



PROJECT DOCUMENTATION

Project Name:	Synapse mORMot Framework
Document Name:	Software Design Document
Document Revision:	1.17
Date:	September 9, 2012
Project Manager:	Arnaud Bouchez

Document License

THE ATTACHED DOCUMENTS DESCRIBE INFORMATION RELEASED BY SYNOPSIS INFORMATIQUE UNDER A GPL 3.0 LICENSE.

Synapse SQLite3/mORMot Framework Documentation.

Copyright (C) 2008-2012 Arnaud Bouchez.

Synapse Informatique - <http://synapse.info..>

This document is free document; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

The *Synapse mORMot Framework Documentation* is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this documentation. If not, see <http://www.gnu.org/licenses..>

Trademark Notice

Rather than indicating every occurrence of a trademarked name as such, this document uses the names only in an editorial fashion and to the benefit of the trademark owner with no intention of infringement of the trademark.

Prepared by:	Title:	Signature:	Date
Arnaud Bouchez	Project Manager		

Document Purpose

The *Software Design Document* document purpose is to summarize the software DI implementation for QA review for the *Synapse mORMot Framework* project.

The current revision of this document is 1.17.

Related Documents

Name	Description	Rev.	Date
DI	Design Input Product Specifications	1.17	September 9, 2012
SWRS	Software Requirements Specifications	1.17	September 9, 2012
SAD	Software Architecture Design	1.17	September 9, 2012

Table of Contents

Introduction

Documentation overview	11
Purpose	11
Responsibilities	12
GNU General Public License	12
Software Design Document Reference Table	23

1. Client Server JSON framework

1.1. SWRS # DI-2.1.1	25
1.1.1. Abstract	25
1.1.2. Implementation	25
1.2. SWRS # DI-2.1.1.1	25
1.2.1. Abstract	25
1.2.2. Implementation	26
1.2.2.1. Server-Side	26
1.2.2.2. Client-Side	26
1.3. SWRS # DI-2.1.1.2	26
1.4. SWRS # DI-2.1.1.2.1	27
1.4.1. Abstract	27
1.4.2. Implementation	27
1.4.2.1. Server-Side	27
1.4.2.2. Client-Side	27

1.5. SWRS # DI-2.1.1.2.2	27
1.5.1. Abstract	27
1.5.2. Implementation	27
1.5.2.1. Server-Side	27
1.5.2.2. Client-Side	28
1.6. SWRS # DI-2.1.1.2.3	28
1.6.1. Abstract	28
1.6.2. Implementation	28
1.6.2.1. Server-Side	28
1.6.2.2. Client-Side	28
1.7. SWRS # DI-2.1.1.2.4	28
1.7.1. Abstract	28
1.7.2. Implementation	29
1.7.2.1. Server-Side	29
1.7.2.2. Client-Side	29
1.8. SWRS # DI-2.1.2	29
1.8.1. Abstract	29
1.8.2. Implementation	30
1.8.2.1. JSON-dedicated functions and classes	30
1.8.2.2. Database record level	31
1.8.2.3. Database request table level	31
1.8.2.4. Fast JSON parsing	31
1.9. SWRS # DI-2.1.3	33
1.9.1. Abstract	33
1.9.2. Implementation	33
1.9.3. TSQLRecord table properties	34
1.9.3.1. Per-class variable needed	34
1.9.3.2. Patching a running process code	35
1.9.3.3. Per-class variable in the VMT	35

2. SQLite3 engine

2.1. SWRS # DI-2.2.1	38
2.1.1. Abstract	38
2.1.2. Implementation	38
2.1.2.1. Low-Level access to the library	38
2.1.2.1.1. Compilation of the SQLite3 engine	38
2.1.2.1.2. SQLite3 API access	40
2.1.2.2. High level access	40
2.2. SWRS # DI-2.2.2	40
2.2.1. Abstract	40
2.2.2. Implementation	41
2.3. SWRS # DI-2.2.3	41
2.3.1. Abstract	41
2.3.2. SynDB classes	42
2.3.3. Faster late binding	42
2.3.3.1. Speed issue	42
2.3.3.2. Fast and furious	42
2.3.3.3. Implementation	42
2.3.3.4. Hacking the VCL	44
3. User interface	
<hr/>	
3.1. SWRS # DI-2.3	46
3.1.1. Abstract	46
3.1.2. SynFile main Demo	47
3.2. SWRS # DI-2.3.1.1	47
3.2.1. Abstract	47
3.2.2. Implementation	48
3.3. SWRS # DI-2.3.1.2	48
3.3.1. Abstract	48
3.3.2. Implementation	48
3.3.2.1. Rendering	49
3.3.2.2. Ribbon-like toolbars	49

3.3.2.3. Stand-alone toolbars	49
3.4. SWRS # DI-2.3.1.3	49
3.4.1. Abstract	49
3.4.2. Implementation	50
3.5. SWRS # DI-2.3.2	50
3.5.1. Abstract	50
3.5.2. Implementation	50

Pictures Reference Table

The following table is a quick-reference guide to all the Pictures referenced in this *Software Design Document* (SDD) document.

Pictures	Page
Design Inputs, FMEA and Risk Specifications	11
TSynTestCase classes hierarchy	41
User Interface generated using TMS components	46
User Interface generated using VCL components	47

Source code File Names Reference Table

The following table is a quick-reference guide to all the Source code File Names referenced in this *Software Design Document* (SDD) document.

Others - Source Reference Table

Source code File Names	Page
Lib\SQLite3\SQLite3.pas	38, 41
Lib\SQLite3\SQLite3Commons.pas	25, 26, 27, 28, 28, 29, 30, 34, 38, 41
Lib\SQLite3\SQLite3HttpClient.pas	29
Lib\SQLite3\SQLite3HttpServer.pas	29, 41
Lib\SQLite3\SQLite3i18n.pas	50
Lib\SQLite3\SQLite3Pages.pas	50
Lib\SQLite3\SQLite3ToolBar.pas	48, 49
Lib\SQLite3\SQLite3UI.pas	48
Lib\SynCommons.pas	30, 41
Lib\SynCrtSock.pas	29
Lib\SynGdiPlus.pas	50
Lib\SynPdf.pas	50
Lib\SynSQLite3.pas	38

Keywords Reference Table

The following table is a quick-reference guide to all the Keywords referenced in this *Software Design Document* (SDD) document.

6

64 bit	34
--------	----

C

Camel	49
Client-Server	25

E

Enumerate	49
-----------	----

I

ISO 8601	48
----------	----

O

ORM	25, 34, 37, 40, 48
-----	--------------------

R

REST	26, 40
RTTI	34, 34, 34, 48, 49

S

Service	30
SQL	40

T

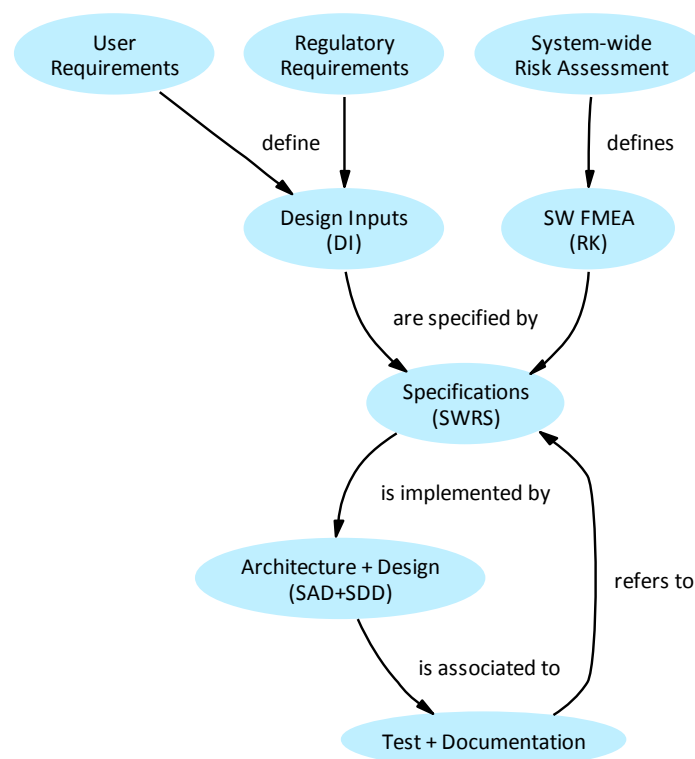
TCreateTime	48
TDateTime	48

Test	41
TModTime	48
TTimeLog	48

Introduction

Documentation overview

The whole Software documentation process follows the typical steps of this diagram:



Design Inputs, FMEA and Risk Specifications

Purpose

This *Software Design Document (SDD)* document applies to the release of the *Synopse mORMot Framework* library.

It summarizes the software implementation of each design input as specified by the *Design Input Product Specifications (DI)* document.

This document is divided into the main parts of the Software implementation:

- Client Server JSON framework (page 25)
- SQLite3 engine (page 38)
- User interface (page 46)

Inside this sections, source code or User Interface modifications are detailed for every *Software Requirements Specifications* (SWRS) document item.

