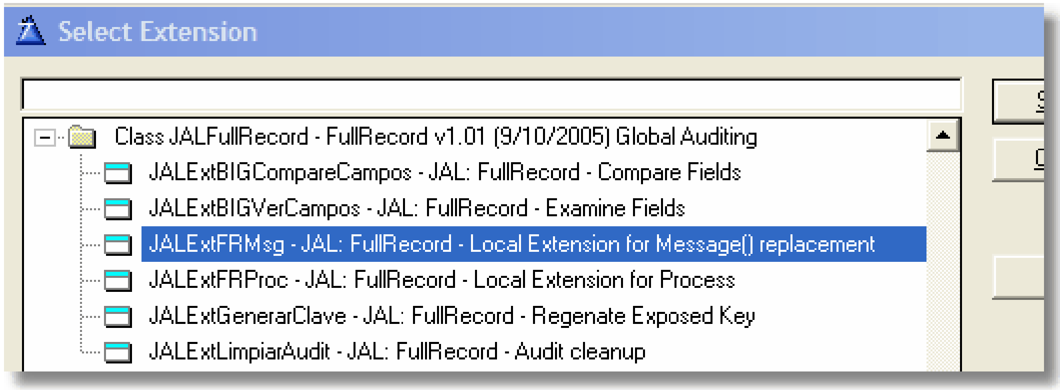
In your Clarion installation, search in the 3rdParty\Examples\FullRecord folder and look for the RSA6.app. In case you are asked for the dictionary, specify FullRecord.DCT. This example app is created with Clarion 6, if you plan to open it with a later Clarion version you should backup it first with a different name, otherwise you won't be able to open it with Clarion 6.x again. If this happens so, you may reinstall the template to get the original example APP and DCT back.
This example is identical to the previous one, with the specific procedures that are only available if you also purchased NeatMessage. In this section you will only find the procedures that are particular to this example. For the rest of the procedures, refer to the previous example.
In the global area, you have to insert the NeatMessage extension and set it up as your convenience (Following the NeatMessage manual directives)

Important: When you use NeatMessage, make sure that all the windows inside the Frame are MDI, otherwise you might have run-time errors. Take special care with wizard-generated procedures, as processes and reports, which are often created without the MDI attribute.

In your application procedure tree, you will find the following:

10.2.1 Mensaj (Example of user's message registration

This is the procedure that registers to the audit file the messages that are shown to the user. If you want to create this procedure in your program, select the “File” menu, option “Import from application” and import the MENSAJ procedure from the NMA55.APP program supplied in your examples folder under NeatMessage.
Go the “Extensions” button and insert the “JALExtFRMsg” extension, the FullRecord local extension for the message box.



Class JALFullRecord - FullRecord v1.01 (9/10/2005) Global Auditing
JALExtBIGCompareCampos - JAL: FullRecord - Compare Fields
JALExtBIGVerCampos - JAL: FullRecord - Examine Fields
JALExtFRMsg - JAL: FullRecord - Local Extension for Message() replacement
JALExtFRProc - JAL: FullRecord - Local Extension for Process
JALExtGenerarClave - JAL: FullRecord - Regenate Exposed Key
JALExtLimpiarAudit - JAL: FullRecord - Audit cleanup

Once you did this, you have to go to the “File Schematic” and specify which file is going to receive the messages descriptions (The file “Mensajes” in our dictionary).
Finally, go to the application global properties, press “Extensions”, select the “NeatMessage” one, go to the “Extras” tab and press the “Message()” button. Enter the “Template Hook” button and activate the checkbox “replace Message() function”.
Below you have to specify the name of the procedure that holds the message box replacement (“Mensaj”, in this example).
When you do that all the messages that the user sees will be recorded in the audit file with the operation code “ME”. The message text itself it's recorded to the specified file (Mensajes), and in the audit file there only be a reference to that file, for saving space.

10.3 ABC - FullRecord - Multi-DLL

For full functionality of FullRecord, the Global extension needs to be included in every app of your system.

The multi-dll example is composed by the following programs:

FRData6.APP for Clarion 6.x is the main data dll. The global extension declares which files will be audited, and the app export all the files definitions.

