PÚLSAR
Index B-Tree and DataBase Manager

Version 3.00 Lite

© 2002 by Ricardo Ponce

Developed by HEPIKA SOLUTIONS
http://www.hepika.com.ar
rponce@lanet.com.ar

Information in this document is subject to change whitout notice. Companie names and data used in examples herein are ficticious unless otherwise noted. No parts of this document may be reproduced or transmited in any form or many means, electronic or mechanical, for any purpose, without the express written permission of RICARDO PONCE.
Copyright © 2002 by Ricardo Ponce. All Rights Reserved.
Printed in REPÚBLICA ARGENTINA.

Púlsar ©2002 by Ricardo Ponce

1. PÚLSAR INDEX B-TREE AND DATABASE MANAGER

Púlsar is a Dynamic Link Library that allows to programmers administer database files in a flexible and effective way, favoring the project integration through a DLL that allows him to become independent of troublesome programming (ordination, search, modification, and indexation routines).

Púlsar Index B-Tree and Database Manager is now distributed in two formats: FULL and LITE versions. The Lite version is completely operable and is my intention maintain it in that way.

This manager is (***besides free!***), transparent, without splash screens, without serial numbers neither limitations of any type. ***Púlsar*** will offer him all the power of his algorithms, without forced limits that are not those characteristic of his design and versatility.

2. THE NEW

- This version (3.0) offer total compatibility with ***PowerBasic***, ***RapidQ*** and ***VisualBasic*** compilers. (***Lite and Full versions***).
- Total and transparent compatibility with databases files generated in 1.0, 1.0a and 2.0 versions (***Lite and Full versions***).
- Not required User Identification (***Lite and Full versions***).
- Lite version ***freeware***.
- Free registration with full technical support: all their questions without additional costs (***Lite and Full versions***).
- You can integrate their applications through packed functions (***Lite and Full versions in PowerBasic, Visual Basic and RapidQ compilers***).
- You can program the Púlsar heart like up to now (***Lite and Full versions Only PowerBasic programmers***).
- English and spanish versions integrated definitively (***Lite and Full versions***)
- Operate Encripted databases: encript their information easy and quickly (***Full Version***).
- Encriptation Password modified by final user (***Full Version***).
- Include memo field in you records without size limits (***Full Version***).
- Upgrade to Full version at excellent cost (***Only 1.0a registered users***).
- Source Code Technical Support (***Full Version***).

3. ¿ WHY PÚLSAR ?

As programmer, I have proven several databases managers, and not any allowed to integrat their benefits with my applications in a simple and transparent way. Big limitations exist in the handling of text format, limiting the number of bytes in inflexibles record fields. ***Púlsar*** manages the texts obtained from **textboxs** and **richedit** controls, without size limitation, and without the expressed necessity of defining a record size in memo fields. Neither I love strange formats, closed algorithms and the "secrets ways" of the classic managers and indexs b-trees. The information is too valuable as to take a risk in an imprecise recording or in tedious recovery sessions because some workstation terminal is turned off. I like to maintain the control in my owns applications, and I imagine that most programmers thinks in the same way. ***Púlsar*** allows him to maintain the whole control, and it was designed with that specific purpose.

4. LICENSE COPYRIGHT AND REGISTRATION

I have decided to distribute ***Púlsar*** in two versions: ***LITE*** and ***FULL***. The Lite version is freeware and is the small brother of this database manager. He has less functions, but non limitations. It is completely operable. I have program it in that way because I love the free software, some much as you. But I also believe that is a good idea to encourage to the freeware soft developers, and is the reason that I offer a FULL registered version.

If he likes this software, and he believes that it is worthwhile, is at your disposal the ***3.0 FULL REGISTERED version*** at ***U$S 40,00***. That is not an excessive cost. If you obtain a ***1.0a Púlsar Registration Number*****, you may get the 3.0 Full version at** ***U$S 15***.

Púlsar ©2002 by Ricardo Ponce

Think that the free software can only stay in that way if the developers have a retribution that justifies the creation and debugging. It is sometimes programmed by pleasure, but the money always is an incentive for any industry, still that software.

Anyone ***Púlsar*** version that you have, it can distribute the DLL ***freely***, without necessity of paying neither royalties. Add a legend in their manuals that he says:

" This program use Púlsar Index B-Tree Manager"

Include my personal e-mail next the legend. Of course, I solicit him that doesn't subject this software to reverse engineering, after everything that novelty is in a index b-tree except that it is developed in pure **PowerBasic** code and it is ***completely free***?

Remember that this software is distributed in this way such which is, without any type of guarantees. There are not explicit neither implicit guarantees. You are the only responsible for the direct or indirect derived problems for use ***Púlsar*** manager and just by using this index b-tree, you is accepting the terms of this license.

5. PÚLSAR TOOLS, UTILITIES ... AND MORE

At this time, it is already available an exceptional tool to control the information of their databases and to recover it: **Púlsar Analizer**. He can obtain it freely in my website. ***Púlsar Analizer*** operate in databases generated in **1.0**, **1.0a**, **2.0** and **3.0 Lite** versions.

He will be able to see also that there is in **Beta Phase** a Visual environment to operate ***Púlsars*** databases: ***Visual Púlsar*** that will allow him to ***create forms and independent applications without necessity of writing code !!!***

VisualPúlsar is programmed in ***Púlsar 3.0 Full*** to generate EXE's and ***NEED BETA TESTERS*** !!! Does he want to be part of our BetaTesters Team and to receive Púlsar 3.0 and VisualPúlsar full versions FREE ??? Yes ??? Great !!!. Visit my web site and fill the form in **VisualPúlsar Beta Testers Program Official Page**.

If you are a registered user of our products, you will obtain full technical support, but also promotional offers on all our products.

During 2003 we will launch these programmers solutions:

* **VisualPúlsar:** rapid applications generation environment
* **Púlsar Internet Conectivity Toolkit** to operate your databases through internet
* **Help Developer System** hipertext generator

Visit **HEPIKA SOLUTIONS** in http://www.hepika.com.ar and obtain you ***Púlsar*** **Registration Code FREE**. The Púlsar Registration Code allow get **FULL TECHNICAL SUPPORT** and obtain **OFFERS** to **UPDATE** ours products at excellents prices.

6. ABOUT THIS MANUAL

If you has arrived to the point of needing an Index B-Tree and a Database Manager for their applications, he means that is not a beginner in this crazy programming world, so I won't play basic topics. If he has doubts that here don't clear up, I invite him to consult me to **rponce@lanet.com.ar** or visit my website in **http://www.hepika.com.ar**.

This is not a program designed for general public, is designed only for programmers, and is for that reason that to operate this database manager you will have some programming language: **PowerBasic**, **Rapid-Q**, ***Visual Basic***, etc. In fact, it can be operated with any language that supports DLL's, maybe with somes modifications to the declarations and calls.
The information that you can obtain about ***Púlsar*** won't be complete if he doesn't give a look to the source code attach in the distribution package. The sources are in **PowerBasic** and **RapidQ**

compilers but I hope to be able to add other languages. Don't forget to visit my website to find novelties, improvements and tools for yours databases.

7. CONVENTIONALS AND OTHERS DATABASES

Púlsar allows him, through some few programmable commands, to create a index system files, containing information and facilitating its consults, modification, deleting, etc.
It manages different types of files: **conventional databases**, **encrypted databases**, **conventional databases with memo fields**, **counter files** and **LOG comment files**.

Lite Version operate:

- Conventional Databases
- Counter files
- Log comment files.

Full Version operate:

- Conventional Databases
- Conventional Databases with memo fields
- Encripted Databases
- Encripted Databases with memo fields
- Counter files
- Log comment files.

In ***Púlsar***, a conventional database is a fixed records group whose information is organized in fields, what is classic in the program's world. The novelty resides in that this administrator offers him all the facilities so that you don't need to program search, index or delete routines. A great part of developed code of our programs is used for file handling routines. If you use ***Púlsar***, he will be able to forget the file handling and to be met organize information better to relate yours databases and to give a bigger flexibility in the programs and more efficiency in information use.