Responsibilities

- Synopse will try to correct any identified issue;
- The Open Source community will create tickets in a public Tracker web site located at <http://synopse.info/fossil..> ;
- Synopse work on the framework is distributed without any warranty, according to the chosen license terms;
- This documentation is released under the GPL (GNU General Public License) terms, without any warranty of any kind.

GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts,

regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided

you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this license along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this license, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this license, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see [<http://www.gnu.org/licenses/>](http://www.gnu.org/licenses/).

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read [<http://www.gnu.org/philosophy/why-not-lgpl.html>](http://www.gnu.org/philosophy/why-not-lgpl.html).

Software Design Document Reference Table

The following table is a quick-reference guide to all the *Software Requirements Specifications* (SWRS) document items.

SWRS #	Description	Page
DI-2.1.1	The framework shall be Client-Server oriented	25
DI-2.1.1.1	A RESTful mechanism shall be implemented	25
DI-2.1.1.2	Communication should be available directly in the same process memory, or remotely using Named Pipes, Windows messages or HTTP/1.1 protocols	26
DI-2.1.1.2.1	Client-Server Direct communication shall be available inside the same process	27
DI-2.1.1.2.2	Client-Server Named Pipe communication shall be made available by some dedicated classes	27
DI-2.1.1.2.3	Client-Server Windows GDI Messages communication shall be made available by some dedicated classes	28
DI-2.1.1.2.4	Client-Server HTTP/1.1 over TCP/IP protocol communication shall be made available by some dedicated classes, and ready to be accessed from outside any Delphi Client (e.g. the implement should be AJAX ready)	28
DI-2.1.2	UTF-8 JSON format shall be used to communicate	29
DI-2.1.3	The framework shall use an innovative ORM (Object-relational mapping) approach, based on classes RTTI (Runtime Type Information)	33
DI-2.2.1	The <i>SQLite3</i> engine shall be embedded to the framework	38
DI-2.2.2	The framework libraries, including all its <i>SQLite3</i> related features, shall be tested using Unitary testing	40
DI-2.2.3	The framework shall be able to access any external database, via OleDb, ODBC or direct access for Oracle (OCI) or <i>SQLite3</i> (for external database files)	41
DI-2.3	User Interface and Report generation should be integrated	46

SWRS #	Description	Page
DI-2.3.1.1	A Database Grid shall be made available to provide data browsing in the Client Application - it shall handle easy browsing, by column resizing and sorting, on the fly customization of the cell content	47
DI-2.3.1.2	Toolbars shall be able to be created from code, using RTTI and enumerations types for defining the action	48
DI-2.3.1.3	Internationalization (i18n) of the whole User Interface shall be made available by defined some external text files: Delphi resourcestring shall be translatable on the fly, custom window dialogs automatically translated before their display, and User Interface generated from RTTI should be included in this i18n mechanism	49
DI-2.3.2	A reporting feature, with full preview and export as PDF or TXT files, shall be integrated	50

1. Client Server JSON framework

1.1. SWRS # DI-2.1.1

The framework shall be Client-Server oriented

1.1.1. Abstract

Design Input 2.1.1 (Initial release): The framework shall be Client-Server oriented.

Client-Server model of computing is a distributed application structure that partitions tasks or workloads between service providers, called servers, and service requesters, called clients.

Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more server programs which share its resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await (listen for) incoming requests.

The *Synopse mORMot Framework* shall implement such a Client-Server model by a set of dedicated classes, over various communication protocols, but in a unified way. Application shall easily change the protocol used, just by adjusting the class type used in the client code. By design, the only requirement is that protocols and associated parameters are expected to match between the Client and the Server.

1.1.2. Implementation

The Client-Server aspect of the framework is implemented in the `SQLite3Commons.pas` unit, with the `TSQLRestServer` and `TSQLRestClientURI` classes.

Both classes inherit from a generic `TSQLRest` class, which implements some generic database access methods and properties (through ORM model for objects descending from `TSQLRecord` or table-based query using `TSQLTableJSON`).

1.2. SWRS # DI-2.1.1.1

A RESTful mechanism shall be implemented

1.2.1. Abstract

Design Input 2.1.1.1 (Initial release): A RESTful mechanism shall be implemented.

REST-style architectures consist of clients and servers, as was stated in *SWRS # DI-2.1.1*. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of "representations" of "resources". A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource.

In the *Synapse mORMot Framework*, so called "resources" are individual records of the underlying database, or list of individual fields values extracted from these databases, by a SQL-like query statement.

1.2.2. Implementation

The RESTful mechanism is implemented using the URI method of both `TSQLRestServer` and `TSQLRestClientURI` classes, as defined in the `SQLite3Commons.pas` unit.

1.2.2.1. Server-Side

In the `TSQLRestServer` class, the URI method is defined as public, and must implement the actual database query or update, according to the REST request:

```
function TSQLRestServer.URI(const url, method: RawUTF8; const SentData: RawUTF8;  
    out Resp, Head: RawUTF8; const RestAccessRights: TSQLAccessRights): Int64Rec;
```

The purpose of this method is to:

- Return internal database state count (used for caching);
- Retrieve URI expecting the RESTful 'ModelRoot[/TableName[/ID[/BlobFieldName]]]' format;
- Call appropriate database commands, by using the protected `EngineList EngineRetrieve EngineAdd EngineUpdate EngineDelete EngineRetrieveBlob EngineUpdateBlob` methods.

The `TSQLRestServer` class itself doesn't implement these database command methods: they are all defined as `virtual; abstract`. Children classes must override these virtual methods, and implement them using the corresponding database engine.

1.2.2.2. Client-Side

In the `TSQLRestClientURI` class, the URI method is defined as protected and as `virtual; abstract`. Children classes must override this method, and implement the remote database query or update, according to the REST request, and its internal protocol.

1.3. SWRS # DI-2.1.1.2

Communication should be available directly in the same process memory, or remotly using Named Pipes, Windows messages or HTTP/1.1 protocols

Design Input 2.1.1.2 (Initial release): Communication should be available directly in the same process memory, or remotly using Named Pipes, Windows messages or HTTP/1.1 protocols.

In computing and telecommunications, a protocol or communications protocol is a formal description of message formats and the rules for exchanging those messages.

The *Synapse mORMot Framework* shall support the following protocols for remote access, according to the Client-Server architecture defined in *SWRS # DI-2.1.1*:

- Direct in-process communication;
- Using GDI messages;
- Using Named pipe;
- Using HTTP/1.1 over TCP/IP.

1.4. SWRS # DI-2.1.1.2.1

Client-Server Direct communication shall be available inside the same process

1.4.1. Abstract

Client-Server Direct communication shall be available inside the same process.

1.4.2. Implementation

The in-process communication is implemented by using a global function, named `URIRequest` and defined in `SQLite3Commons.pas`:

```
function URIRequest(url, method, SendData: PUTF8Char; Resp, Head: PPUTF8Char): Int64Rec; cdecl;
```

1.4.2.1. Server-Side

This function can be exported from a DLL to remotely access to a `TSQLRestServer`, or used in the same process:

- Use `TSQLRestServer.ExportServer` to assign a server to this function;
- Return *501 NOT IMPLEMENTED* error if no `TSQLRestServer.ExportServer` has been assigned yet;
- Memory for `Resp` and `Head` parameters are allocated with `GlobalAlloc()` Win32 API function: client must release this pointers with `GlobalFree()` after having retrieved their content - you can force using the Delphi heap (and `GetMem` function which is much faster than `GlobalAlloc`) by setting the `USEFASTMM4ALLOC` variable to `TRUE`: in this case, client must release this pointers with `FreeMem()`.

1.4.2.2. Client-Side

The Client should simply use a `TSQLRestClientURIDll` instance to access to an exported `URIRequest()` function.

1.5. SWRS # DI-2.1.1.2.2

Client-Server Named Pipe communication shall be made available by some dedicated classes

1.5.1. Abstract

Client-Server Named Pipe communication shall be made available by some dedicated classes.

1.5.2. Implementation

1.5.2.1. Server-Side

The communication is implemented by using the `TSQLRestServer` class, defined in `SQLite3Commons.pas`.

This class implements a server over Named Pipe communication, when its `ExportServerNamedPipe` method is called.

1.5.2.2. Client-Side

A dedicated `TSQLRestClientURINamedPipe` class has been defined. It inherits from `TSQLRestClientURI`, and override its `URI` protected method so that it communicates using a specified Named Pipe.

1.6. SWRS # DI-2.1.1.2.3

Client-Server Windows GDI Messages communication shall be made available by some dedicated classes

1.6.1. Abstract

Client-Server Windows GDI Messages communication shall be made available by some dedicated classes.

1.6.2. Implementation

Communication using Win32 GDI messages is very handy and efficient on the same computer. It's also perfectly safe, because, by design, it can't be access remotely. Performances for small messages is also excellent. Named pipe could be faster only when bigger messages are transmitted.

1.6.2.1. Server-Side

The communication is implemented by using the `TSQLRestServer` class, defined in `SQLite3Commons.pas`.

