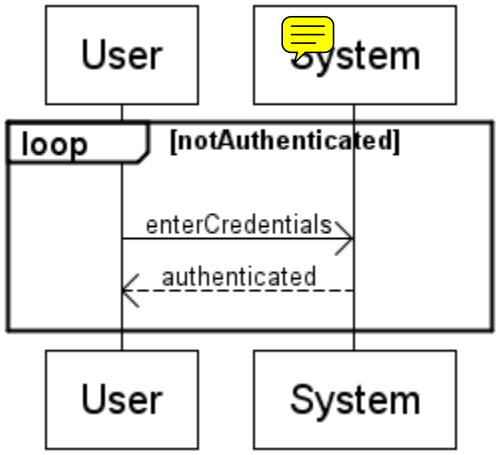
Milestone 3

Software Security API

Eric Crockett
Josh Dash
David Pick

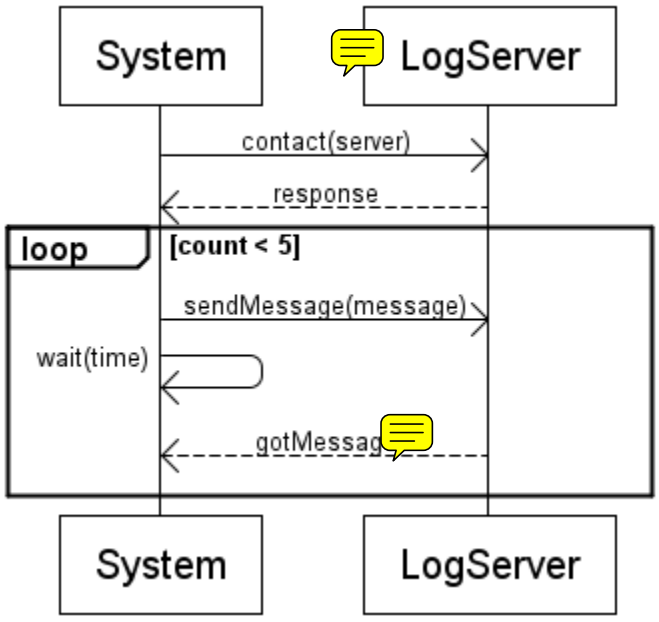
System Sequence Diagrams

Login SSD:



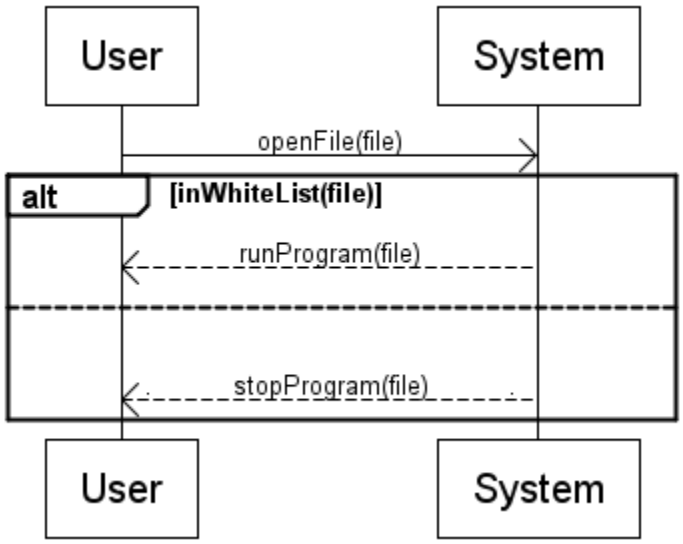
When a user attempts to log into the system the user sends a message to the system with their credentials, the system will then reply with whether their credentials were accepted or not.