One of this manager key points, is the internal form that organizing memo fields in you databases. A database with memo field is not a conventional database. ***Púlsar*** doesn't manage the memo field like conventional information, but with flexible records, allowing to take advantage to the maximum space in disk and avoiding the lost bytes that implies to declare a great size field that doesn't take advantage entirety. In this manager, when creating a database with memo fields, you won't have necessity to declare memo size neither data index forms, because ***Púlsar*** gathers the flexibility of the binary data with the random organization files.

Too he will be able to program counter and LOG's files easy and quickly, forgetting completely to create update and read routines.

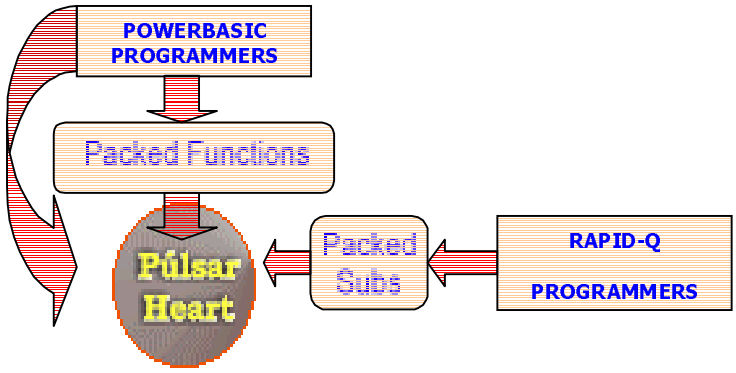
Púlsar was programmed in **PowerBasic 7.0**, pure basic code and consists of some 7.300 lines of source code integrated in ***Púlsar exportable function*** and the news ***Packed Functions***. ***Púlsar*** is prepared to work in network, offering the possibility to share their files. It can operate until a maximum of 99.999.999 records in data files. The fields of each records are programmed as strings, and each one can contain up to 999 characters. A record, in ***Púlsar***, cannot contain more than 999 fields. The **memo fields** in you records are not neither size limits.

8. NEW PACKED FUNCTIONS

Until **2.5** version, ***Púlsar*** was programmed through an only exportable function. The programmer sent a numeric command into the function and ***Púlsar*** acted in consequence. That programming form, operating the ***Púlsar*** heart directly, still it stays for the **PowerBasic** programmers.
But starting from the **3.0** version, the user has the possibility to operate ***Púlsar*** through a series of packed functions that add power, stability and facility of use in **PowerBasic**, adding compatibility with other compilers and other languages.

In this version they have been integrated the English and Spanish languages definitively, existing equivalent functions in both languages and compatible routines for freeware **RapidQ** compiler.
The ***Púlsar*** design contemplates a heart (the ***Púlsar*** function), now this heart can be operated through ***packed functions***, giving to programmer freedom election and a organized tools group that will work only in a more efficient way when the programmer requires it.

The **PowerBasic programmers** can operate ***Púlsar*** through these two modalities. The **RapidQ programmers** can only see the packed subroutines and only operate ***Púlsar*** through them.



These new paked functions are available in 3.0 Lite version:

POWERBASIC PROGRAMMERS (Paked Functions)		RAPID-Q PROGRAMMERS (Packed Subroutines)
ENGLISH FUNCTIONS	**SPANISH FUNCTIONS**	**ENGLISH SUBS**
PulsarBuild	PulsarBuild	RQ_PulsarBuild
PulsarEngine	PulsarEngine	RQ_PulsarEngine
NewConventionalDB	NuevaBDDConvencional	RQ_NewConventionalDB
SaveIndexConvDB	GrabaIndexaBDDConv	RQ_SaveIndexConvDB
GetQuantityOfRecords	VerCantidadRegistros	RQ_GetQuantityOfRecords
GetRecordConvDB	VerRegistroBDDConv	RQ_GetRecordConvDB
SINEConvDB	GSNEBDDConv	RQ_SINEConvDB
SaveConvDB LIB	GrabaBDDConv	RQ_SaveConvDB LIB
ReindexConvDB	ReidexaBDDConv	RQ_ReindexConvDB
ReturnRecordByte	RetornaByteRegistro	RQ_ReturnRecordByte
SARIndexConDB	RIndexBDDCon	RQ_SARIndexConDB
SARByteConDB	BYRByteBDDCon	RQ_SARByteConDB
DeleteRecordConvDB	BorraRecordBDDConv	RQ_DeleteRecordConvDB
ModifyRecordConvDB	ModificaRecordBDDConv	RQ_ModifyRecordConvDB
GetCounter	VerContador	RQ_GetCounter
SetCounter	SetearContador	RQ_SetCounter
DeleteLog	BorraLog	RQ_DeleteLog
SaveInLog	GrabaEnLog	RQ_SaveInLog
ReadLog	LeeLog	RQ_ReadLog

9. PÚLSAR FILES

When you operate a file, the manager assigns him automatically an extension that describes the file operation and operability:

Púlsar file's extensions:

.PLS
.IDX
.PLM
.CNT
.LOG

.PLS:	Binary file, witch contains the DataBase records data
.IDX:	DataBase index file
.PLM:	Binary file, witch contains the DataBase Memo fields
.CNT:	Counter file
.LOG:	Comment file

This manager divides the database information in three files, one contains the information itself, and it is identified easily by the extension **.PLS**, other, identified by the extension **.IDX**, contains the information referred to the database structure and save in his interior the ordination table (or index table) hisself, finally, the memo fields are save in a file identified by **.PLM** estension.
The .IDX file is the main structure of ***Púlsar*** database, being constituted in the file that contains the structure, initialization data, tables and order method of its database. But at the same time, all information is recorded in the .PLS and .PLM files, in the .IDX file destruction event, you data will be preserved by the system.

10. POWERBASIC PACKED FUNCTIONS PROGRAMMERS SECTION

This section only describe how operate ***Púlsar*** in **PowerBasic 6.x and 7.x** through the new Packed Functions. If you are a **PowerBasic Programmer**, but desire operate the **Púlsar Heart directly** as old versions, please got to **11 SECTION** that describe how to do.
If you are a **Rapid-Q programmer**, please go to **12 SECTION**, that describe how operate ***Púlsar*** under Rapid-Q compiler.

10.1 OPERATING PÚLSAR THROUGH PACKED FUNCTIONS

GENERALITIES

Púlsar is a DLL specially designed to operate with **PowerBasic**, although in fact, he can make it with any language that support DLL's as **Rapid-Q**.

First step consist in declare this packed functions into you program

ENGLISH FUNCTIONS:

```
DECLARE FUNCTION PulsarBuild LIB "pulsar.dll"()AS STRING
DECLARE FUNCTION PulsarEngine LIB "pulsar.dll"()AS STRING
DECLARE FUNCTION NewConventionalDB LIB "pulsar.dll"(Structure$,IndexField$, DataBasePath$,_
                 DataBaseFile$)AS STRING
DECLARE FUNCTION SaveIndexConvDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                  RecordToSave$)AS STRING
DECLARE FUNCTION GetQuantityOfRecords LIB "pulsar.dll"( DataBasePath$, DataBaseFile$)AS STRING
DECLARE FUNCTION GetRecordConvDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                  RecordNumber$)AS STRING
DECLARE FUNCTION SINEConvDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                   RecordToSave$)AS STRING
DECLARE FUNCTION SaveConvDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                   RecordToSave$)AS STRING
DECLARE FUNCTION ReindexConvDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$)AS STRING
DECLARE FUNCTION ReturnRecordByte LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                 RecordNumber$)AS STRING
DECLARE FUNCTION SARIndexConDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                 DataToSearch$)AS STRING
DECLARE FUNCTION SARByteConDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                 DataToSearch$)AS STRING
```

```
DECLARE FUNCTION DeleteRecordConvDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                        RecordNumber$)AS STRING
DECLARE FUNCTION ModifyRecordConvDB LIB "pulsar.dll"( DataBasePath$, DataBaseFile$,_
                        NewRecordInDB$, RecordToModify$)AS STRING
```