This class implements a server over Win32 GDI messages communication, when its `ExportServerMessage` method is called.

1.6.2.2. Client-Side

A dedicated `TSQLRestClientURIMessage` class has been defined. It inherits from `TSQLRestClientURI`, and override its `URI` protected method so that it communicates using Win32 GDI messages.

1.7. SWRS # DI-2.1.1.2.4

Client-Server HTTP/1.1 over TCP/IP protocol communication shall be made available by some dedicated classes, and ready to be accessed from outside any Delphi Client (e.g. the implement should be AJAX ready)

1.7.1. Abstract

Client-Server HTTP/1.1 over TCP/IP protocol communication shall be made available by some dedicated classes, and ready to be accessed from outside any Delphi Client (e.g. the implement should be AJAX ready).

1.7.2. Implementation

1.7.2.1. Server-Side

The communication is not implemented directly in the `TSQLRestServer` class, defined in `SQLite3Commons.pas`, but by a dedicated `TSQLite3HttpServer` class defined in `SQLite3HttpServer.pas`.

This class will instantiate a `THttpServerGeneric` instance, defined in `SynCrtSock.pas`, which implements a HTTP/1.1 server over TCP/IP communication.

This server is implemented either:

- Via `THttpApiServer` for using the fast kernel-mode `http.sys` server;
- Via `THttpServer`, which is an optimized pure Delphi HTTP/1.1 compliant server, using *Thread pool* to reduce resources, and provide best possible performance in user land.

You can register several `TSQLRestServer` instance to the same HTTP server, via its `AddServer` method. Each `TSQLRestServer` class must have an unique `Model.Root` value, to identify which instance must handle a particular request from its URI root string.

A dedicated property, named `DBServer`, is an array to all registered `TSQLRestServer` instances, which are used to process any request, and answer to it by using the corresponding URI method - via the `OnRequest` standard event prototype.

1.7.2.2. Client-Side

A dedicated `TSQLite3HttpClient` class has been defined in `SQLite3HttpClient.pas`. It inherits from `TSQLRestClientURI`, and override its URI protected method so that it communicates using HTTP/1.1 protocol over TCP/IP, according to the supplied HTTP address name.

By default, `TSQLite3HttpClient` maps to a `TSQLite3HttpClientWinHTTP` class, which was found out to perform well on most configurations and networks (whereas `TSQLite3HttpClientWinSock` should be a bit faster on a local computer).

1.8. SWRS # DI-2.1.2

UTF-8 JSON format shall be used to communicate

1.8.1. Abstract

Design Input 2.1.2 (Initial release): UTF-8 JSON format shall be used to communicate.

JSON, as defined in the *Software Architecture Design (SAD)* document, is used in the *Synapse mORMot Framework* for all Client-Server communication. JSON (an acronym for JavaScript Object Notation) is a lightweight text-based open standard designed for human-readable data interchange. Despite its relationship to JavaScript, it is language-independent, with parsers available for virtually every programming language.

JSON shall be used in the framework for returning individual database record content, in a disposition

which could make it compatible with direct JavaScript interpretation (i.e. easily creating JavaScript object from JSON content, in order to facilitate AJAX application development). From the Client to the Server, record content is also JSON-encoded, in order to be easily interpreted by the Server, which will convert the supplied field values into proper SQL content, ready to be inserted to the underlying database.

JSON should be used also within the transmission of request rows of data. It therefore provide an easy way of data forming between the Client and the Server.

The *Synapse mORMot Framework* shall use UTF-8 encoding for the character transmission inside its JSON content. UTF-8 (8-bit Unicode Transformation Format) is a variable-length character encoding for Unicode. UTF-8 encodes each character (code point) in 1 to 4 octets (8-bit bytes). The first 128 characters of the Unicode character set (which correspond directly to the ASCII) use a single octet with the same binary value as in ASCII. Therefore, UTF-8 can encode any Unicode character, avoiding the need to figure out and set a "code page" or otherwise indicate what character set is in use, and allowing output in multiple languages at the same time. For many languages there has been more than one single-byte encoding in usage, so even knowing the language was insufficient information to display it correctly.

1.8.2. Implementation

The JSON parsing and producing is implemented in the `SynCommons.pas` and `SQLite3Commons.pas` units.

The JSON encoding and decoding is handled at diverse levels:

- With some JSON-dedicated functions and classes;
- At the database record level;
- At the database request table level.

1.8.2.1. JSON-dedicated functions and classes

The main class for producing JSON content is `TJSONWriter`. This class is a simple writer to a Stream, specialized for the JSON format. Since it makes

use of an internal buffer, and avoid most temporary string allocation (e.g. using the stack instead of a temporary string via `IntToStr()` when converting a numerical value to text), it is much faster than a string append (standard Delphi string `:= string+string` clauses) to produce its content. In particular, its `AddJSONEscape` method will handle JSON content escape, according to the official JSON RFC - see <http://www.ietf.org/rfc/rfc4627.txt>, paragraph 2.5, directly into the destination buffer. It was also designed to scales well on multi-core sytems.

Some JSON-dedicated function are also available:

- `GetJSONObjectAsSQL` decodes a JSON fields object into an UTF-8 encoded SQL-ready statement;
- `IsJSONString` returns TRUE if the supplied content must be encoded as a JSON string according to the JSON encoding schema, i.e. if it's some null/false/true content or any pure numerical data (integer or floating point);
- `UnJSONFirstField` can be used to retrieve the FIRST field value of the FIRST row, from a JSON content: it may be useful to get an ID without converting the whole JSON content into a `TSQLTableJSON`;
- `JSONEncode` and `JSONDecode` functions are available to directly encode or decode some UTF-8 JSON content (used in the remote Service implementation, for instance).

1.8.2.2. Database record level

The TJSONWriter class (based on TTextWriter) is used by the GetJSONValues method of the TSQLRecord class to get all the data of a database record as JSON content.

Here is an extract of the main loop of this method:

```
procedure TSQLTable.GetJSONValues(JSON: TStream; Expand: boolean;
RowFirst: integer=0; RowLast: integer=0);
(...)
  for R := RowFirst to RowLast do
  begin
    if Expand then
      W.Add('{');
    for F := 0 to FieldCount-1 do
    begin
      if Expand then
        W.AddString(W.ColNames[F]); // '''+ColNames[...]+''': '
      if Assigned(QueryTables) then
        if IsJSONString(U^) then
          begin
            W.Add('');
            W.AddJSONEscape(U^,0);
            W.Add('');
          end else
            W.AddNoJSONEscape(U^,0);
      W.Add(',');
      inc(U); // points to next value
    end;
    W.CancellLastComma; // cancel last ','
    if Expand then
      W.Add('}');
    W.Add(',');
  end;
  (...)
```

1.8.2.3. Database request table level

Most high-level Client-sided list request methods returns a TSQLTableJSON instance as a result. This TSQLTableJSON class has been created from a pure JSON content, retrieved from the Server using one of the protocols defined in *SWRS # DI-2.1.1.2*.

Its Create constructor method call its internal protected method named FillFrom(), which make the JSON conversion into pure UTF-8 text fields, as expected by the TSQLTable class and its various Get*() methods. The FillFrom() method implements a very fast parsing of the supplied JSON content, then un-escape its content according to the JSON RFC quoted above.

1.8.2.4. Fast JSON parsing

When it deals with parsing some (textual) content, two directions are usually envisaged. In the XML world, you have usually to make a choice between:

- A DOM parser, which creates an in-memory tree structure of objects mapping the XML nodes;
- A SAX parser, which reads the XML content, then call pre-defined *events* for each XML content element.

In fact, DOM parsers use internally a SAX parser to read the XML content. Therefore, with the overhead of object creation and their property initialization, DOM parsers are typically three to five times slower than SAX. But, DOM parsers are much more powerful for handling the data: as soon as it's mapped in native objects, code can access with no time to any given node, whereas a SAX-based

access will have to read again the whole XML content.

Most JSON parser available in Delphi use a DOM-like approach. For instance, the *DBXJSON* unit included since Delphi 2010 or the *SuperObject* library create a class instance mapping each JSON node.

In a JSON-based Client-Server ORM like ours, profiling shows that a lot of time is spent in JSON parsing, on both Client and Server side. Therefore, we tried to optimize this part of the library.

In order to achieve best speed, we try to use a mixed approach:

- All the necessary conversion (e.g. un-escape text) is made in-memory, from and within the JSON buffer, to avoid memory allocation;
- The parser returns *pointers* to the converted elements (just like the *vtd-xml* library).

In practice, here is how it is implemented:

- A private copy of the source JSON data is made internally (so that the Client-Side method used to retrieve this data can safely free all allocated memory);
- The source JSON data is parsed, and replaced by the UTF-8 text un-escaped content, in the same internal buffer (for example, strings are un-escaped and #0 are added at the end of any field value; and numerical values remains text-encoded in place, and will be extracted into Int64 or double only if needed);
- Since data is replaced in-memory (JSON data is a bit more verbose than pure UTF-8 text so we have enough space), no memory allocation is performed during the parsing: the whole process is very fast, not noticeably slower than a SAX approach;
- This very profiled code (using pointers and tuned code) results in a very fast parsing and conversion.