Log Message SSD:



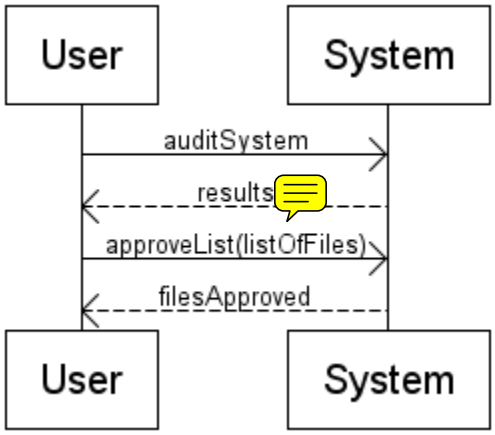
When the system needs to make a log message, it first tries to contact the remote sever, if it receives a response it then tries to send the message, if after a specific wait time no confirmation has been received from the server, the system will try again, up to five times.

Open File SSD:



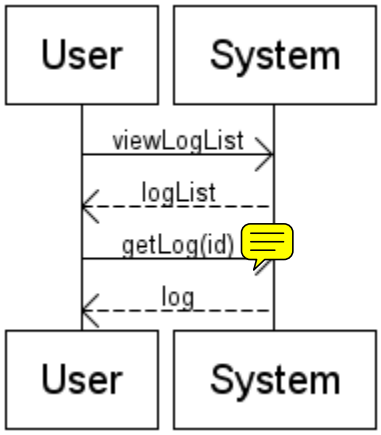
When a user tries to open a file it will first send an openFile request to the system through Windows. The system will then check to see if the file is in the whitelist, if it is, the system will run the program for the user, if it is blacklisted the program will be prevented from running.

Audit SSD:



When a user tries to perform an audit of the system, they first send an audit message to the system. The system then replies with the results of the audit, and a list of any new files that were found. This way the user can then tell the system which files should be whitelisted, by sending an approveList message. The user only needs to send files to be whitelisted, because the system already has a full list of files, and files that are not going to be whitelisted, are assume to be blacklisted.

View Log SSD:



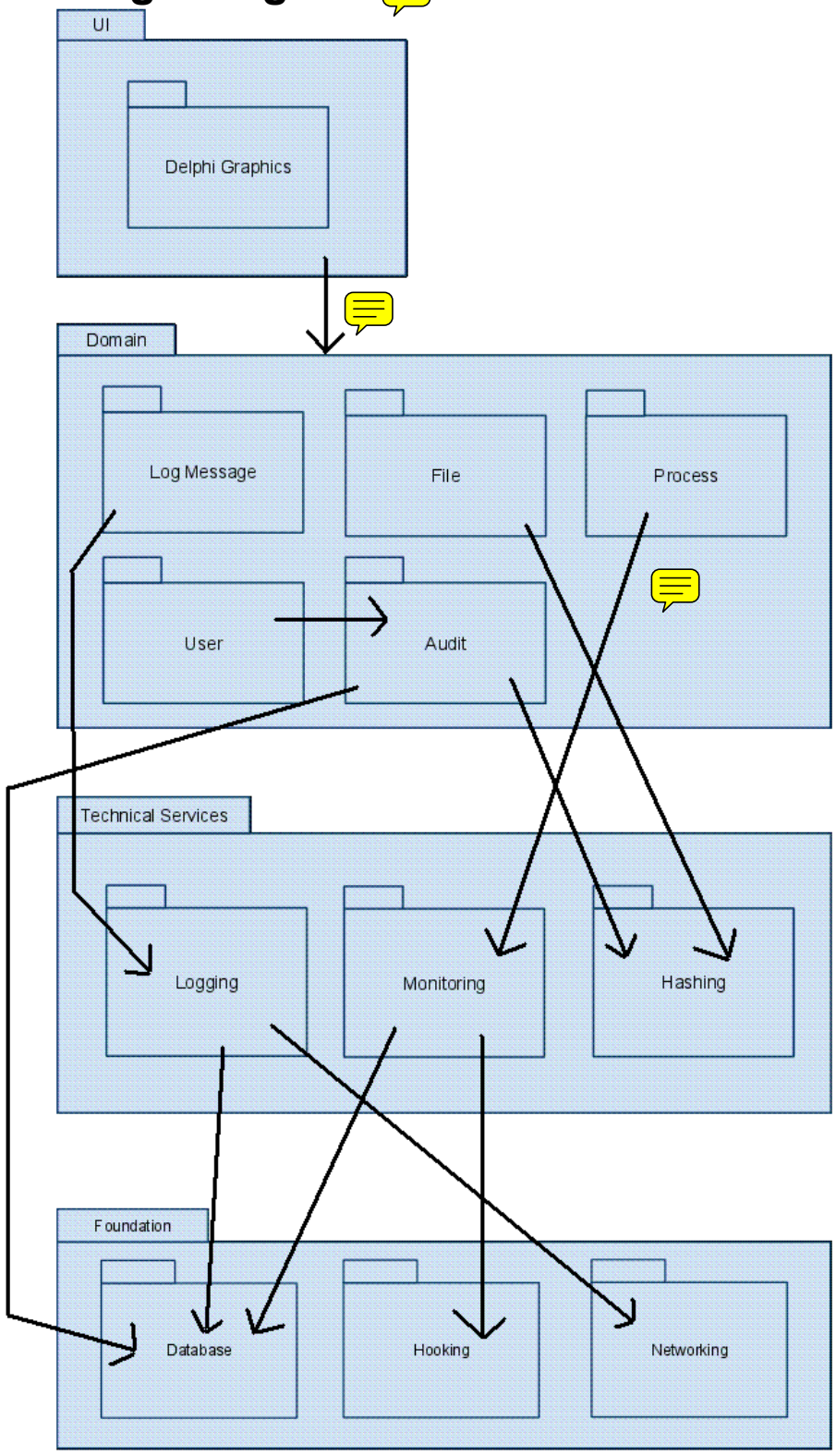
When a user would like to view a log, they first must query the system for a list of all of the logs. The system will reply with this list, when the user has decided which log message they would like to view, they send a message to the system with it's ID. The system will then reply with the correct log message.

Operation Contracts

Operation Contract for approveList
Operation: approveList(listOfFiles:FilePath)
Cross References: Audit SSD
Preconditions: Initial scan has been performed.
Postconditions:
For each file location in the list, a new File fn was created.
fn.location was set to listOfFiles[n].
fn.name was set to name of file.
fn.LastSolidified was set to current datetime.
fn.GUID was generated.
fn.LastAccessed was set to file's last access datetime.
fn.Instance was set to "local".
fn.hash was generated from the file at fn.location.
fn.size was recorded from the file at fn.location.
fn.whitelisted was set to true.
For each file location that was scanned but was not in the approved list, a new File fn was created.
For each file location in the list, a new File fn was created.
fn.location was set to listOfFiles[n].
fn.name was set to name of file.
fn.LastSolidified was set to current datetime.
fn.GUID was generated.
fn.LastAccessed was set to file's last access datetime.
fn.Instance was set to "local".
fn.hash was generated from the file at fn.location.
fn.size was recorded from the file at fn.location.
fn.whitelisted was set to false.

Operation Contract for stopProgram
Operation: stopProgram(file)
Cross References: OpenFile SSD
Preconditions:Monitor is running and user has opened a file that is not in the whitelist.
Postconditions:
A new Log_Message m was created.
m.GUID was generated.
m.classification was set to "execution denied".
m.severity was set to "important".
m.body was set to include the file name and location.
m.destination was set to external_server.
m.timestamp was set to current datetime.
m was added to the internal database.
m was sent to external_server.

Package Diagram



s diagram groups classes into packages and layers. The Foundation layer includes the basic technologies that the project relies on. Above that are the Technical Services which

provides the upper layers with an interface for the low level Foundation packages. The Domain layer contains packages that run the software and holds current system information in memory. The UI layer rests above everything else and allows the user to interact with the Domain layer to control the system.