SPANISH FUNCTIONS:

```
DECLARE FUNCTION PulsarBuild LIB "pulsar.dll"()AS STRING
DECLARE FUNCTION PulsarEngine LIB "pulsar.dll"()AS STRING
DECLARE FUNCTION NuevaBDDConvencional LIB "pulsar.dll"(Estructura$,Indice$,Ubicacion$,_
                        Archivo$)AS STRING
DECLARE FUNCTION GrabaIndexaBDDConv LIB "pulsar.dll"(Ubicacion$,Archivo$,RecordToSave$)AS STRING
DECLARE FUNCTION VerCantidadRegistros LIB "pulsar.dll"(Ubicacion$,Archivo$)AS STRING
DECLARE FUNCTION VerRegistroBDDConv LIB "pulsar.dll"(Ubicacion$,Archivo$,RecordNumber$)AS STRING
DECLARE FUNCTION GSNEBDDConv LIB "pulsar.dll"(Ubicacion$,Archivo$,RecordToSave$)AS STRING
DECLARE FUNCTION GrabaBDDConv LIB "pulsar.dll"(Ubicacion$,Archivo$,RecordToSave$)AS STRING
DECLARE FUNCTION ReidexaBDDConv LIB "pulsar.dll"(Ubicacion$,Archivo$)AS STRING
DECLARE FUNCTION RetornaByteRegistro LIB "pulsar.dll"(Ubicacion$,Archivo$,RecordNumber$)AS STRING
DECLARE FUNCTION BYRIndexBDDCon LIB "pulsar.dll"(Ubicacion$,Archivo$,DataToSearch$)AS STRING
DECLARE FUNCTION BYRByteBDDCon LIB "pulsar.dll"(Ubicacion$,Archivo$,DataToSearch$)AS STRING
DECLARE FUNCTION BorraRecordBDDConv LIB "pulsar.dll"(Ubicacion$,Archivo$,RecordNumber$)AS STRING
DECLARE FUNCTION ModificaRecordBDDConv LIB "pulsar.dll"(Ubicacion$,Archivo$,NewRecordInDB$,_
                        RecordToModify$)AS STRING
```

The only difference between the english and spanish functions, is the text that composes the error code returned in the event of a bug occur during the operations invoqued. Operatively they are identical. It is not necessary to declare all functions, alone declares the languaje functions that you operated in their programs. These declarations can see them in the included samples programs (view Pulsar30Lite.Inc file)

OPERATING PÚLSAR THROUGH STRINGS VARIABLES

As you can see, all functions are declared as strings, as well as all required variables. ***Púlsar*** operates only with string variables. If you tries to operate with numeric codes without to transform them before to string format, their compiler will return an error code.

The functions return different messages types (always in string format) when being invoked. If error occur, function return error code in this string format:

"ERROR CODE:xxx.Error Description"

Where **xxx** is a numeric error code and **Error Description** is the error description in corresponding language (see **ERROR CODES SECTION**).

DATABASES FIELDS FORMAT

Púlsar operates the fields into database records as ***text fields***. You will transform the numeric information to save to string format with the purpose of the field is not rejected by the database manager. In the same way, when reading the record information, remember to transform the content of a string to numeric format.

USING USER DEFINED TYPES (UDT) IN YOUR PROGRAMS

To save and to read records ***Púlsar*** operate only with strings variables. When create a database, ***Púlsar*** verifies the exact correspondence among the bytes defined in record and the bytes entered by the user during the operations. If there is not an exact correspondence between the size of the string and the size of defined record, ***Púlsar*** rejects the operation to avoid data damages.

This way to work the information demands precision when entering the characters and a very special care when working with variables that don't contain the quantity defined characters for each field, for what you must implement verification routines and stuffing the data with empty spaces. A more precise and effective form to works is to use **U**sers **D**efined **T**ypes variables (**UDT**), in such way is the compiler the one that assumes the control tasks on the quantity of characters assigned for each field.

In this manual we will use in all examples a database that contains records formed by these fields:

FIELD NAME	*SIZE*
APELLIDO	30 characters
NOMBRES	40 characters
E-MAIL	20 characters
WEB	50 characters

It would be convenient to define the following UDT structure for our database:

```
TYPE Fields
    Apellido    AS STRING * 30   'last name field
    Nombres     AS STRING * 40   'first name field
    Email       AS STRING * 20   'e-mail field
    Web         AS STRING * 50   'web field
END TYPE

GLOBAL DatabaseRecord AS Fields      'DatabaseRecord global variable allow
                                     'operate with database records
```

This way simplifies vastly the database records's operations and avoids human errors.

10.2 THE PACKED FUNCTIONS

10.2.1 PULSARBUILD

This function return the ***Púlsar*** Build version installed in you system. In other words, return the DLL Build version installed in the Windows system directory. It allows to programmers prevent conflicts with yours applications that require a specific Púlsar version. It doesn't generate screens neither interactions of any type. This version return the “3011112002L” string.

Example:

```
Build$ = PulsarBuild$()

IF Build$<>"3011112002L" THEN
    MSGBOX "This program require Púlsar 3.0 Lite Build 3011112002L installed in you system."
END IF
```

Spanish and english functions are identicals.

10.2.2 PULSARENGINE

This function return the ***Púlsar*** Engine version installed in you system. This function prevent conflicts with yours applications that require a specific Púlsar compiled version. This version return the “3.0L” string.

Example:

```
Engine$ = PulsarEngine$()

IF Engine$<>"3.0L" THEN
    MSGBOX "This program require Púlsar Engine 3.0L installed in you system."
END IF
```

Spanish and english functions are identicals.

10.2.3 CREATE NEW DATABASE

English function

NewConventionalDB$
Format: a$=NewConventionalDB$(Structure$,IndexField$,DataBasePath$,DataBaseFile$)
Return: "ok" if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

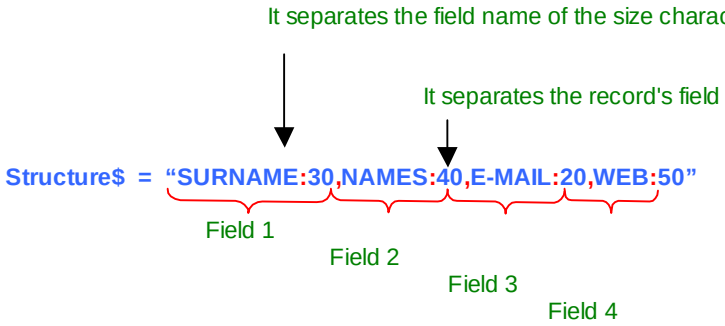
NuevaBDDConvencional$
Formato: A$= NuevaBDDConvencional$ (Estructura$,Indice$,Ubicación$,Archivo$)
Retorna: "ok" si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

NewConventionalDB$ packed function allow create a new database file in your disk.

It requires to define the following variables:

Structure$	Use Structure$ variable to define the structure coincident with the UDT definded in you personal database. In this example we use this UDT: TYPE Fields Apellido AS STRING * 30 'last name field Nombres AS STRING * 40 'first name field Email AS STRING * 20 'e-mail field Web AS STRING * 50 'web field END TYPE GLOBAL DatabaseRecord AS Fields You ill indicate through this variable in appropriate format each field name and size, separated by the two points character (":"). Among each field definition, it should interpose a comma character (",").

Let us see:



Observe the separater chars.