This parsing "magic" is done in the `GetJSONField` function, as defined in the `SynCommons.pas` unit:

```
/// decode a JSON field in an UTF-8 encoded buffer (used in TSQLTableJSON.Create)
// - this function decodes in the P^ buffer memory itself (no memory allocation
// or copy), for faster process - so take care that it's an unique string
// - PDest points to the next field to be decoded, or nil on any unexpected end
// - null is decoded as nil
// - "strings" are decoded as 'strings'
// - strings are JSON unescaped (and \u0123 is converted to UTF-8 chars)
// - any integer value is left as its ascii representation
// - wasString is set to true if the JSON value was a "string"
// - works for both field names or values (e.g. '"FieldName":' or 'Value,')
// - EndOfObject (if not nil) is set to the JSON value char (',' ':' or '}' e.g.)
function GetJSONField(P: PUTF8Char; out PDest: PUTF8Char;
  wasString: PBoolean=nil; EndOfObject: PUTF8Char=nil): PUTF8Char;
```

This function allows to iterate throughout the whole JSON buffer content, retrieving values or property names, and checking `EndOfObject` returning value to handle the JSON structure.

This in-place parsing of textual content is one of the main reason why we used UTF-8 (via `RawUTF8`) as the common string type in our framework, and not the generic string type, which would have introduced a memory allocation and a char-set conversion.

For instance, here is how JSON content is converted into SQL, as fast as possible:

```
function GetJSONObjectAsSQL(var P: PUTF8Char; const Fields: TRawUTF8DynArray;
  Update, InlinedParams: boolean): RawUTF8;
(...)
  // get "COL1"="VAL1" pairs, stopping at '}' or ']'
  FieldsCount := 0;
  repeat
    FU := GetJSONField(P,P);
    inc(Len,length(FU));
    if P=nil then break;
```



```
Fields2[FieldsCount] := FU;  
Values[FieldsCount] := GetValue; // update EndOfObject  
inc(FieldsCount);  
until EndOfObject in [#0,'}',' ',''];  
Return(@Fields2,@Values,InlinedParams);  
(...)
```

And the sub-function GetValue makes use of GetJSONField also:

```
function GetValue: RawUTF8;  
var wasString: boolean;  
res: PUTF8Char;  
begin  
res := P;  
if (PInteger(res)^ and $DFDFDFDF=NULL_DF) and (res[4] in [#0,',','}',' ','']) then  
  // GetJSONField('null') returns '' -> check here to make a diff with ''''  
  result := 'null' else begin  
    // any JSON string or number or 'false'/'true' in P:  
    res := GetJSONField(res,P,@wasString,@EndOfObject);  
    if wasString then  
      if not InlinedParams and  
        (PInteger(res)^ and $0FFFFFFF=JSON_BASE64_MAGIC) then  
        // \uFFFF0base64encodedbinary -> 'X'hexaencodedbinary''  
        // if not inlined, it can be used directly in INSERT/UPDATE statements  
        result := Base64MagicToBlob(res+3) else  
        { escape SQL strings, cf. the official SQLite3 documentation }  
        result := QuotedStr(pointer(res),''') else  
        result := res;  
    end;  
    Inc(Len,length(result));  
end;  
end;
```

This code will create a string for each key/value in Fields2[] and Values[] arrays, but only once, with the definitive value (even single quote escape and BLOB un-serialize from Base-64 encoding are performed directly from the JSON buffer).

1.9. SWRS # DI-2.1.3

The framework shall use an innovative ORM (Object-relational mapping) approach, based on classes RTTI (Runtime Type Information)

1.9.1. Abstract

Design Input 2.1.3 (Initial release): The framework shall use an innovative ORM (Object-relational mapping) approach, based on classes RTTI (Runtime Type Information).

ORM, as defined in the *Software Architecture Design (SAD)* document, is used in the *Synapse mORMot Framework* for accessing data record fields directly from Delphi Code.

Object-relational mapping (ORM, O/RM, and O/R mapping) is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the Delphi programming language.

The published properties of classes inheriting from a new generic type named TSQLRecord are used to define the field properties of the data. Accessing database records (for reading or update) shall be made by using these classes properties, and some dedicated Client-side methods.

1.9.2. Implementation

Some Delphi RTTI (Runtime Type Information) objects and classes are implemented in the `SQLite3Commons.pas` unit. The *Synopse mORMot Framework* uses this custom functions and objects in order to access to the Delphi RTTI.

The generic functions supplied by the standard `TypeInfo.pas` unit where not found to be easy to use: there are some record types from one hand, which details the internal RTTI memory layout generated by the compiler, and there are some functions on the other hand. So the framework unified both RTTI memory layout and methods by defining some object types (i.e. not Delphi classes, but raw objects which can map directly the RTTI memory layout via a pointer) with some methods dedicated for RTTI handling and ORM. These object types are `TClassProp`, `TClassType`, `TEnumType`, `TTypeInfo` and `TPropInfo`.

Since this ORM is the core of the framework, the code of most of these objects has been tuned for performance: quit all of the methods have two versions in the framework, one in pure pascal code (easy to maintain and understand, and 64 bit compatible), and one in optimized i386 assembler.

As a result, ORM code based on RTTI is fairly easy to use. See for example who a database field index is retrieved for a `TSQLRecord` class:

```
function ClassFieldIndex(ClassType: TClass; const PropName: shortstring): integer;
var P: PPropInfo;
    CP: PClassProp;
begin
  if ClassType<>nil then
  begin
    CP := InternalClassProp(ClassType);
    if CP<>nil then
    begin
      P := @CP^.PropList;
      for result := 0 to CP^.PropCount-1 do
        if IdemPropName(P^.Name, PropName) then
          exit else
            P := P^.Next;
      end;
    end;
    result := -1;
  end;
end;
```

Internally, the `TSQLRecord` will cache some of this RTTI derived data into an internal `TSQLRecordProperties` instance, global for the whole process. For instance, the method used to retrieve a field index from its property name is the following:

```
function TSQLRecordProperties.FieldIndex(const PropName: shortstring): integer;
begin
  if self<>nil then
  for result := 0 to high(Fields) do
    if IdemPropName(Fields[result]^Name, PropName) then
      exit;
  result := -1;
end;
```

And will be available from `TSQLRecord.RecordProps.FieldIndex`.

1.9.3. TSQLRecord table properties

1.9.3.1. Per-class variable needed

For our ORM, we needed a *class variable* to be available for each `TSQLRecord` class type. This variable is used to store the properties of this class type, i.e. the database Table properties (e.g. table and

column names and types) associated with a particular TSQLRecord class, from which all our ORM objects inherit.

The `class var` statement was not enough for us:

- It's not available on earlier Delphi versions, and we try to have our framework work with Delphi 6-7;
- This `class var` instance will be shared by all classes inheriting from the class where it is defined - and we need ONE instance PER class type, not ONE instance for ALL

We need to find another way to implement this *class variable*. An unused VMT slot in the class type description was identified, then each class definition was patched in the process memory to contain our class variable.

1.9.3.2. Patching a running process code

The first feature we have to do is to allow on-the-fly change of the assembler code of a process.

When an executable is mapped in RAM, the memory page corresponding to the process code is marked as *Read Only*, in order to avoid any security attack from the outside. Only the current process can patch its own code.

We'll need to override a pointer value in the code memory. The following function, defined in `SynCommons.pas` will handle it:

```
procedure PatchCodePtrUInt(Code: PPtrUInt; Value: PtrUInt);
var RestoreProtection, Ignore: DWORD;
begin
  if VirtualProtect(Code, SizeOf(Code^), PAGE_EXECUTE_READWRITE, RestoreProtection) then
  begin
    Code^ := Value;
    VirtualProtect(Code, SizeOf(Code^), RestoreProtection, Ignore);
    FlushInstructionCache(GetCurrentProcess, Code, SizeOf(Code^));
  end;
end;
```

The `VirtualProtect` low-level Windows API is called to force the corresponding memory to be written (via the `PAGE_EXECUTE_READWRITE` flag), then modify the corresponding pointer value, then the original memory page protection setting (should be `PAGE_EXECUTE_READ`) is restored.

According to the MSDN documentation, we'd need to flush the CPU operation cache in order to force the modified code to be read on next access.

1.9.3.3. Per-class variable in the VMT

The VMT is the *Virtual-Method Table*, i.e. a Table which defines every Delphi class. In fact, every Delphi class is defined internally by its VMT, contains a list of pointers to the class's virtual methods. This VMT also contains non-method values, which are class-specific information at negative offsets:

Name	Offset	Description
vmtSelfPtr	-76	points back to the beginning of the table
vmtIntfTable	-72	TObject.GetInterfaceTable method value
vmtAutoTable	-68	class's automation table (deprecated)

vmtInitTable	-64	reference-counted fields type information
vmtTypeInfo	-60	the associated RTTI type information
vmtFieldTable	-56	field addresses
vmtMethodTable	-52	method names
vmtDynamicTable	-48	dynamic methods table
vmtClassName	-44	PShortString of the class name
vmtInstanceSize	-40	bytes needed by one class Instance
vmtParent	-36	parent VMT

We'll implement the low-level trick as detailed in this reference article available at <http://hallvards.blogspot.com/2007/05/hack17-virtual-class-variables-part-ii.html> in order to use the vmtAutoTable deprecated entry in the VMT. This entry was used in Delphi 2 only for implementing *Automation*. Later version of Delphi (our goal) won't use it any more. But the slot is still here, ready for being used by the framework.