FRInpect6.APP for Clarion 6.x contains the auditing procedures. The app refers to FRData6. The global extension declared which files will be inspected and/or recovered. It should match the list of files from FRData6.app, and as the files are not used in the app, they generation should be forced in the "individual files overrides" global option.

FRMod6.APP for Clarion 6.x contains the user procedures. Note that it does contains the global extension and refers to FRData6. It does not refer to FRInspect6. The files in this procedure to be audited are the ones declared in the global list from FRData6.app.

FRExe6.APP for Clarion 6.x is the main EXE caller program. Note that it does contains the global extension too. It refers to the other three apps.

The procedures in the example are the very same as in the ABC - FullRecord example. This example is provided so you can check the proper global settings for a multi-dll system to run.

Take note: In the case that the inspection and recovery templates are not in the data-dll, you have to select the files to inspect in the global extension the same way as you did in the data-dll, as shown in the FRInspect6.app.

10.4 Legacy - FullRecord (Without NeatMessage)

In your Clarion installation, search in the 3rdParty\Examples\FullRecord folder and look for the FRL6.app. In case you are asked for the dictionary, specify FullRecord.DCT. This example app is created with Clarion 6, if you plan to open it with a later Clarion version you should backup it first with a different name, otherwise you won't be able to open it with Clarion 6.x again. If this happens so, you may reinstall the template to get the original example APP and DCT back.
This example is identical to the ABC one. In this section, you will only find the procedures that are particular to this example or have differences with the ABC version. For the rest of the procedures, refer to the ABC example.

MS-SQL Server:
There is a MS-SQL Server Legacy example, open the FR6LSQL.app in the examples folder. This example app was created with Clarion 6.3, and it should work with any version of Clarion 6.x. Entering the global area, there is just one embed, where you may change the database connection string and other data for the MS-SQL database.
You have to manually create the MS-SQL database externally prior to run the example program. Match the embedded data with your database (path, name, user and password). Once created the database, the files will be created between it as the program runs.
This example was tested with MS-SQL Server Express 2005.
The database can be shared with the ABC MS-SQL example.

Following the next sections, you will understand how these procedures were created and you may recreate them in your own applications.

In your application procedure tree, you will find the following:

10.4.1 BrowseAudit (Example of auditing inspector)

First, read all about BrowseAudit (Example of auditing inspector) in the ABC example.

For the Legacy chain, you have to specify the audit file in the File Schematic for both Control Templates. Both should have the same file specified, as you can see in the following window:

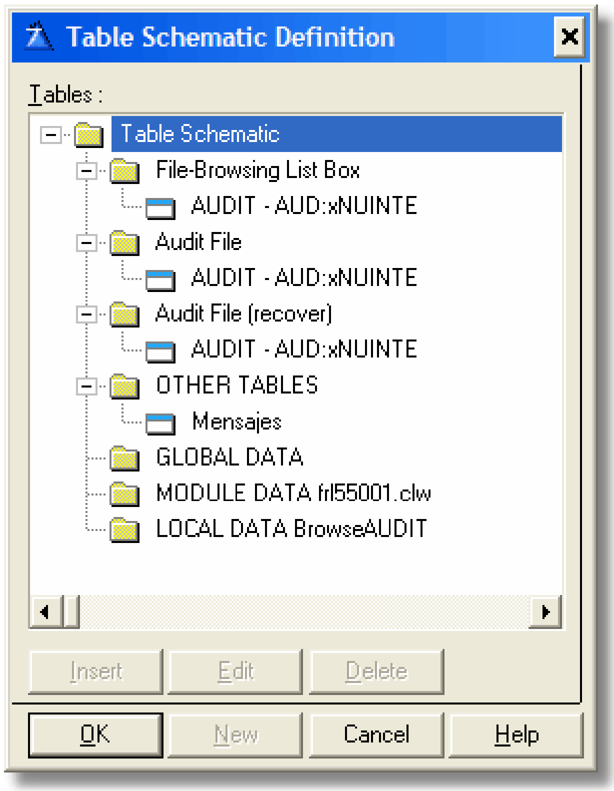


Table Schematic
- File-Browsing List Box
  - AUDIT - AUD:xNUINTE
- Audit File
  - AUDIT - AUD:xNUINTE
- Audit File (recover)
  - AUDIT - AUD:xNUINTE
- OTHER TABLES
  - Mensajes
- GLOBAL DATA
- MODULE DATA frl55001.clw
- LOCAL DATA BrowseAUDIT

The same applies for *BrowseAuditBck*.