During the database creation, the Structure$ variable will contain the field definition

	at least with a minimum of 1 char. They are not allowed name of repeated fields inside oneself database. ***Púlsar*** distinguishes between upper and lowercase, in such way the field names like "**telephone**" and "**TELEPHONE**" are coincident, and the manager will reject them. The **names** which you identify each record field can only have up to **30 chars**, but their content can arrive to **999 characters**. If you define a field name with more than 30 chars, the manager will proceed to cut it and he will only read the first 30 chars. If you tries to assign to field's content more than 999 characters, ***Púlsar*** will assign him for defect the allowed maximum value (999). **Example:** **Structure$="APELLIDO:30,NOMBRES:40,E-MAIL:20,WEB:50"**
IndexField$	***Púlsar*** orders the information in database through an index table (or index). This index is ordered according the content of one field. The field that defines the ordination in database will be defined through the IndexField$ variable. To define an index field in database you will enter in this variable the name field. Following our example, if he wants that the database is ordered through the E-MAIL field, you will indicate through the IndexField$ assignment the database index field name: IndexField$ = “E-MAIL” He must make sure that the index field defined in this variable coincides exactly with one of the fields defined in Structure$ variable, or otherwise, you will receive an error message. The index field definition is not obligatory. In case the IndexField$ variable remains empty during the database inicializacion, ***Púlsar*** will assign as defect index field the first field defined in the Structure$ variable, in our case, the defect index field will be made through the field APELLIDO. **Example:** **IndexField$ = “APELLIDO”**
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. Indistinctly the path can finish or not with the " \ " character. If during the command execution the path defined by user doesn't exist, it is created automatically by ***Púlsar***. **Example:** **DataBasePath$ = “c:\pulsar\ficheros” or** **DataBasePath$ = “c:\pulsar\ficheros\”**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. ***Púlsar*** will assign automatically to generated files a defined extension and he will prepare the files to receive information. This manager divides the database information in three files, one contains the information itself, and it is identified easily by the extension **.PLS**, other, identified by the extension **.IDX**, contains the information referred to the database structure and save in his interior the ordination table (or index table) hisself, finally, the memo fields are save in a file identified by **.PLM** estension.

	The .IDX file is the main structure of ***Púlsar*** database, being constituted in the file that contains the structure, initialization data, tables and order method of its database. But at the same time, all information is recorded in the .PLS and .PLM files, in the .IDX file destruction event, you data will be preserved by the system. **Example:** **DataBaseFile$ = "CLIENTES"**

Example english function:

```
'Use the Structure$ variable to define the structure 'coincident with the UDT:
'TYPE Fields
'   Apellido    AS STRING * 30   'last name field
'   Nombres   AS STRING * 40   'first name field
'   Email       AS STRING * 20   'e-mail field
'   Web         AS STRING * 50    'web field
'END TYPE
'GLOBAL DatabaseRecord AS Fields

Structure$="APELLIDO:30,NOMBRES:40,E-MAIL:20,WEB:50"
IndexField$="APELLIDO"
DataBasePath$="c:\pulsar"
DataBaseFile$="CLIENTES"

CallPulsar$=NewConventionalDB$(Structure$,IndexField$,DataBasePath$,DataBaseFile$)

IF CallPulsar$<>"ok" THEN
      'If error occur, function return error code in string
      'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
      MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$
ELSE
      'If database file is created, the packed function return "ok".
      MSGBOX "OPERACION OK."
END IF
```

Example spanish function:

```
'Usar la variable Estructura$ para definir los campos coincidentemente con la estructura UDT:
'TYPE Fields
'   Apellido    AS STRING * 30   'campo apellido
'   Nombres   AS STRING * 40   'campo nombres
'   Email       AS STRING * 20   'campo e-mail
'   Web         AS STRING * 50    'campo web
'END TYPE
'GLOBAL DatabaseRecord AS Fields

Estrucura$="APELLIDO:30,NOMBRES:40,E-MAIL:20,WEB:50"
Indice$="APELLIDO"
Ubicación$="c:\pulsar"
Archivo$="clientes"

CallPulsar$= NuevaBDDConvencional$ (Estructura$,Indice$,Ubicación$,Archivo$)

IF CallPulsar$<>"ok" THEN
      'Si se genera un error, la función retorna un codigo en formato de cadena
      'En este formato: "ERROR CODE:xxx...." (xxx es el código de error - vea la seccion de errores-)
      MSGBOX "OPERACION NO COMPLETADA: "+CallPulsar$
ELSE
      'Si se crea la base de datos, la función retorna "ok".
```

```
    MSGBOX "OPERACION OK."
END IF
```

10.2.4 SAVE A RECORD AND REINDEX YOU DATABASE

English function

SaveIndexConvDB$
Format: a$= SaveIndexConvDB $(DataBasePath$,DataBaseFile$,RecordToSave$)
Return: "ok" if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

GrabaIndexaBDDConv$
Formato: A$= GrabaIndexaBDDConv$ (Ubicación$,Archivo$,RegistroAGrabar$)
Retorna: "ok" si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

SaveIndexConvDB$ packed function allow save a record into you database and reorder the index table automatically. You remember that ***Púlsar*** will save the record still when the index information is duplicated in another record. To verify the existence (or not) of a similar record saved previously, you will make use the SARIndexConDB$ or SARByteConDB$ functions, especialized functions that search record into you database. Other mode is to use SINEConvDB$ function, that save a record only if not exist previously in you database.
Remember that this function save the record whit the automatic database ordering (indexation).

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordToSave$	RecordToSave$ variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use UDT structures. **Example:** DatabaseRecord.Apellido = Ponce 'Surname DatabaseRecord.Nombres = Richard 'Names DatabaseRecord.Email = rponce@lanet.com.ar DatabaseRecord.Web = http://www.hepika.com.ar RecordToSave$ = DatabaseRecord 'assign the record fields saved in

		'DatabaseRecord UDT into 'RecordToSave$ variable for call 'packed function

Example english function:

```
DatabaseRecord.Apellido = Ponce       'Surname
DatabaseRecord.Nombres = Richard   'Names
DatabaseRecord.Email = rponce@lanet.com.ar
DatabaseRecord.Web = http://www.hepika.com.ar

RecordToSave$ = DatabaseRecord   'assign the record fields saved in DatabaseRecord UDT into
                                 'RecordToSave$ variable for call packed function
DataBasePath$="c:\pulsar"
DataBaseFile$="Conventional_DB"

CallPulsar$=SaveIndexConvDB(DataBasePath$,DataBaseFile$,RecordToSave$)

'if not error occur, function return "ok", if error occur, return '"ERROR CODE:XXX.Error_Description"
'string into CallPulsar$ variable

IF CallPulsar$<>"ok" THEN
      MSGBOX "OPERATION IS NOT COMPLETED:"+CallPulsar$
END IF
```

Example spanish function:

```
DatabaseRecord.Apellido = Ponce       'Apellido
DatabaseRecord.Nombres = Richard   'Nombres
DatabaseRecord.Email = rponce@lanet.com.ar
DatabaseRecord.Web = http://www.hepika.com.ar

RegistroAGrabar$ = DatabaseRecord   'asigna el registro contenido en DatabaseRecord UDT a
                                    'la variable RegistroAGrabar$ para llamar a la funcion
Ubicacion$="c:\pulsar"
Archivo$="Conventional_DB"

CallPulsar$= GrabaIndexaBDDConv $ (Ubicación$,Archivo$,RegistroAGrabar$)

'Si no ocrrue ningún error, la función retorna "ok", en caso de error, retorna la cadena
'"ERROR CODE:XXX.Description_del_error" dentro de la variable CallPulsar$

IF CallPulsar$<>"ok" THEN
      MSGBOX "LA OPERACION NO FUE COMPLETADA:"+CallPulsar$
END IF
```

10.2.5 SAVING A RECORD IN DATABASE WHITHOUT INDEXATION

English function

SaveConvDB $
Format: a$= SaveConvDB$(DataBasePath$,DataBaseFile$,RecordToSave$)
Return: "ok" if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

GrabaBDDConv $
Formato: A$= GrabaBDDConv $ (Ubicación$,Archivo$,RegistroAGrabar$)
Retorna: "ok" si no ocurre ningun error

Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

SaveConvDB$ packed function allow save a record into you database **without** reorder the index table. You remember that ***Púlsar*** will save the record still when the index information is duplicated in another record. To verify the existence (or not) of a similar record saved previously, you will make use the SARIndexConDB$ or SARByteConDB$ functions, especialized functions that search record into you database. Other mode is to use SINEConvDB$ function, that save a record only if not exist previously in you database.