We'll therefore be able to store a pointer to the TSQLRecordProperties instance corresponding to a TSQLRecord class, which will be retrieved as such:

```
class function TSQLRecord.RecordProps: TSQLRecordProperties;
begin
  if Self<>nil then begin
    result := PPointer(PtrInt(Self)+vmtAutoTable)^;
    if result=nil then
      result := PropsCreate(self);
    end else
      result := nil;
  end;
end;
```

Since this method is called a lot of time by our ORM, there is an asm-optimized version of the pascal code above:

```
class function TSQLRecord.RecordProps: TSQLRecordProperties;
asm
  or eax, eax
  jz @null
  mov edx, [eax+vmtAutoTable]
  or edx, edx
  jz PropsCreate
  mov eax, edx
@null:
end;
```

Most of the time, this method will be executed very quickly. In fact, the PropsCreate global function is called only once, i.e. the first time this RecordProps method is called.

The TSQLRecordProperties instance is therefore created within this function:

```
function PropsCreate(aTable: TSQLRecordClass): TSQLRecordProperties;
begin // private sub function makes the code faster in most case
  if not aTable.InheritsFrom(TSQLRecord) then
    // invalid call
    result := nil else begin
    // create the properties information from RTTI
    result := TSQLRecordProperties.Create(aTable);
    // store the TSQLRecordProperties instance into AutoTable unused VMT entry
```

```
PatchCodePtrUInt(pointer(PtrInt(aTable)+vmtAutoTable),PtrUInt(result));  
// register to the internal garbage collection (avoid memory leak)  
GarbageCollector.Add(result);  
end;  
end;
```

The GarbageCollector is a global TObjectList, which is used to store some global instances, living the whole process time, just like our TSQLRecordProperties values.

A per-class TSQLRecordProperties was made therefore available for each kind of TSQLRecord class.

Even most sophisticated methods of the ORM (like TSQLRecord. GetJSONValues) make use of these low-level object types. In most cases, the GetValue and SetValue methods of the TPropInfo object are used to convert any field value stored inside the current TSQLRecord instance in or from UTF-8 encoded text.

2. SQLite3 engine

2.1. SWRS # DI-2.2.1

The *SQLite3* engine shall be embedded to the framework

2.1.1. Abstract

Design Input 2.2.1 (Initial release): The SQLite3 engine shall be embedded to the framework.

The *SQLite3* database engine is used in the *Synopse mORMot Framework* as its kernel database engine. *SQLite3* is an ACID-compliant embedded relational database management system contained in a C programming library.

This library shall be linked statically to the *Synopse mORMot Framework*, and interact directly from the Delphi application process.

The *Synopse mORMot Framework* shall enhance the standard *SQLite3* database engine by introducing some new features stated in the *Software Architecture Design (SAD)* document, related to the Client-Server purpose or the framework - see *SWRS # DI-2.1.1*.

2.1.2. Implementation

It's worth noting that the *Synopse SQLite3 database engine*, whatever its name states, is not bound to *SQLite3* (you can use another database engine for data storage, for example we provide a *TSQLRestServerStaticInMemory* class which implements a fast but limited in-memory database engine). Therefore, the *SQLite3* engine itself is not implemented in the *SQLite3Commons.pas* unit, but in dedicated units.

The *SQLite3* engine is accessed at two levels:

- A low-level direct access to the *SQLite3* library, implemented in *SynSQLite3.pas*;
- A high-level access, implementing a Client-Side or Server-Side native *TSQLRest* descendant using the *SQLite3* library for its data persistence, in *SQLite3.pas*.

2.1.2.1. Low-Level access to the library

2.1.2.1.1. Compilation of the SQLite3 engine

First of all, the original source code of the library, which is retrieved from the official *SQLite3* web site in the form of the optimized Amalgamation file - see <http://www.sqlite.org/amalgamation.html>.. - is

compiled using the free Borland C++ command-line compiler.

Here are the defines used for this compilation:

```
//#define SQLITE_ENABLE_FTS3
// this unit is FTS3-ready, but not compiled with it by default
// if you don't use FTS3, dont define this conditional: you'll spare 50KB of code
// this conditional is defined at compile time, in order to create sqlite3fts3.obj
#define SQLITE_DEFAULT_MEMSTATUS 0
// don't need any debug here
#define SQLITE_THREADSafe 2
// assuming multi-thread safety is made by caller - in our framework, there is
// only one thread using the database connection at the same time, but there could
// be multiple database connection at the same time (previous was 0 could be unsafe)
#define SQLITE_OMIT_SHARED_CACHE 1
// no need of shared cache in a threadsafe calling model
#define SQLITE_OMIT_AUTOINIT 1
// sqlite3_initialize() is done in initialization section below -> no AUTOINIT
#define SQLITE_OMIT_DEPRECATED 1
// spare some code size
#define SQLITE_OMIT_TRACE 1
// we don't need sqlite3_profile() and sqlite3_trace() interfaces
#define SQLITE_OMIT_LOAD_EXTENSION 1
// we don't need extension in an embedded engine
#define SQLITE_OMIT_COMPILEOPTION_DIAGS 1
// we don't need Compilation Options Diagnostics in our embedded engine
#define SQLITE_OMIT_PROGRESS_CALLBACK 1
// we don't need sqlite3_progress_handler() API function
#define SQLITE_ENABLE_RTREE 1
// the RTREE extension is now (from v.1.8/3.7) compiled into the engine
//#define SQLITE_OMIT_LOOKASIDE
// since we use FastMMA, LookAside is not needed but seems mandatory in c source
```

The only code modification made to the official *SQLite3* engine source code is to make *winRead* and *winWrite* function external, which will be coded in pure pascal code, in order to implement our on-the-fly encryption of the database file:

```
extern int winRead(
    sqlite3_file *id,          /* File to read from */
    void *pBuf,               /* Write content into this buffer */
    int amt,                  /* Number of bytes to read */
    sqlite3_int64 offset       /* Begin reading at this offset */
);
extern int winWrite(
    sqlite3_file *id,          /* File to write into */
    const void *pBuf,         /* The bytes to be written */
    int amt,                  /* Number of bytes to write */
    sqlite3_int64 offset       /* Offset into the file to begin writing at */
);
```

Two .obj files are created, named *sqlite3.obj* and *sqlite3fts3.obj*, using the following batch command (named *c.bat* in the source code repository):

```
\\dev\\bcc\\bin\\bcc32 -6 -O2 -c -d -DSQLITE_ENABLE_FTS3 -u- sqlite3.c
copy sqlite3.obj sqlite3fts3.obj
\\dev\\bcc\\bin\\bcc32 -6 -O2 -c -d -u- sqlite3.c
```

The *sqlite3.obj* file won't include FTS3/FTS4, whereas *sqlite3fts3.obj* will include the FTS3/FTS4 module: the code size is a bit bigger. The *INCLUDE_FTS3* conditional must be defined for the whole application, to embed this module, as stated by the following code extracted from *SynSQLite3.pas*:

```
{ifdef INCLUDE_FTS3}
{$L sqlite3fts3.obj} // Link SQLite3 database engine with FTS3
{$else}
{$L sqlite3.obj}     // Link SQLite3 database engine
{$endif}
```

Some low-level functions, necessary for linking to the *Borland C++* generated .obj files, are coded in asm. These runtime functions will call Delphi equivalences, which are indeed close from the BCC32 need - see for instance `_ftol()` `_ftoul()` `malloc()` `free()` `memset()` `memmove()` `atol()` `_lldiv()` `strlen()` and such. Even higher-level functions - like `localtime()` or `qsort()` - are coded in pure Delphi code.

2.1.2.1.2. SQLite3 API access

Some types are defined in `SynSQLite3.pas` to map the types used by *SQLite3*: `TSQLite3DB`, `TSQLite3Statement`, `TSQLite3Blob`, `TSQLite3Value`, `TSQLite3FunctionContext`, which are mapped to a `PtrUInt`, i.e. an unsigned integer matching the current pointer size. This is the *handle* type exposed by the *SQLite* API.

Then most C-language interface to *SQLite* has been converted into pure Delphi external function or procedure calls (see <http://www.sqlite.org/c3ref/intro.html> for a complete reference). The conversion rule was to match the API name (all `sqlite3_*` identifiers), then provide the most Delphi-standard access to the parameters: for instance, we use standard `Integer/Int64/PUTF8Char` types, or a `var` declaration instead of a C pointer.