The Foundation includes the Database, Hooking, and Networking packages. The Database holds the Accuracer database classes. Hooking consists of Madshi madCodeHook and is used to hook onto the Windows API to monitor processes. Networking contains components to send log messages to Tempus' servers.

The Technical Services layer holds the Logging package to save, access, and send log messages and the Monitoring package that utilizes the Database and Hooking packages to intercept and validate files running on the system. Logging needs the database to save messages to the local database log, and the network to send the log to Tempus' remote servers. The Monitoring package reads file validation from the database and monitors the system through the hooking service. Hashing provides a format for recording files.

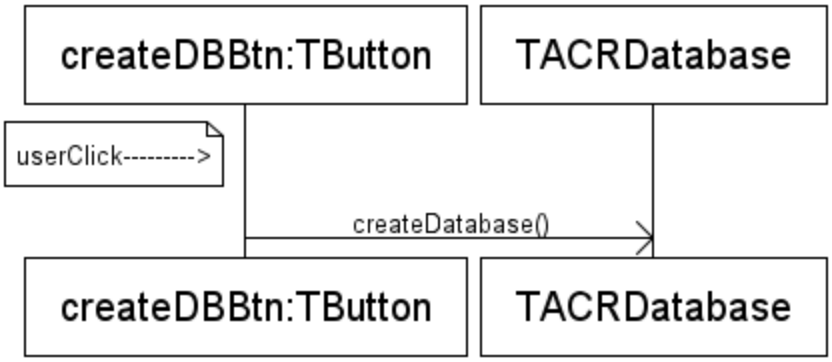
The Domain layer contains individual Log Messages that are recorded with the Logging package. An Audit package exists to periodically scan the entire system and allow new files to be allowed to run. The Audit system needs the Hashing service to store information in the Database. Files represent various files on the system and relies on Hashing in Technical Services to compute the hash. Likewise, Processes represent processes currently running on the system and need to be approved by the monitor. The User package exists to contain information about the current user of the system and allow them to update it as necessary. It needs to access Audit to start a periodic scan of the system.

The UI layer consists of various Delphi Graphics classes. It allows control of the system and needs access to various parts of the Domain layer to appropriately instruct the system.

Interaction Diagrams

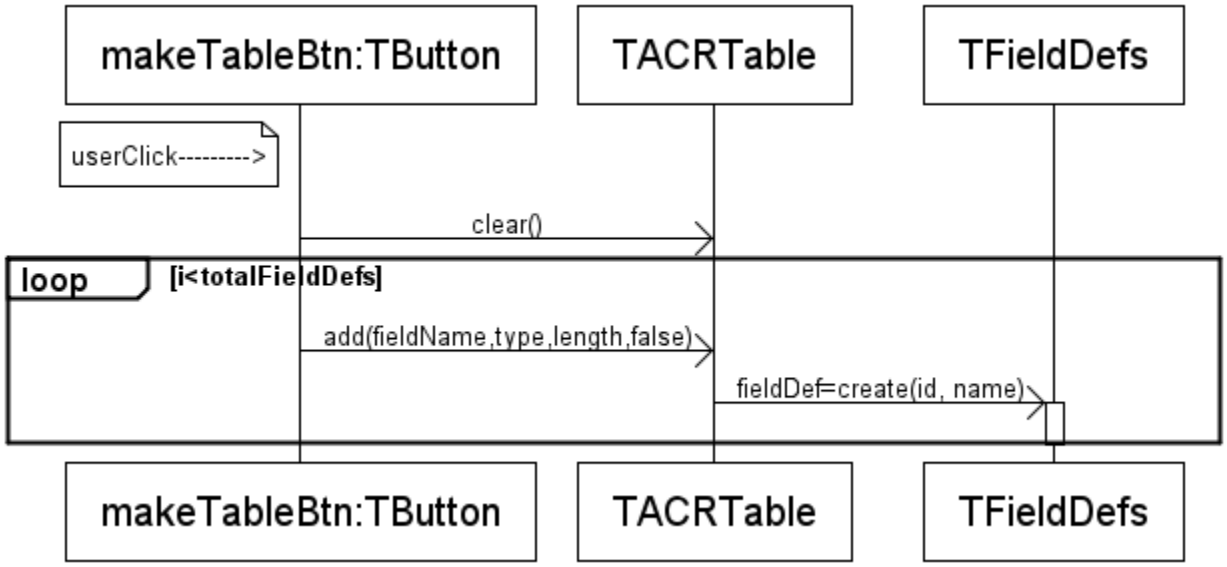
These diagrams detail the user interactions involved for the code developed in iteration 1.