10.4.2 CleanAudit (Example of audit cleanup)

First, read all about CleanAudit (Example of audit cleanup) in the ABC example.

For the Legacy chain, the “Process” won't allow you to define a Window in the same process, therefore the input of the “cut date” have to be done before to calling this procedure.
In the example, a Global Variable is used (GLO:CutDate) which is entered in the *RangCleanAudit* window. This window simply loads the date, and with the OK button calls the CleanAudit process.
In the local extension for CleanAudit you specify the global variable instead of a local one.
Everything else works the same.

10.4.3 ProcessLegacy (Example of auditing a legacy "process")

Like in the ABC one (ProcessABC (Example of using the "process" extension)), this Wizard created “Process” template processes the records from the “People” file and add a random value from 1 to 5 to a “points” field, which is a 4-dimensioned field in the “People” file.
FullRecord automatically audits the movements performed over the primary file (Either puts or deletes) without you having to do anything special.

Part 11

11 Template Considerations

11.1 BLOB fields

FullRecord from version 1.70 on, does support BLOB fields.

File record audited	Yes
BLOB audited	Yes
BLOB can be inspected	Limited (*)
BLOB can be recovered	Yes

For auditing BLOB fields, you have to add a Blob field to the audit file definition in your DCT, and name it BLOB1 for the first Blob field, BLOB2 for the second and so on.

(*) You can see the size (in bytes) of the stored Blob.

The Stored Record can be stored in a Blob field as well.

11.2 Cascade deletes

When a cascade delete is made in your system, FullRecord audits all the "children" deleted at the same time the deletion occurs. However, if the cascade is cancelled because of a RI restriction, the records already recorded as deleted will remain in the audit file, if the Audit file is not included in the Logout() frame. This may lead to have false "delete" entries in the audit file in these particular case.

11.3 Caution with IMDD files

When using IMDD files, you need to take care if you use the **/THREADEDCONTENT** switch. As this will make each thread instance of the file with its own data, you may have difficulties to audit or undelete records to the audited file.

11.4 Compatibility with other templates

There is no way to test this template with all other combination of template available, although because of the way it is programmed it should not have any problems. However, if you note strange behavior because of using it at the same time of another third party, please contact us so we can try to fix the problem.

We noted that FullRecord is incompatible with the DET templates.

11.5 Recovering of cascade operations

Automated recover from cascade operations is not supported yet.
As for now, you have to recover the changes/deletes one by one.

11.6 Dimensioned GROUPS in not-fullrecord mode.

When the audit information is saved in "Mixed" or "Changed Fields" mode, the system cannot handle DIMensioned GROUPs. If you have such GROUPS, any changes made to them or fields within them, will be skipped (Not audited).
They will be properly saved in Full Record mode.

Part 12

12 Generation, Compiler and Run-Time Errors

12.1 General Troubleshooting

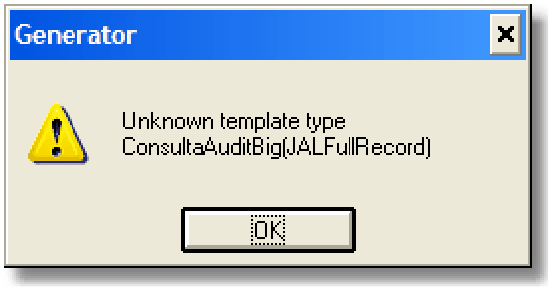
a) Each time you move a procedure from one dll to another, you HAVE TO recompile ALL the dlls and the exe. This is not an option. Unpredictable results will happen if you fail to do so.

b) When you includes the FullRecord global extension in a secondary dll for using the "inspect" or the "recover" features, you have to verify that ALL selections are the same and correct, and that the list of files to audit is the same in both dll (the global and the secondary). Pay special attention to the "Definitions1" and "Files to Audit" tabs.

c) Trying to determine what's wrong with a list of files might be overwhelming. Start with ONE file and see what happens. Audit just this one file, and inspect just this one file. Does it works?