2.1.2.2. High level access

Some Delphi classes are introduced to manage all calls and statements to C-language interface to *SQLite*, mapping all `sqlite3_*` functions and methods to object-oriented methods.

The `SynSQLite3.pas` unit defines the following classes:

- `ESQLException` is a custom *SQLite3* dedicated Exception type;
- `TSQLDataBase` is a simple wrapper for direct *SQLite3* database manipulation;
- `TSQLRequest` encapsulates a *SQLite3* request;
- `TSQLTableDB` executes a SQL statement in the local *SQLite3* database engine, and get result in memory, as JSON content;
- `TSQLBlobStream` is available to access to a *SQLite3* BLOB Stream.

Those database access types are then used by the following Client-Server RESTful classes, to implement *SQLite3* storage for persistence of our ORM (the so called objects hibernation) in *SQLite3.pas*:

- `TSQLRestClientDB` implements a REST client with direct access to a *SQLite3* database, that is without the Client-Server aspect of the framework;
- `TSQLRestServerDB` can be used to implement a REST server using *SQLite3* as its storage engine.

2.2. SWRS # DI-2.2.2

The framework libraries, including all its *SQLite3* related features, shall be tested using Unitary testing

2.2.1. Abstract

Design Input 2.2.2 (Initial release): The framework libraries, including all its SQLite3 related features, shall be tested using Unitary testing.

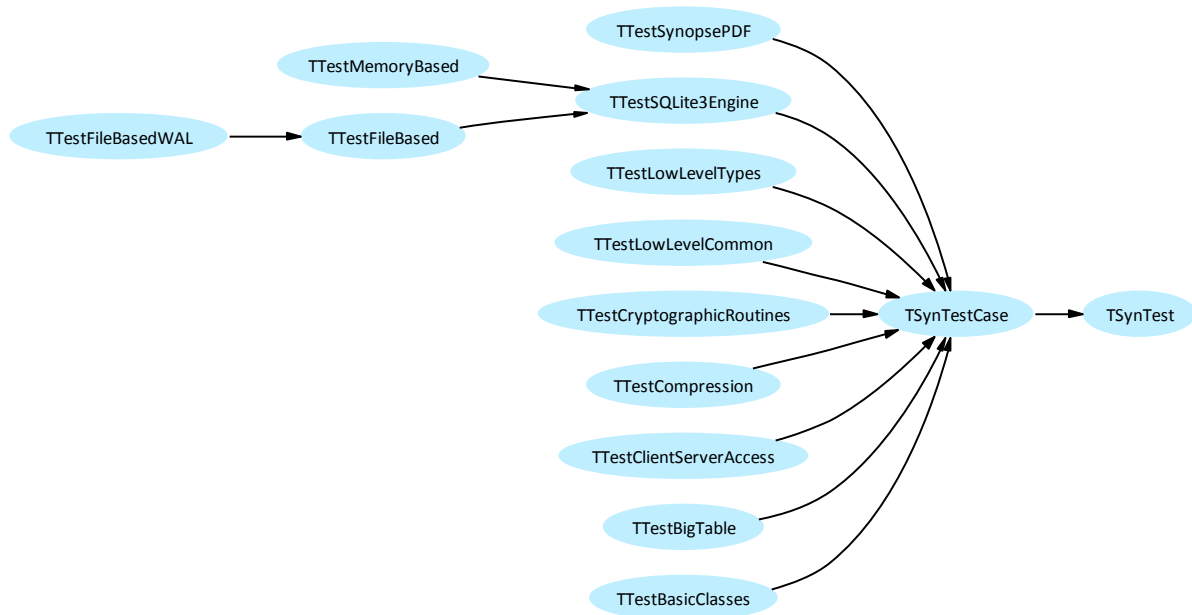
The *Synapse mORMot Framework* shall use all integrated Unitary testing features provided by a common testing framework integrated to all Synapse products. This testing shall be defined by classes,

in which individual published methods define the actual testing of most framework features.

All testing shall be run at once, for example before any software release, or after any modification to the framework code, in order to avoid most regression bug.

2.2.2. Implementation

Some tests classes have been developed, which methods cover most aspect of the framework:



TSynTestCase classes hierarchy

Those classes are implemented in SynCommons.pas, SQLite3Commons.pas, SQLite3.pas and SQLite3HttpServer.pas units.

2.3. SWRS # DI-2.2.3

The framework shall be able to access any external database, via OleDB, ODBC or direct access for Oracle (OCI) or SQLite3 (for external database files)

2.3.1. Abstract

Design Input 2.2.3 (Initial release): The framework shall be able to access any external database, via OleDB, ODBC or direct access for Oracle (OCI) or SQLite3 (for external database files).

The following external database providers shall be made available to the framework ORM:

- Any *OleDB* provider;
- Any *ODBC* provider;
- *Oracle* database, via direct OCI client access;
- *SQLite3* database engine.

A dedicated set of classes shall implement access, together with some advanced syntactic sugar, like fast late-binding, or advanced ORM mechanism, like Virtual Tables.

2.3.2. SynDB classes

The *Software Architecture Design* (SAD) document detailed the architecture, and main implementation part of the database-agnostic features of the framework.

2.3.3. Faster late binding

For both our *SynDB* and *SynBigTable* units, we allow *late-binding* of data row values, using a variant and direct named access of properties. It's a very convenient way of accessing result rows values.

2.3.3.1. Speed issue

But, in practice, this approach is slow. It uses the internal mechanism used for *Ole Automation*, here to access column content as if column names were native object properties. There is plenty of space for speed improvement here.

So, how does the variant type used by *Ole Automation* and our custom variant types (i.e. *TSynTableVariantType* or *TSynLDBRowVariantType*) handle their properties access?

Behind the scene, the Delphi compiler calls the *DispInvoke* function, as defined in the *Variant.pas* unit.

The default implementation of this *DispInvoke* is some kind of slow:

- It uses a *TMultiReadExclusiveWriteSynchronizer* under Delphi 6, which is a bit over-sized for its purpose: since Delphi 7, it uses a lighter critical section;
- It makes use of *WideString* for string handling (not at all the better for speed), and tends to define a lot of temporary string variables;
- For the getter method, it always makes a temporary local copy during process, which is not useful for our classes.

2.3.3.2. Fast and furious

So we rewrite the *DispInvoke* function with some enhancements in mind:

- Will behave exactly the same for other kind of variants, in order to avoid any compatibility regression, especially with *Ole Automation*;
- Will quick intercept our custom variant types (as registered via the global *SynRegisterCustomVariantType* function), and handle those with less overhead: no critical section nor temporary *WideString* allocations are used.

2.3.3.3. Implementation

Here is the resulting code, from our *SynCommons* unit:

```
procedure SynVarDispProc(Result: PVarData; const Instance: TVarData;
  CallDesc: PCallDesc; Params: Pointer); cdecl;
const DO_PROP = 1; GET_PROP = 2; SET_PROP = 4;
var i: integer;
    Value: TVarData;
    Handler: TCustomVariantType;
begin
  if Instance.VType=varByRef or varVariant then // handle By Ref variants
    SynVarDispProc(Result,PVarData(Instance.VPointer)^,CallDesc,Params) else begin
    if Result<>nil then
      VarClear(Variant(Result^));
    case Instance.VType of
```

```

varDispatch, varDispatch or varByRef,
varUnknown, varUnknown or varByRef, varAny:
  // process Ole Automation variants
  if Assigned(VarDispProc) then
    VarDispProc(pointer(Result),Variant(Instance),CallDesc,@Params);
  else begin
    // first we check for our own TSynInvokeableVariantType types
    if SynVariantTypes<>nil then
      for i := 0 to SynVariantTypes.Count-1 do
        with TSynInvokeableVariantType(SynVariantTypes.List[i]) do
          if VarType=TVarData(Instance).VType then
            case CallDesc^.CallType of
              GET_PROP, DO_PROP: if (Result<>nil) and (CallDesc^.ArgCount=0) then begin
                IntGet(Result^,Instance,@CallDesc^.ArgTypes[0]);
                exit;
              end;
              SET_PROP: if (Result=nil) and (CallDesc^.ArgCount=1) then begin
                ParseParamPointer(@Params,CallDesc^.ArgTypes[0],Value);
                IntSet(Instance,Value,@CallDesc^.ArgTypes[1]);
                exit;
              end;
            end;
          // here we call the default code handling custom types
          if FindCustomVariantType(Instance.VType,Handler) then
            TSynTableVariantType(Handler).DispInvoke(
              {$ifdef DELPHI6OROLDER}Result^{else}Result{$endif},
              Instance,CallDesc,@Params)
          else raise EInvalidOp.Create('Invalid variant invoke');
        end;
      end;
    end;
  end;
end;

```

Our custom variant types have two new virtual protected methods, named IntGet/IntSet, which are the getter and setter of the properties. They will to the property process, e.g. for our *OleDB* column retrieval:

```

procedure TSQLDBRowVariantType.IntGet(var Dest: TVarData;
  const V: TVarData; Name: PAnsiChar);
var Rows: TSQLDBStatement;
begin
  Rows := TSQLDBStatement(TVarData(V).VPointer);
  if Rows=nil then
    EOleDBException.Create('Invalid SQLDBRowVariant call');
  Rows.ColumnToVariant(Rows.ColumnIndex(RawByteString(Name)),Variant(Dest));
end;