Button createDBBtn sequence diagram:



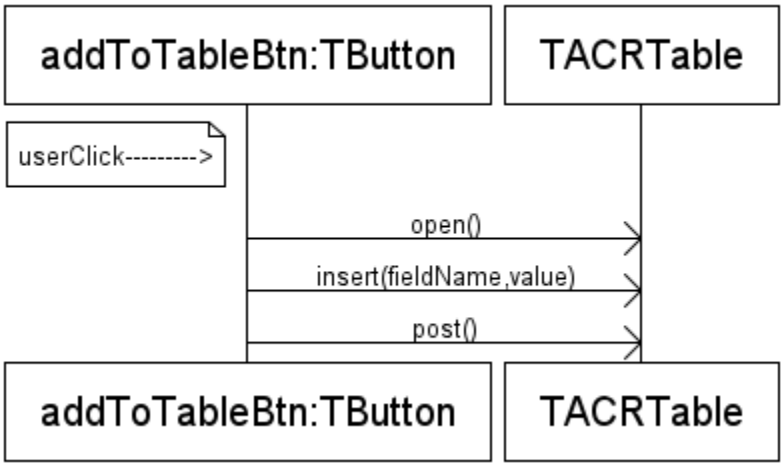
In Delphi in order to create a database, a button must be added to the UI, and then the database is set up using that button. The button sends a message to the database initializing it and creating the physical database on the computer.

Button makeTableBtn sequence diagram:



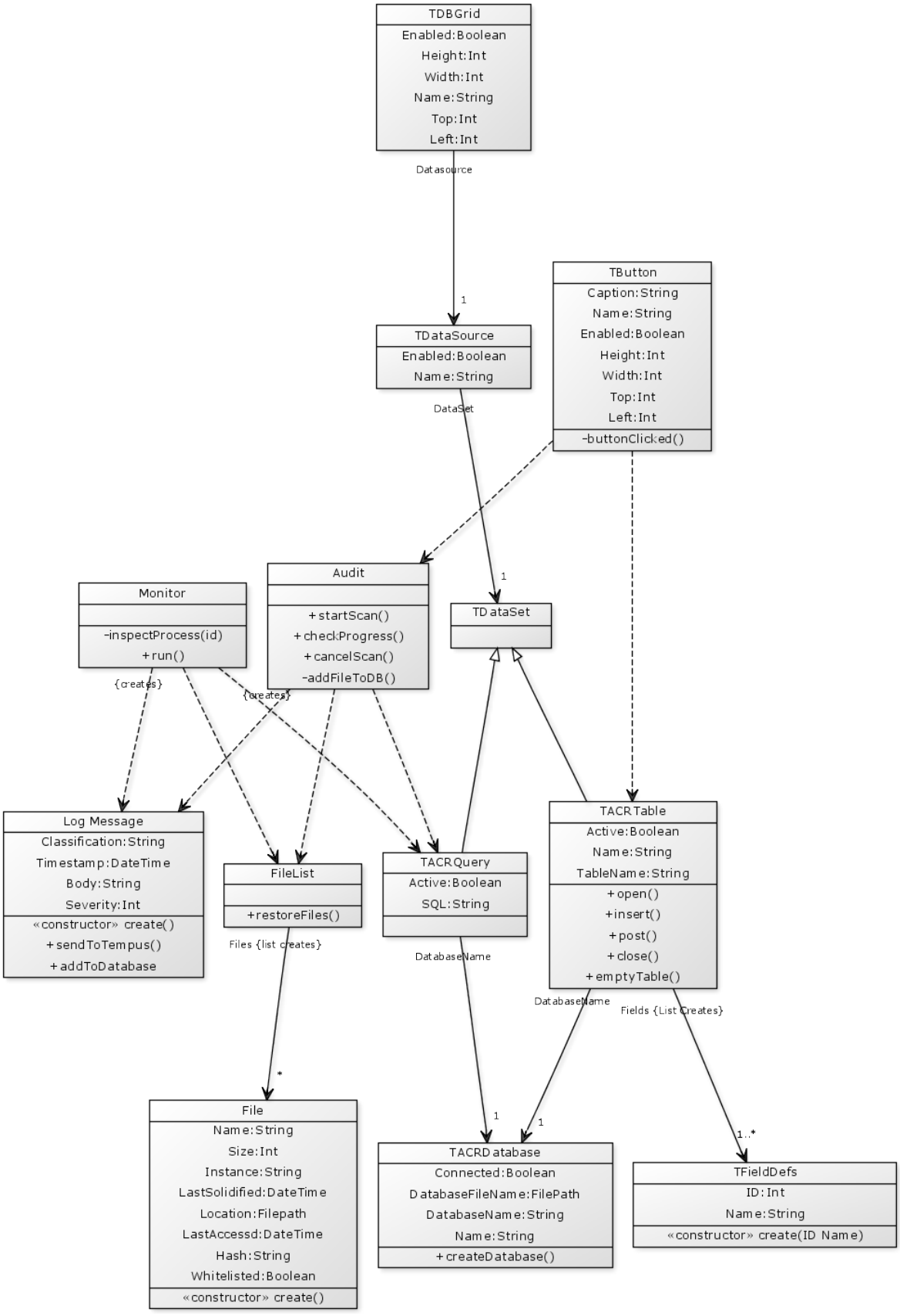
In Delphi in order to create a table with certain columns a button must be added to the UI. The button clears the table, then sends a message for each column to the table. As the table receives a message, it creates a TFieldDef for it, and the new TFieldDef is added to the table's list.

Button addToTableBtn:



To insert a record into a table, a button is added to the UI. When this button is clicked it sends an open message to the table. Then it creates a record with the value specified, only one field must be specified, and the others will be assumed to be null. A post message is then sent to the table, telling it to store the information in the database.

Design Class Diagram



This design class diagram describes the software perspective of the design. Some objects reflect the Delphi environment, such as a TButton and TDBGrid. Other objects are based on domain model concepts, such as Audit, Monitor, and File.
A TButton can start an audit, or access a TACRTable in the database.
Audit creates log messages as it performs a system scan, and also access the FileList and queries the database. The Audit is started by a TButton, and can be canceled by a TButton as well. Some other GUI component will check the progress of a scan, while the Audit itself will add files to the database (via a query) and FileList.
Monitor creates log messages as it protects the system, and also access the FileList and queries the database. The Monitor is started by Delphi, it exists when the program starts. It accesses the database via a query to update the fields of files as they execute (such as lastAccessed), and it uses the FileList instead of the database for reading data to boost performance.
FileList is a list of files, which may be either whitelisted or blacklisted. This list mirrors the File table of the database, but is kept as an object for performance purposes.
Monitor is tasked with repopultating FileList from the database, but it does not <<create>> FileList; Delphi handles the creation of some objects.
TACRTables creates a list of TFieldDefs which become its fields in the database. It also contains a field which indicates the database to which it belongs.