WARNING: Use this function only for massive entry data that not require search duplicated records in database. Inmediately end the data entry, call the ReindexConvDB$ function to force the file reindexation

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordToSave$	**RecordToSave$** variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use **UDT** structures. **Example:** DatabaseRecord.Apellido = Ponce 'Surname DatabaseRecord.Nombres = Richard 'Names DatabaseRecord.Email = rponce@lanet.com.ar DatabaseRecord.Web = http://www.hepika.com.ar RecordToSave$ = DatabaseRecord 'assign the record fields saved in 'DatabaseRecord UDT into 'RecordToSave$ variable for call 'packed function

Example english function:

```
DatabaseRecord.Apellido="BBB - ultimo"
DatabaseRecord.Nombres ="bebe"
DatabaseRecord.Email   ="bebe@rapidq.com.ar
DatabaseRecord.Web     ="www.bebe.com.ar"

RecordToSave$=DatabaseRecord   'assign the UDT to RecordToSave$
                               'variable to call the packed function
```

```
DataBasePath$=”c:\pulsar”
DataBaseFile$=”Conventional_DB”

CallPulsar$=SaveConvDB$(PulsarPath$,PulsarFile$,RecordToSave$)

IF CallPulsar$<>"ok" THEN
   'If record is saved, the packed function return "ok".
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx..." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$
 ELSE
   'record is saved
   MSGBOX "OPERACION OK."
END IF
```

Example spanish function:

```
DatabaseRecord.Apellido="BBB - ultimo"
DatabaseRecord.Nombres ="bebe"
DatabaseRecord.Email   ="bebe@rapidq.com.ar
DatabaseRecord.Web     ="www.bebe.com.ar"

RegistroAGrabar$=DatabaseRecord   'assign the UDT to RecordToSave$
                                  'variable to call the packed function

Ubicacion$=”c:\pulsar”
Archivo$=”Conventional_DB”

CallPulsar$= GrabaBDDConv $ (Ubicación$,Archivo$,RegistroAGrabar$)

'Si no ocrrue ningún error, la función retorna "ok", en caso de error, retorna la cadena
'"ERROR CODE:XXX.Description_del_error" dentro de la variable CallPulsar$

IF CallPulsar$<>"ok" THEN
      MSGBOX "LA OPERACION NO FUE COMPLETADA:"+CallPulsar$
END IF
```

10.2.6 FORCED DATABASE REINDEXATION

English function

ReindexConvDB $
Format: a$= ReindexConvDB$(DataBasePath$,DataBaseFile$)
Return: “ok” if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

ReindexaBDDConv$
Formato: A$= ReidexaBDDConv$ (Ubicación$,Archivo$)
Retorna: “ok” si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

**ReindexConvDB$** packed function forced the database reindexation. This function must used after SaveConvDB$ (save a record in database an NOT reindex it)

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the

	file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**

Example english function:

```
DataBasePath$="c:\pulsar"
DataBaseFile$="Conventional_DB"

CallPulsar$=ReindexConvDB$(PulsarPath$,PulsarFile$)

IF CallPulsar$<>"ok" THEN
   'If database is reindexed, the packed function return "ok".
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$
ELSE
   MSGBOX "OPERACION OK."
END IF
```

Example spanish function:

```
Ubicacion$="c:\pulsar"
Archivo$="Conventional_DB"

CallPulsar$= ReindexaBDDConv $ (Ubicación$,Archivo$)

'Si no ocrrue ningún error, la función retorna "ok", en caso de error, retorna la cadena
'"ERROR CODE:XXX.Description_del_error" dentro de la variable CallPulsar$

IF CallPulsar$<>"ok" THEN
      MSGBOX "LA OPERACION NO FUE COMPLETADA:"+CallPulsar$
END IF
```

10.2.7 GET THE QUANTITY OF RECORD SAVED IN DATABASE

English function

GetQuantityOfRecords$
Format: a$= GetQuantityOfRecords$ (DataBasePath$,DataBaseFile$)
Return: if not error occur, return a number in string format
If error occur, function return error code in string format (see error section)

Spanish function

VerCantidadRegistros$
Formato: A$= VerCantidadRegistros$ (Ubicación$,Archivo$)
Retorna: si no ocurre ningun error retorna un numero en formato de cadena

Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

GetQuantityOfRecords$ function read the quantity of records saved in you database

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**

Example english function:

```
DataBasePath$="c:\pulsar"
DataBaseFile$="Conventional_DB"

RecordsInDB$=GetQuantityOfRecords$(FilePath$,FileNameDataBase$)

IF LCASE$(LEFT$(RecordsInDB$,5))="error" THEN

  'If not error occur, function return the quantity or records saved in
  'you database in string format
  'If error occur, return "ERROR CODE:XXX.Error_Description" string into
  'RecordsInDB$ variable
  MSGBOX "Error reading quantity of records saved: "+RecordsInDB$

ELSE

    'the RecordsInDB$ variable contain the quantity of record saved in
    'database
    MSGBOX "Records saved: "+RecordsInDB$

END IF
```

Example spanish function:

```
Ubicación$="c:\pulsar"
Archivo$="Conventional_DB"

RecordsInDB$= VerCantidadRegistros$ (Ubicacion$,Archivo$)

IF LCASE$(LEFT$(RecordsInDB$,5))="error" THEN
  'Si no ocrrue ningún error, la función retorna "ok", en caso de error, retorna la cadena
   ""ERROR CODE:XXX.Description_del_error" dentro de la variable RecordsInDB$
  MSGBOX "Error leyendo cantidad de registros grabados: "+RecordsInDB$

ELSE
```

```
'La variable RecordsInDB$ contiene la cantidad de registros grabados en la
'base de datos
MSGBOX "Registros grabados: "+RecordsInDB$

END IF
```

10.2.8 READ A RECORD IN DATABASE

English function

GetRecordConvDB$
Format: a$= GetRecordConvDB$ (DataBasePath$,DataBaseFile$,RecordNumberToRead$)
Return: if not error occur, return a complete record saved
If error occur, function return error code in string format (see error section)

Spanish function

VerRegistroBDDConv$
Formato: A$= VerRegistroBDDConv$ (Ubicación$,Archivo$,NumeroRegistroaLeer$)
Retorna: si no ocurre ningun error, retorna un registro completo
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

GetRecordConvDB$ allow read a record saved into you database.

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordNumberToRead$	RecordNumberToRead$ variable must contain a valid record number to read into database. This record number must not be greater than the last record saved neither smaller than 1 **Example:** RecordNumberToRead$ ="25"

Example english function:

```
DataBasePath$="c:\pulsar"
DataBaseFile$="Conventional_DB"
RecordNumberToRead$="15"
```

```
CallPulsar$= GetRecordConvDB$ (DataBasePath$,DataBaseFile$,RecordNumberToRead$)

IF LCASE$(LEFT$( CallPulsar$,5))="error" THEN

  'If not error occur, function return the record into CallPulsar$ variable
  'If error occur, return "ERROR CODE:XXX.Error_Description" string into
  ' CallPulsar$ variable
  MSGBOX "Error reading record: "+ CallPulsar$

ELSE

        'error no occur, assign the CallPulsar$ variable data to
        'DatabaseRecord UDT structure
        POKE$ VARPTR(DatabaseRecord), CallPulsar$

        'show the record
        Msg$=Msg$+"APELLIDO: "+DatabaseRecord.Apellido+CHR$(10)+CHR$(13)
        Msg$=Msg$+"NOMBRES: " +DatabaseRecord.Nombres+CHR$(10)+CHR$(13)
        Msg$=Msg$+"E-MAIL: "  +DatabaseRecord.Email+CHR$(10)+CHR$(13)
        Msg$=Msg$+"WEB: "     +DatabaseRecord.Web

        MSGBOX Msg$

END IF
```

Example spanish function:

```
Ubicación$=”c:\pulsar”
Archivo$=”Conventional_DB”
NumeroRegistroaLeer$=”15”

CallPulsar$= VerRegistroBDDConv$ (Ubicación$,Archivo$,NumeroRegistroaLeer$)

IF LCASE$(LEFT$( CallPulsar$,5))="error" THEN
  'Si no ocrrue ningún error, la función retorna dentro de CallPulsar$ al registro leido, en
   ‘caso de error, retorna la cadena
   ‘”ERROR CODE:XXX.Description_del_error" dentro de la variable RecordsInDB$
  MSGBOX "Error leyendo registro: "+ CallPulsar$

ELSE

        'no ocurre ningún error, se asigna el contenido de CallPulsar$ a
        'la estructura UDT DatabaseRecord
        POKE$ VARPTR(DatabaseRecord), CallPulsar$

        'muestra el registro leído
        Msg$=Msg$+"APELLIDO: "+DatabaseRecord.Apellido+CHR$(10)+CHR$(13)
        Msg$=Msg$+"NOMBRES: " +DatabaseRecord.Nombres+CHR$(10)+CHR$(13)
        Msg$=Msg$+"E-MAIL: "  +DatabaseRecord.Email+CHR$(10)+CHR$(13)
        Msg$=Msg$+"WEB: "     +DatabaseRecord.Web

        MSGBOX Msg$

END IF
```