```

As you can see, the returned variant content is computed with the following method:

```

function TOleDBStatement.ColumnToVariant(Col: integer;
  var Value: Variant): TSQLDBFieldType;
var Value: Variant): TSQLDBFieldType;
const FIELDTYPE2VARTYPE: array[TSQLDBFieldType] of Word = (
  varEmpty, varNull, varInt64, varDouble, varCurrency, varDate,
  {$ifdef UNICODE}varUString{$else}varOleStr{$endif}, varString);
var C: PSQLDBColumnProperty;
  V: PColumnValue;
  P: pointer;
  Val: TVarData absolute Value;
begin
  V := GetCol(Col,C);
  if V=nil then
    result := ftNull else
    result := C^.ColumnType;
  VarClear(Value);
  Val.VType := FIELDTYPE2VARTYPE[result];
  case result of
    ftInt64, ftDouble, ftCurrency, ftDate:

```

```
Val.VInt64 := V^.Int64; // copy 64 bit content
ftUTF8: begin
  Val.VPointer := nil;
  if C^.ColumnValueInlined then
    P := @V^.VData else
    P := V^.VAnsiChar;
  SetString(SynUnicode(Val.VPointer), PWideChar(P), V^.Length shr 1);
end;
ftBlob: begin
  Val.VPointer := nil;
  if C^.ColumnValueInlined then
    P := @V^.VData else
    P := V^.VAnsiChar;
  SetString(RawByteString(Val.VPointer), PAnsiChar(P), V^.Length);
end;
end;
```

This above method will create the variant content without any temporary variant or string. It will return TEXT (ftUTF8) column as SynUnicode, i.e. into a generic WideString variant for pre-Unicode version of Delphi, and a generic UnicodeString (=string) since Delphi 2009. By using the fastest available native Unicode string type, you will never loose any Unicode data during char-set conversion.

2.3.3.4. Hacking the VCL

In order to enable this speed-up, we'll need to change each call to DispInvoke into a call to our custom SynVarDispProc function.

With Delphi 6, we can do that by using GetVariantManager /SetVariantManager functions, and the following code:

```
GetVariantManager(VarMgr);
VarMgr.DispInvoke := @SynVarDispProc;
SetVariantManager(VarMgr);
```

But since Delphi 7, the DispInvoke function is hard-coded by the compiler into the generated asm code. If the *Variants* unit is used in the project, any late-binding variant process will directly call the `_DispInvoke` private function of *Variants.pas*.

First of all, we'll have to retrieve the address of this `_DispInvoke`. We just can't use `_DispInvoke` or `DispInvoke` symbol, which is not exported by the Delphi linker... But this symbol is available from asm!

So we will first define a pseudo-function which is never called, but will be compiled to provide a pointer to this `_DispInvoke` function:

```
procedure VariantsDispInvoke;
asm
  call Variants.@DispInvoke;
end;
```

Then we'll compute the corresponding address via this low-level function, the asm call opcode being `$E8`, followed by the relative address of the sub-routine:

```
function GetAddressFromCall(AStub: Pointer): Pointer;
begin
  if AStub=nil then
    result := AStub else
  if PBYTE(AStub)^ = $E8 then begin
    Inc(PtrInt(AStub));
    Result := Pointer(PtrInt(AStub)+SizeOf(integer)+PInteger(AStub)^);
```

```
end else
  Result := nil;
end;
```

And we'll patch this address to redirect to our own function:

```
RedirectCode(GetAddressFromCall(@VariantsDispInvoke),@SynVarDispProc);
```

The resulting low-level asm will just look like this at the call level:

```
Test0leDb.dpr.28: assert(Copy(Customer.AccountNumber,1,8)='AW000001');
00431124 8D45D8      lea eax,[ebp-$28]
00431127 50           push eax
00431128 6828124300   push $00431228
0043112D 8D45E8      lea eax,[ebp-$18]
00431130 50           push eax
00431131 8D45C4      lea eax,[ebp-$3c]
00431134 50           push eax
00431135 E86ED1FDFF   call @DispInvoke
```

It will therefore call the following hacked function:

```
0040E2A8 E9B3410100   jmp SynVarDispProc
0040E2AD E853568B5D   call +$5d8b5653
... (previous function content, never executed)
```

That is, it will jump (jmp) to our very own SynVarDispProc, just as expected.

In fact, the resulting code is very close to a direct `ISQLDBRows.Column['AccountNumber']` call. Using *late-binding* can be both fast on the execution side, and easier on the code side.

3. User interface

3.1. SWRS # DI-2.3

User Interface and Report generation should be integrated

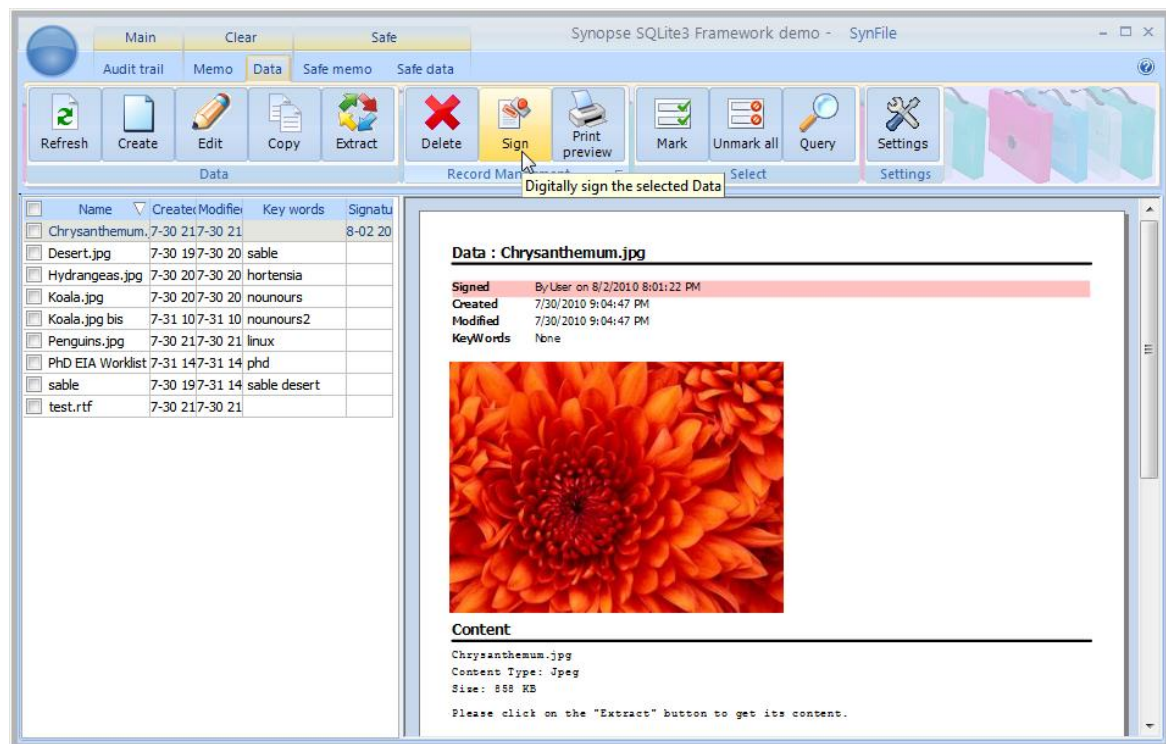
3.1.1. Abstract

Design Input 2.3 (Initial release): User Interface and Report generation should be integrated.

The *Synopse mORMot Framework* shall provide User Interface and Report generation from code.