12.2 Generator: Unknown template type

If you are upgrading from a very old version of FullRecord (previous to 1.10), you may see this window when you open your app:



Unknown template type
ConsultaAuditBig(JALFullRecord)

This is because the "big" DCT support was dropped from version 1.10 of FullRecord. Just press "ok" as many times as needed (that would probably be 3 for each audit browse you did).

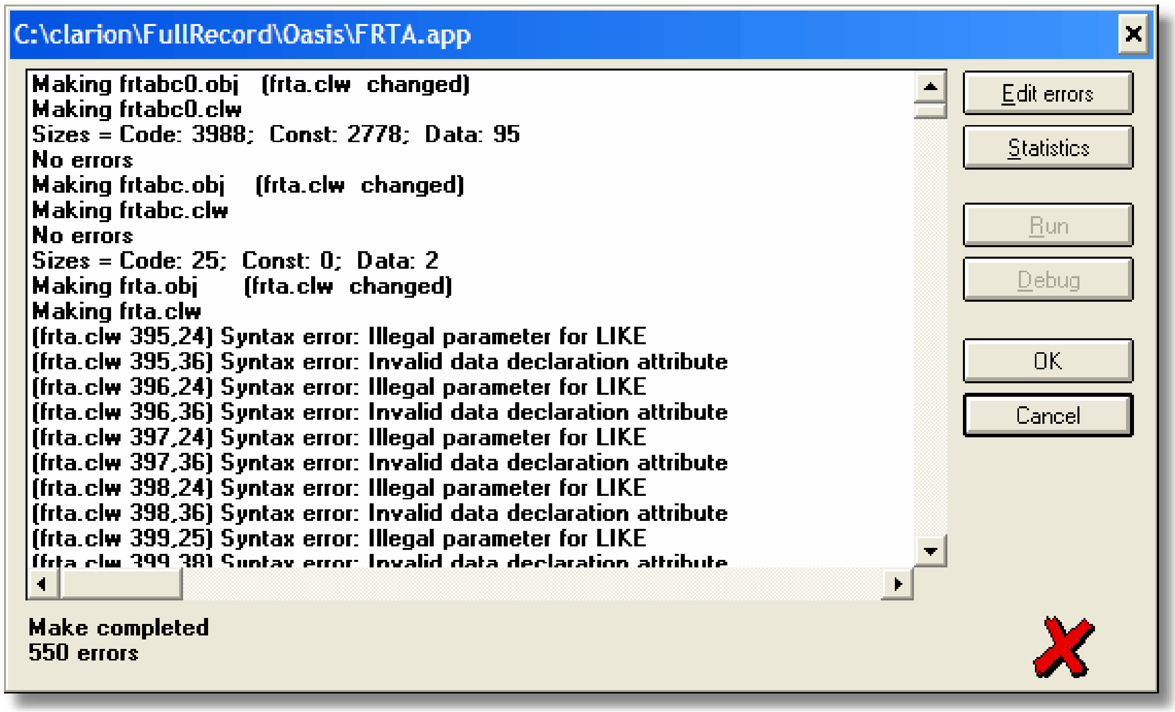
You have to delete the "source" procedures generated with version 1.0x (ViewFields and CompFields), and delete and populate again the "View Record" control template in your browse inspector windows. If you fail to do this, you will have the old buttons on the window, but without functionality (Nothing is going to happen when you press the button).

12.3 Syntax error: Illegal parameter for LIKE

This error occurs when you specify to audit a file that in fact it's not in use anywhere in your system. Meaning, the file exists in your dictionary, but Clarion is not generating its definition code because the file it's not in use in any template. To avoid this error, either

- Unmark this file from the audit list file, or
- Manually specify in your project that you are going to use this file in your system.