10.2.9 SAVE A RECORD ONLY IF NOT EXIST PREVIOUSLY

English function

SINEConvDB$
Format: a$= SINEConvDB$(DataBasePath$,DataBaseFile$,RecordToSave$)
Return: "ok" if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

GSNEBDDConv$
Formato: A$= GSNEBDDConv$(Ubicacion$,Archivo$,RecordToSave$)
Retorna: "ok" si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

SINEConvDB$ is **S**ave **I**f **N**ot **E**xist previously in **Conv**encional **D**ata**B**ase

SINEConvDB$ function allow save a record and assure it that this record is not saved prevously. Only save the record if the data contain into index file is not find between the saved records of you database

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordToSave$	**RecordToSave$** variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use **UDT** structures. RecordToSave$ variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use UDT structures. **Example:** DatabaseRecord.Apellido = Ponce 'Surname DatabaseRecord.Nombres = Richard 'Names DatabaseRecord.Email = rponce@lanet.com.ar DatabaseRecord.Web = http://www.hepika.com.ar RecordToSave$ = DatabaseRecord 'assign the record fields saved in 'DatabaseRecord UDT into 'RecordToSave$ variable for call 'packed function

Example english function:

```
DatabaseRecord.Apellido= "Rosas"          '¿is this record (Rosas) saved?
DatabaseRecord.Nombres = "Ezequiel Edelmiro"
DatabaseRecord.Email   = "ezequiel_ponce@rapidq.com.ar
DatabaseRecord.Web     = "www.ezequiel.com.ar"

RecordToSave$=DatabaseRecord   'assign the UDT to RecordToSave$
                               'variable to call the packed function
DataBasePath$="c:\pulsar"
DataBaseFile$="Conventional_DB"

CallPulsar$=SINEConvDB$(PulsarPath$,PulsarFile$,RecordToSave$)

IF CallPulsar$<>"ok" THEN
   'If record is saved, the packed function return "ok".
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

   'NOTE:If record is not saved because exist in you database, SINEConvDB$
   'function return the 123 code error: "CANNOT SAVE RECORD BECAUSE RECORD
   'EXIST IN DATABASE"

ELSE

   'record is saved
   MSGBOX "OPERACION OK."

END IF
```

Example spanish function:

```
DatabaseRecord.Apellido= "Rosas"          '¿ el registro (Rosas) ya está  grabado?
DatabaseRecord.Nombres = "Ezequiel Edelmiro"
DatabaseRecord.Email   = "ezequiel_ponce@rapidq.com.ar
DatabaseRecord.Web     = "www.ezequiel.com.ar"

RecordToSave$=DatabaseRecord   'asigna el contenido de la estructura UDT a RecordToSave$
                               'para llamar a la funcion
Ubicacion$="c:\pulsar"
Archivo$="Conventional_DB"

CallPulsar$= GSNEBDDConv$(Ubicacion$,Archivo$,RecordToSave$)

IF CallPulsar$<>"ok" THEN
   'Si no existia previamente y fue grabado, la función retorna  "ok".
   'si ocurre algun error, la funcion retorna un error en formato de cadena
   '"ERROR CODE:xxx...." (xxx es el codigo de error –vea los codigos de error -)
   MSGBOX "OPERACION NO COMPLETADA: "+CallPulsar$

   'NOTE:If record is not saved because exist in you database, SINEConvDB$
   'function return the 123 code error: "CANNOT SAVE RECORD BECAUSE RECORD
   'EXIST IN DATABASE"

ELSE

   'el registro no existía previamente, fue grabado.
   MSGBOX "OPERACION OK."

END IF
```

10.2.10 SEARCH A RECORD IN YOU DATABASE AND RETURN INDEX NUMBER

English function

SARIndexConDB$
Format: a$= SARIndexConDB$(DataBasePath$,DataBaseFile$,DataToSearch$)
Return: if not error occur, return a number in string format
If error occur, function return error code in string format (see error section)

Spanish function

BYRIndexBDDCon$
Formato: A$= BYRIndexBDDCon$(Ubicacion$,Archivo$,DataToSearch$)
Retorna: si no ocurre ningun error, retorna un numero en formato de cadena
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

SARIndexConDB$ is **S**earch record **A**nd **R**eturn **Index** in **Conv**encional **D**ata**B**ase

SARIndexConDB$ function allow search a record into you database. If record is find return the record number in string format. If record is not find, return error code.

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
DataToSearch$	To search information inside database, you should enter through the DataToSearch$ variable the string to look for. The search is perfomed into the index field that order you database and the system tries to find the first coincidence with the searched string. Remember that the search only is efficient when you database is ordered (indexed). If previously you has entered data through the save without index function, you should call the reindexacion function before search data **Examples:** **DataToSearch $ = "Rosas"** **DataToSearch $ = "Ros"** **DataToSearch $ = "R"** If ***Púlsar*** finds a coincidence, will return the record number (in **string format**) that contains the first coincidence.

Example english function:

```
DataBasePath$=”c:\pulsar”
DataBaseFile$=”Conventional_DB”

DataToSearch$="B"   ' search the first record that begin with “B”

CallPulsar$=SARIndexConDB$(PulsarPath$,PulsarFile$,DataToSearch$)

IF LEFT$(CallPulsar$,5)="ERROR" THEN

   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

ELSE

   MSGBOX "RECORD CONTAIN THE DATA: Record number "+CallPulsar$

END IF
```

Example spanish function:

```
Ubicacion$=”c:\pulsar”
Archivo$=”Conventional_DB”

DataToSearch$="B"   'busca el primer registro que empiece con “B”

CallPulsar$= BYRIndexBDDCon$(Ubicacion$,Archivo$,DataToSearch$)

IF LEFT$(CallPulsar$,5)="ERROR" THEN

   'si ocurre algún error, retorna una cadena conteniendo codigo y descripcion del error
   ' "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERACION NO COMPLETADA: "+CallPulsar$

ELSE

   MSGBOX "ENCONTRADO EN REGISTRO NUMERO: "+CallPulsar$

END IF
```

10.2.11 SEARCH A RECORD IN YOU DATABASE AND RETURN RECORD INITIAL BYTE IN DATA FILE

English function

SARByteConDB$
Format: a$= SARByteConDB$(DataBasePath$,DataBaseFile$,DataToSearch$)
Return: if not error occur, return a number in string format
If error occur, function return error code in string format (see error section)

Spanish function

BYRByteBDDCon$
Formato: A$= BYRByteBDDCon$(Ubicacion$,Archivo$,DataToSearch$)
Retorna: si no ocurre ningun error, retorna un numero en formato de cadena
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

SARByteConDB$ is **S**earch record **A**nd **R**eturn initial **Byte** in **Conv**encional **D**ata**B**ase

SARByteConDB$ function allow search a record into you database. If record is find return the initial byte in data file to manipulate it obviating púlsar intervention

¡Warning!: The Púlsar record manipulation contemplate the network permissions access. Through this method you expose the data at corruption or loss.

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
DataToSearch$	To search information inside database, you should enter through the DataToSearch$ variable the string to look for. The search is perfomed into the index field that order you database and the system tries to find the first coincidence with the searched string. Remember that the search only is efficient when you database is ordered (indexed). If previously you has entered data through the save without index function, you should call the reindexacion function before search data **Examples:** **DataToSearch $ = "Rosas"** **DataToSearch $ = "Ros"** **DataToSearch $ = "R"** If ***Púlsar*** finds a coincidence, will return the the database binary file initial byte (in **string format**) witch contains the record searched.