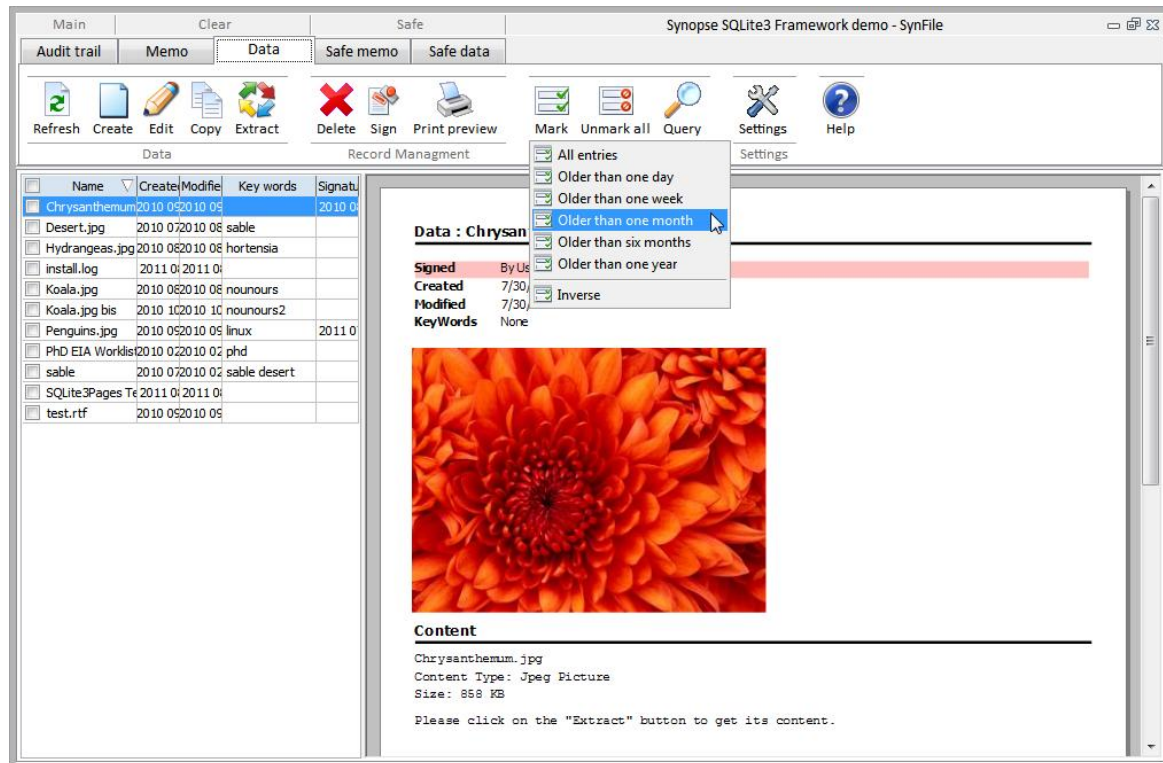
Such a ribbon-oriented interface shall be made available, in a per-table approach, and associated reports.

Here is a sample of screen content, using proprietary TMS components:



User Interface generated using TMS components

And here is the same application compiled using only VCL components, available from Delphi 6 up to XE2:



User Interface generated using VCL components

3.1.2. SynFile main Demo

The *Main SynFile Demo* section of the associated *Software Architecture Design* (SAD) document has already detailed the architecture and the code used to produce a full featured application, including the User Interface generation.

Please refer to these pages for sample code and general explanation about this feature of the framework.

3.2. SWRS # DI-2.3.1.1

A Database Grid shall be made available to provide data browsing in the Client Application - it shall handle easy browsing, by column resizing and sorting, on the fly customization of the cell content

3.2.1. Abstract

Design Input 2.3.1 (Initial release): An User Interface, with buttons and toolbars shall be easily being created from the code, with no RAD needed, using RTTI and data auto-description.

A Database Grid shall be made available to provide data browsing in the Client Application - it shall handle easy browsing, by column resizing and sorting, on the fly customization of the cell content.

3.2.2. Implementation

A standard TDrawGrid can be associated to a TSQLTable instance by using a TSQLTableToGrid object, as defined in the SQLite3UI.pas:

- Just call TSQLTableToGrid.Create(Grid,Table) to initiate the association;
- The Table will be released when no longer necessary;
- Any former association by TSQLTableToGrid.Create() will be overridden;
- Handle Unicode, auto column size, field sort, incremental key lookup, optional hide ID;
- *Ctrl + click* on a cell to display its full Unicode content.

For instance, here is how the SQLite3ToolBar.pas unit creates a grid for every TSQLRecord class it refers to:

```
constructor TSQLLister.Create(aOwner: TComponent; aClient: TSQLRestClientURI;
(...))
  fTableToGrid := TSQLTableToGrid.From(fGrid);
  if fTableToGrid=nil then begin
    // this Grid has no associated TSQLTableToGrid -> create default one
    if fClient.InheritsFrom(TSQLRestClientURI) then
      C := TSQLRestClientURI(fClient) else
      C := nil;
    fTableToGrid := TSQLTableToGrid.Create(fGrid,aTable,C);
    if aIDColumnHide then
      fTableToGrid.IDColumnHide;
  end;
  fTableToGrid.OnRightClickCell := OnRightClickCell;
  TableToGrid.OnValueText := aOnValueText;
  fGrid.DefaultDrawing := false; // we force full redraw
  TableToGrid.OnDrawCellBackground := OnDrawCellBackground;
  TableToGrid.OnSelectCell := OnSelectCell;
  (...)
```

All the process will be done in an automated manner, using the methods of the TDrawGrid component.

The current implementation is very fast, since the data is taken directly from the TSQLTable content. A grid with more than 200,000 rows is displayed with no delay. All content is converted into pure text, according to the RTTI information associated with the TSQLTable columns. If it was created as a TSQLTableJSON, from an ORM call of the framework, it will contain the RTTI information for each column. For instance, time and date will be displayed with the current internationalization settings, from either ISO 8601 encoded text (for TDateTime published property) or our optimized Int64 format (for TTimeLog / TModTime / TCreateTime published property).

3.3. SWRS # DI-2.3.1.2

Toolbars shall be able to be created from code, using RTTI and enumerations types for defining the action

3.3.1. Abstract

Toolbars shall be able to be created from code, using RTTI and enumerations types for defining the action.

3.3.2. Implementation

3.3.2.1. Rendering

The current implementation of the framework User Interface generation handles two kind of rendering:

- Native VCL components;
- Proprietary TMS components.

You can select which set of components are used, by defining - globally to your project (i.e. in the *Project/Options/Conditionals* menu) - the USETMSPACK conditional. If it is not set (which is by default), it will use VCL components.

3.3.2.2. Ribbon-like toolbars

As stated by the *Main SynFile Demo* section of the associated *Software Architecture Design* (SAD) document, ribbon-like toolbars can be generated by using the TSQLRibbon class, as defined in SQLite3ToolBar.pas.

This class will use one TSQLRibbonTab instance per TSQLRecord class type it handles, displayed on its own ribbon page, with an associated TDrawGrid instance and a TGDIPages report, via a corresponding TSQLLister instance. Parameters provided from code to the TSQLRibbon. Create method can customize the toolbar content on purpose. Actions will be provided as an enumeration type, and button captions will be extracted by *Un Camel Casing* of each enumerated value, using RTTI.

3.3.2.3. Stand-alone toolbars

A TSQLCustomToolBar object can be used to create some generic toolbars, with just some icons and actions on screen, with no reference to any associated TSQLRecord class. See for instance this sample code:

```
procedure TMainLogView.FormCreate(Sender: TObject);
begin
  FToolBar.Init(self, TypeInfo(TLogViewAction), ActionClick, ImageList, '');
  FToolBar.AddToolBar('Test')
end;
```

The above lines will create a panel on the owner form, with a toolbar containing one button per each TLogViewAction element. Icons will be taken from the supplied ImageList component, and the ActionClick event handler will be called when a button is pressed.

3.4. SWRS # DI-2.3.1.3

Internationalization (i18n) of the whole User Interface shall be made available by defined some external text files: Delphi resourcestring shall be translatable on the fly, custom window dialogs automatically translated before their display, and User Interface generated from RTTI should be included in this i18n mechanism

3.4.1. Abstract

Internationalization (i18n) of the whole User Interface shall be made available by defined some external text files: Delphi resourcestring shall be translatable on the fly, custom window dialogs automatically translated before their display, and User Interface generated from RTTI should be

included in this i18n mechanism.

3.4.2. Implementation

The `SQLite3i18n.pas` unit is able to handle both Internationalization (i18n) and Localization (L10n).

The `TLanguageFile` class is able to retrieve a custom list of text, and use it for all resourcestring and screen captions. The global `_()` function, or the `Translate` method of the `TLanguageFile` class can be used to translate any English text into the corresponding language.

The generic string type is used when some text is to be displayed on screen. Dedicated `U2S` and `S2U` functions, or even better the `UTF8ToString` and `StringToUTF8` methods of a `TLanguageFile` instance can be used for proper conversion.

Localization is performed via some dedicated methods of the `TLanguageFile` class, like `DateToText`, `DateTimeToText`, `TimeToText`.

3.5. SWRS # DI-2.3.2

A reporting feature, with full preview and export as PDF or TXT files, shall be integrated

3.5.1. Abstract

Design Input 2.3.2 (Initial release): A reporting feature, with full preview and export as PDF or TXT files, shall be integrated.

The *Synapse mORMot Framework* shall provide a reporting feature, which could be used stand-alone, or linked to its database mechanism. Reports shall not be created using a RAD approach (e.g. defining bands and fields with the mouse on the IDE), but shall be defined from code, by using some dedicated methods, adding text, tables or pictures to the report. Therefore, any kind of report shall be generated.

This reports shall be previewed on screen, and exported as PDF or TXT on request.

3.5.2. Implementation

The `SQLite3Pages.pas` unit implements a reporting component named `TGDIPages`, with full preview and txt/pdf export.

Anti-aliased drawing is using the `SynGdiPlus.pas` unit, and the `TGDIPlus`. `DrawAntiAliased` method.

The pdf export itself is implemented via the `SynPdf.pas` unit, via a `TPdfDocument` component: every page content (in fact, a `TMetaFile` instance) is rendered via the `TPdfCanvas`. `RenderMetaFile` method.