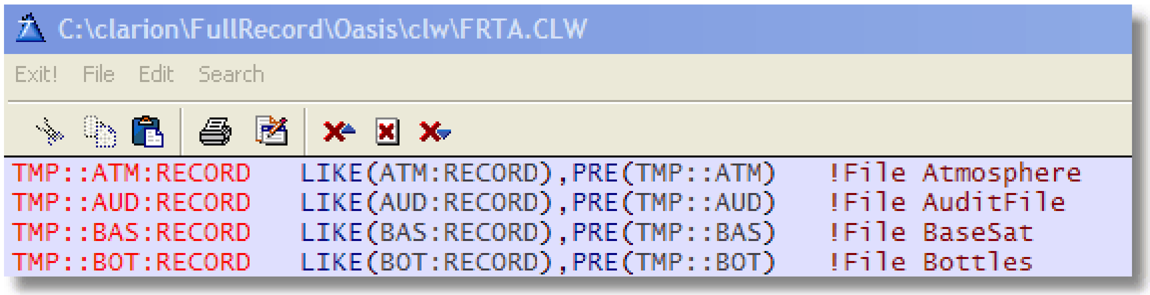
Which file is it?
When you see the error list, press the "Edit Errors" button.



```
Making frtabc0.obj  (frta.clw  changed)
Making frtabc0.clw
Sizes = Code: 3988; Const: 2778; Data: 95
No errors
Making frtabc.obj   (frta.clw  changed)
Making frtabc.clw
No errors
Sizes = Code: 25; Const: 0; Data: 2
Making frta.obj     (frta.clw  changed)
Making frta.clw
(frta.clw 395,24) Syntax error: Illegal parameter for LIKE
(frta.clw 395,36) Syntax error: Invalid data declaration attribute
(frta.clw 396,24) Syntax error: Illegal parameter for LIKE
(frta.clw 396,36) Syntax error: Invalid data declaration attribute
(frta.clw 397,24) Syntax error: Illegal parameter for LIKE
(frta.clw 397,36) Syntax error: Invalid data declaration attribute
(frta.clw 398,24) Syntax error: Illegal parameter for LIKE
(frta.clw 398,36) Syntax error: Invalid data declaration attribute
(frta.clw 399,25) Syntax error: Illegal parameter for LIKE
(frta.clw 399,38) Syntax error: Invalid data declaration attribute
```

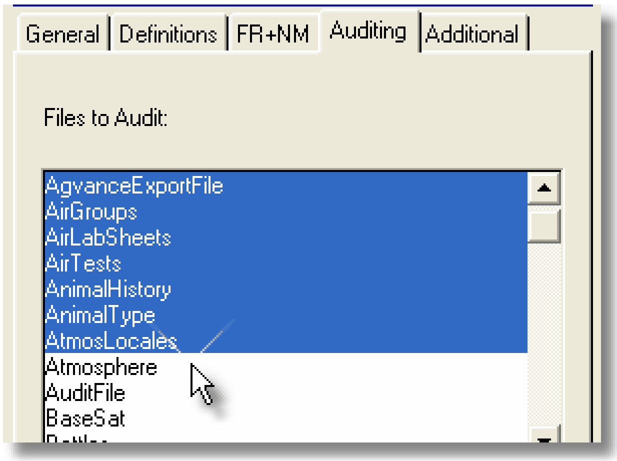
```
Make completed
550 errors
```

This takes you to the generated source code. You will see a list of LIKE declarations, and to the right, a comment with the name of the offending file.



```
TMP::ATM:RECORD    LIKE(ATM:RECORD),PRE(TMP::ATM)    !File Atmosphere
TMP::AUD:RECORD    LIKE(AUD:RECORD),PRE(TMP::AUD)    !File AuditFile
TMP::BAS:RECORD    LIKE(BAS:RECORD),PRE(TMP::BAS)    !File BaseSat
TMP::BOT:RECORD    LIKE(BOT:RECORD),PRE(TMP::BOT)    !File Bottles
```

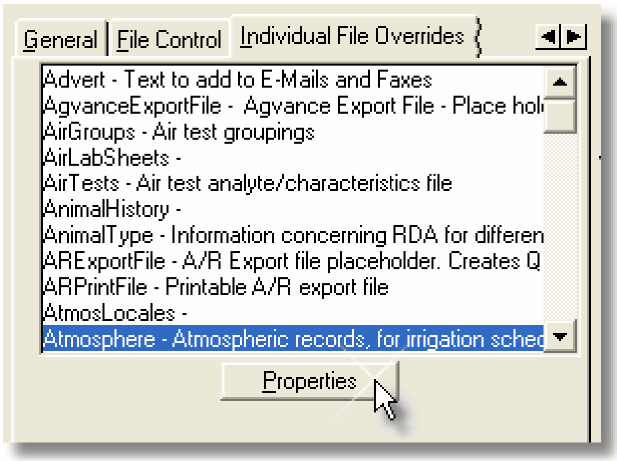
To remove this file from the list of audited files, go to Global Properties, Extensions, and in the "Auditing" tab, unmark the unwanted file.