Example english function:

```
DataBasePath$="c:\pulsar"
DataBaseFile$="Conventional_DB"

DataToSearch$="B"   ' search the first record that begin with "B"

CallPulsar$=SARByteConDB$(PulsarPath$,PulsarFile$,DataToSearch$)

IF LEFT$(CallPulsar$,5)="ERROR" THEN

   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

ELSE
```

```
   MSGBOX "RECORD CONTAIN THE DATA: Record number "+CallPulsar$

END IF
```

Example spanish function:

```
Ubicacion$=”c:\pulsar”
Archivo$=”Conventional_DB”

DataToSearch$="B"   'busca el primer registro que empiece con “B”

CallPulsar$= BYRByteBDDCon$(Ubicacion$,Archivo$,DataToSearch$)

IF LEFT$(CallPulsar$,5)="ERROR" THEN

   'si ocurre algún error, retorna una cadena conteniendo codigo y descripcion del error
   ' "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERACION NO COMPLETADA: "+CallPulsar$

ELSE

   MSGBOX "ENCONTRADO EN REGISTRO NUMERO: "+CallPulsar$

END IF
```

10.2.12 RETURN RECORD INITIAL BYTE

English function

ReturnRecordByte$
Format: a$= ReturnRecordByte$ (DataBasePath$,DataBaseFile$,RecordNumber$)
Return: if not error occur, return a number in string format
If error occur, function return error code in string format (see error section)

Spanish function

RetornaByteRegistro$
Formato: A$= RetornaByteRegistro$(Ubicacion$,Archivo$,RecordToSearch$)
Retorna: si no ocurre ningun error, retorna un numero en formato de cadena
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

ReturnRecordByte$ is **Return Record** initial **Byte** in data file.

ReturnRecordByte$ function allow get the record initial byte in data file to manipulate it obviating púlsar intervention. User must be enter the record number.

¡Warning!: The Púlsar record manipulation contemplate the network permissions access. Through this method you expose the data at corruption or loss.

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = “c:\pulsar\ficheros” or** **DataBasePath$ = “c:\pulsar\ficheros\”**

DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordNumber$	RecordNumber$ variable must contain a valid record number to read into database. This record number must not be greater than the last record saved neither smaller than 1 **Example:** RecordNumber$ ="15"

Example english function:

```
DataBasePath$="c:\pulsar"
DataBaseFile$="Conventional_DB"

RecordNumber$="15"   ' search the 15 record saved in database

ByteInicio$=ReturnRecordByte$(PulsarPath$,PulsarFile$, RecordNumber$)

IF LEFT$(ByteInicio$,5)="ERROR" THEN

   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERATION IS NOT COMPLETED: "+ ByteInicio$

END IF

'if error not occur, the ByteInicio$ variable contain in
'string format the byte number

File$="c:\pulsar\Conventional_DB"+".PLS"  'this is the púlsar data file in this example

'open the púlsar data file and read the record
CLOSE #75
OPEN File$ FOR BINARY ACCESS READ WRITE LOCK READ WRITE AS #75
GET #75, VAL(ByteInicio$),DatabaseRecord  'save the record into
CLOSE #75                          'DatabaseRecord UTD structure

'show record readed
Msg$=Msg$+DatabaseRecord+CHR$(10)+CHR$(13)
MSGBOX Msg$
```

Example spanish function:

```
Ubicacion$="c:\pulsar"
Archivo$="Conventional_DB"

RecordNumber$="15"   ' buscar el byte inicial del registro 15

ByteInicio$= RetornaByteRegistro$(Ubicacion$,Archivo$, RecordNumber$)

IF LEFT$(ByteInicio$,5)="ERROR" THEN
```

```
   'si ocurre un error, la funcion retorna un string conteniendo informacion del
   'error en este formato
   '"ERROR CODE:xxx...." (xxx es el codigo de error – vea el manual -)
   MSGBOX "OPERACION NO PUDO SER COMPLETADA: "+ ByteInicio$

END IF

'si no ocurre ningun error, la variable ByteInicio$ contiene
'en forma de cadena el byte inicial

File$="c:\pulsar\Conventional_DB"+".PLS"  'en este ejemplo, este es el archivo de datos

'abrir en modo binario el archivo de datos para leer el registro directamente sin intervencion
'del manager
CLOSE #75
OPEN File$ FOR BINARY ACCESS READ WRITE LOCK READ WRITE AS #75
GET #75, VAL(ByteInicio$),DatabaseRecord  'save the record into
CLOSE #75                        'DatabaseRecord UTD structure

'mostrar el registro leído
Msg$=Msg$+DatabaseRecord+CHR$(10)+CHR$(13)
MSGBOX Msg$
```

10.2.13 MODIFY A RECORD SAVED IN DATABASE

English function

ModifyRecordConvDB$
Format
a$=ModifyRecordConvDB$(DataBasePath$,DataBaseFile$,NewRecordInDB$,RecordToModify$)
Return: "ok" if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

GrabaBDDConv $
Formato:
A$= ModificaRecordBDDConv$(Ubicacion$,Archivo$,NewRecordInDB$,RecordToModify$)
Retorna: "ok" si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

ModifyRecordConvDB$ packed function allow modify the data contain in a record saved in you database. If you modify data contain into index field, this function reordering automatically you database.

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database

	file. **Example:** **DataBaseFile$ = "CLIENTES"**
NewRecordInDB$	NewRecordInDB$ variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use **UDT** structures. **Example:** DatabaseRecord.Apellido = Ponce 'Surname DatabaseRecord.Nombres = Richard 'Names DatabaseRecord.Email = rponce@lanet.com.ar DatabaseRecord.Web = http://www.hepika.com.ar NewRecordInDB$ = DatabaseRecord 'assign the record fields saved in 'DatabaseRecord UDT into 'NewRecordInDB$ variable for call 'packed functio
RecordToModify$	RecordToModify$ variable must contain a valid record number to read into database. This record number must not be greater than the last record saved neither smaller than 1 **Example:** RecordNumber$ ="15"

Example english function:

```
PulsarPath$="c:\pulsar"
PulsarFile$="Conventional_DB"

DatabaseRecord.Apellido="Ponce (Modify)"
DatabaseRecord.Nombres ="Richard D."
DatabaseRecord.Email   ="richard@lanet.com.ar
DatabaseRecord.Web     ="www.hepika.com.ar/index.htm"

NewRecordToSave$=DatabaseRecord   'assign the UDT to NewRecordToSave$
                                  'variable to call the packed function

RecordToModify$="8"   'record number to modify, this number must not
                      'be greater than the last record saved neither
                      'smaller than 1

CallPulsar$=ModifyRecordConvDB$(PulsarPath$,PulsarFile$,_
                                NewRecordToSave$,RecordToModify$)

IF CallPulsar$<>"ok" THEN

   'If record is modified, the packed function return "ok".
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$
```

```
ELSE

    MSGBOX "RECORD WAS MODIFIED."

END IF
```

Example spanish function:

```
Ubicacion$="c:\pulsar"
Archivo$="Conventional_DB"

DatabaseRecord.Apellido="Ponce (Modify)"
DatabaseRecord.Nombres ="Richard D."
DatabaseRecord.Email   ="richard@lanet.com.ar
DatabaseRecord.Web     ="www.hepika.com.ar/index.htm"

NewRecordToSave$=DatabaseRecord   'asigna el contenido de la estructura UDT
                                  'a NewRecordToSave$ para llamar a la funcion

RecordToModify$="8"   'numero de registro a amodificar, este numero no puede ser menor
                      'a "1" ni mayor al último registro grabado en la base de datos

CallPulsar$=ModificaRecordBDDConv$(Ubicacion$,Archivo$,_
                                   NewRecordToSave$,RecordToModify$)

IF CallPulsar$<>"ok" THEN

   'si el registro es modificado, la funcion retorna "ok".
   'si ocurre algun error, retorna una cadena conteniendo el codigo y la descripción del error
   'en formato: "ERROR CODE:xxx...." (xxx es el codigo del error – vea el manual -)
   MSGBOX "OPERACION NO COMPLETADA: "+CallPulsar$

ELSE

    MSGBOX "EL REGISTRO FUE MODIFICADO."

END IF
```

10.2.14 DELETE A RECORD IN DATABASE

English function

DeleteRecordConvDB$
Format a$= DeleteRecordConvDB$(DataBasePath$,DataBaseFile$,RecordNumber$)
Return: "ok" if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

GrabaBDDConv $
Formato: a$= BorraRecordBDDConv$(Ubicacion$,Archivo$,RecordNumber$)
Retorna: "ok" si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

DeleteRecordConvDB$ allow delete a record in you database and reordering automatically the index table.