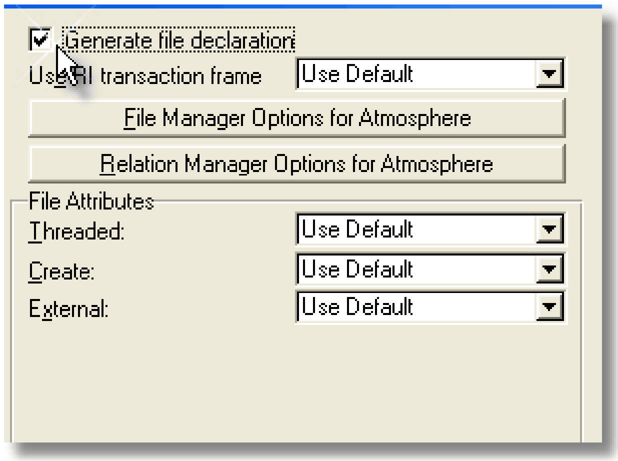
Files to Audit:

AgvanceExportFile
AirGroups
AirLabSheets
AirTests
AnimalHistory
AnimalType
AtmosLocales
Atmosphere
AuditFile
BaseSat

Alternatively, if you want to audit this file but you still are not using it in your system (or you are using it all by manual code), you may force the generation of this file declaration by entering the global properties, "Individual File Overrides". Entering the "Properties" button for this file, you may check "Generate file declaration" so the declaration is enforced in your app



Advert - Text to add to E-Mails and Faxes
AgvanceExportFile - Agvance Export File - Place hol
AirGroups - Air test groupings
AirLabSheets -
AirTests - Air test analyte/characteristics file
AnimalHistory -
AnimalType - Information concerning RDA for differen
ARExportFile - A/R Export file placeholder. Creates Q
ARPrintFile - Printable A/R export file
AtmosLocales -
Atmosphere - Atmospheric records, for irrigation sched


File Manager Options for Atmosphere
Relation Manager Options for Atmosphere
File Attributes

☑ Generate file declaration
Use RI transaction frame: Use Default
Threaded: Use Default
Create: Use Default
External: Use Default

12.4 Syntax error: Unknown identifier: FRGLO:PROCNAMEQUEUE

This error will appear if you are migrating your app from a FullRecord version previous to 1.6 and you forget to change the procedure variables in the dct (See Installation over previous versions).

12.5 The program closes unexpectedly or issues an error WGSDIAL 04

This may occur if you modify the definition of the AUDIT or MESSAGES files once you already created them, and if you didn't convert the physical files to the new definition, and you use NeatMessage in combination with FullRecord.
The normal behavior of a program when opening an invalid file is to show an error. But if the error is in one of these files, inside the message routine, then showing the message will make a recursive call to show the message about the error... entering a cycle until Windows resources are exhausted and the program closes (with or without GPF), or issue the Wgsdial 04 error.
You may either erase the AUDIT and MESSAGES files (if they don't have important information) or do the proper conversion from the old format to the new one.

12.6 You get duplicated entries in the audit file.

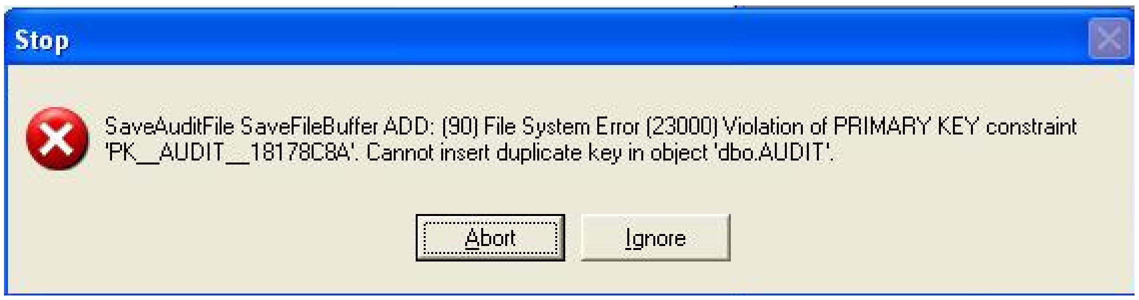
Symptom: When you record a single "change" operation, the audit file records a "read", a "change", a "read" and a "change" again (double reading and changing).
The problem is that your "Store Record" field in the Audit File and/or the global memory is not long enough to store the whole record. You should increase its size in the global data area, and it should have the same size in the audit file, the auditbck file and the auditnew file. Check the Definitions Tab chapter, and try the TXD Analyzer companion program to verify the maximum length of your records.

12.7 No data shows in the Inspect window

See the "Beware" note in the Inspect Tab chapter.

12.8 Violation of Primary Key Constrain

When you uncheck the "Audit file autonumbering" option to activate SQL server-side autonumbering of the audit file, and you forget to define the ID field as identity, you will get the following error (or something alike) as soon as you generate the second entry in the audit file:



SaveAuditFile SaveFileBuffer ADD: (90) File System Error (23000) Violation of PRIMARY KEY constraint 'PK__AUDIT__18178C8A'. Cannot insert duplicate key in object 'dbo.AUDIT'.

This is because being both auto-numbering methods disabled, the audit trail will be saved with an ID number of zero; thus on the second save it will be zero again, therefore, a duplicate entry.

12.9 The "view" control works, but the "recover" one don't

Things to check:

1) Does your file has a key with the PRIMARY attribute?
The file you are going to recover must have a key defined as PRIMARY in the DCT, otherwise it cannot determine on which key it should act to look for duplicates.

2) Did you fill the "Definitions 3" global tab?

12.10 Syntax error: No matching prototype available

This error may appear when you upgrade FullRecord and a procedure in use by FullRecord need new parameters. Check the Installation over previous versions chapter for specific notes on your current upgrade.

Part 13

13 Legals

13.1 Copyright and Limitations

FullRecord is a commercial product. Each developer needs his own license to use FullRecord. You may not distribute or copy the template FullRecord to other people. The details of the present agreement are in the End User License Agreement that you accepted when you installed this software.
Though the template is thoroughly tested, FullRecord is used entirely under your own risk. Jorge Alejandro Lavera, as the material author of the template, assumes no responsibility at all for the correct working of the applications that incorporates the FullRecord template. Hence, it is recommended that you do all the pertinent tests before you deliver a production version of a program with this template incorporated.
FullRecord Copyright © 2011 Jorge Alejandro Lavera.

13.2 Warranty, Registration and Technical Support.

FullRecord supposes to work as indicated in this manual, in any version of Clarion 6 or Clarion 7 with ABC or Legacy template chains. It was thoroughly tested with Clarion 6.3 9056 Enterprise and Professional Editions and Clarion 7 up to 7.3. If after reading the whole manual you still has problems, do not hesitate in contact to Jorge Alejandro Lavera at templates@larogroup.com or alternative at jlavera@gmail.com. We expect queries, suggestions and yes, even criticism.
Normally your ask for support messages will be answered between 24 working hs., from Monday to Friday, at least with a receipt. If you do not receive that answer, please insist using both e-mails or even from a different e-mail of your own, since e-mail today is not as trustable as it used to be, and the many spam filter mechanisms often delete perfectly valid messages.
Registered users may even obtain on-line support (ID Skype: jlavera) and warranty to obtain all future updates of the template for free and major upgrades on a case by case basis (At this time we cannot guarantee that all future upgrades will be free, but we also aren't saying that they'll be paid. We just do not know right now). You may give us suggestions to incorporate enhances to the template, as well as being informed of what are we doing with it in any moment.
Important: if we did not receive your purchase information from the distributor prior to your first contact, you may be delayed up to 24 hs. until your data is verified.
To register yourself the quickest way is to send an e-mail from your regular e-mail address to the e-mail addressed previously, stating your full name, date and place of purchase and you will be assigned a client number. Following communications from this same e-mail will be considered valid ones.

Index