It requires to define the following variables:

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordNumber$	RecordNumber$ variable must contain a valid record number to read into database. This record number must not be greater than the last record saved neither smaller than 1 **Example:** RecordNumber$ ="25"

Example english function:

```
PulsarPath$="c:\pulsar"
PulsarFile$="Conventional_DB"

RecordToDelete$="8"   'record number to delete, this number must not
                      'be greater than the last record saved neither
                      'smaller than 1 (Use the GetQuantityOfRecords$
                      'packed function to know the last record)

CallPulsar$=DeleteRecordConvDB$(PulsarPath$,PulsarFile$,RecordToDelete$)

IF CallPulsar$<>"ok" THEN

    'If record is deleted, the packed function return "ok".
    'If error occur, function return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

ELSE

     MSGBOX "OPERACION OK."

END IF
```

Example spanish function:

```
Ubicacion$="c:\pulsar"
Archivo$="Conventional_DB"

RecordToDelete$="8"   'registro a borrar, el numero contenido en esta variable
                      'no debe ser menor a "1" ni mayor al último registro grabado
```

```
                              'en la base de datos

CallPulsar$= BorraRecordBDDConv$(Ubicacion$,Archivo$,RecordNumber$)

IF CallPulsar$<>"ok" THEN

   'si el registro es borrado, la funcion retorna "ok".
   'si ocurre algun error, retorna una cadena conteniendo el codigo y la descripción del error
   'en formato: "ERROR CODE:xxx...." (xxx es el codigo del error – vea el manual -)
   MSGBOX "OPERACION NO COMPLETADA: "+CallPulsar$

ELSE

    MSGBOX "EL REGISTRO FUE BORRADO."

END IF
```

10.2.15 GET NUMBER SAVED IN COUNTER FILE

English function

GetCounter$
Format a$= GetCounter$(PulsarPath$,PulsarFile$)
Return: if not error occur, return a number in string format
If error occur, function return error code in string format (see error section)

Spanish function

VerContador $
Formato: a$= VerContador$(Ubicacion$,Archivo$)
Retorna: si no ocurre ningun error, retorna un numero en formato de cadena
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

GetCounter$ allow obtain the actual number saved in a counter file and simultaneously increasy it in 1. If counter file not exist, this command create it automatically and save the 1 value as actual data.

It requires to define the following variables:

PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:** **PulsarPath$ = "c:\pulsar\ficheros" or** **PulsarPath$ = "c:\pulsar\ficheros\"**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = "COUNTER"**

Example english function:

```
PulsarPath$="c:\pulsar"
PulsarFile$="Counter"
```

```
CallPulsar$=GetCounter$(PulsarPath$,PulsarFile$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'If error occur, function return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

ELSE

      'If number saved into counter file is returned, CallPulsar$ variable contain it
      MSGBOX "COUNTER ACTUAL:"+CallPulsar$

END IF
```

Example spanish function:

```
Ubicacion$=”c:\pulsar”
Archivo$=”Counter”

CallPulsar$= VerContador$(Ubicacion$,Archivo$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'si ocurre algun error, la funcion retorna una cadena con el codigo de
    ‘error y su descripcion
    "'ERROR CODE:xxx...." (xxx es el codigo de error- vea el manual-)
    MSGBOX "OPERATION NO FUE COMPLETADA: "+CallPulsar$

ELSE

      'Si fue leido, el valor del contador ha sido retornado dentro de la variable CallPulsar$
      MSGBOX "VALOR ACTUAL:"+CallPulsar$

END IF
```

10.2.16 SAVE A NUMBER IN COUNTER FILE

English function

SetCounter$
Format a$= SetCounter$(PulsarPath$,PulsarFile$,NumberToSet$)
Return: “ok” if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

SetearContador$
Formato: a$= SetarContador$(Ubicacion$,Archivo$,NumberToSet$)
Retorna: “ok” si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

SetCounter$ allow save a value in counter file between 1 and 99.999.999. If counter file not exist, create it automatically and save the selected number.

It requires to define the following variables:

PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created.

	Example: **PulsarPath$ = “c:\pulsar\ficheros” or** **PulsarPath$ = “c:\pulsar\ficheros\”**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = “COUNTER”**
NumberToSet$	NumberToSet$ must contain a number between 1 and 99.999.999. **Example:** **NumberToSet$ = “425”**

Example english function:

```
PulsarPath$=”c:\pulsar”
PulsarFile$=”Counter”

NumberToSet$= “1001”

CallPulsar$=SetCounter$(PulsarPath$,PulsarFile$, NumberToSet$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'If error occur, function return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

ELSE

      'If number  is saved, function return “ok”
      MSGBOX "COUNTER IS SETTED"

END IF
```

Example spanish function:

```
Ubicacion$=”c:\pulsar”
Archivo$=”Counter”

NumberToSet$=”1001”

CallPulsar$= SeteaContador$(Ubicacion$,Archivo$, NumberToSet$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'si ocurre algun error, la funcion retorna una cadena con el codigo de
    ‘error y su descripcion
    "'ERROR CODE:xxx...." (xxx es el codigo de error- vea el manual-)
    MSGBOX "OPERATION NO FUE COMPLETADA: "+CallPulsar$

ELSE
```

```
'Si el numero fue grabado, la funcion retorna “ok”
MSGBOX "CONTADOR SETEADO"

END IF
```

10.2.17 SAVE A COMMENT IN LOG FILE

English function

SaveInLog$
Format a$= SaveInLog$(PulsarPath$,PulsarFile$,CommentToSave$)
Return: “ok” if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

GrabaEnLog$
Formato: a$= GrabaEnLog$(Ubicacion$,Archivo$, CommentToSave$)
Retorna: “ok” si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

SaveInLog$ allow save a comment text into LOG file.

It requires to define the following variables:

PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:** **PulsarPath$ = “c:\pulsar\ficheros” or** **PulsarPath$ = “c:\pulsar\ficheros\”**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = “LogFile”**
CommentToSave$	CommentToSave$ must contain a comment in string format. **Example:** **CommentToSave$ = “Operation completed.”**

Example english function:

```
PulsarPath$=”c:\pulsar”
PulsarFile$=”LogFile”

CommentToSave$= “This the comment saved in LOG file”

CallPulsar$= SaveInLog$( PulsarPath$,PulsarFile$,CommentToSave$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN
```

```
    'If error occur, function return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

ELSE

    'If comment  is saved, function return “ok”
    MSGBOX "COMMENT SAVED"

END IF
```

Example spanish function:

```
Ubicacion$=”c:\pulsar”
Archivo$=”LogFile”

CommentToSave$=”Este es el comentario a grabar”

CallPulsar$= GrabaEnLog$(Ubicacion$,Archivo$, CommentToSave$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'si ocurre algun error, la funcion retorna una cadena con el codigo de
    ‘error y su descripcion
    "'ERROR CODE:xxx...." (xxx es el codigo de error- vea el manual-)
    MSGBOX "OPERATION NO FUE COMPLETADA: "+CallPulsar$

ELSE

    'Si el comentario fue grabado, la funcion retorna “ok”
    MSGBOX "COMENTARIO GRABADO"

END IF
```

10.2.18 READ THE COMMENTS SAVED IN A LOG FILE

English function

ReadLog$
Format a$= ReadLog$(PulsarPath$,PulsarFile$)
Return: if not error occur, return comments in string format
If error occur, function return error code in string format (see error section)

Spanish function

VerContador$
Formato: a$= LeeLog$ (Ubicacion$,Archivo$)
Retorna: si no ocurre ningun error, retorna los comentarios en formato de cadena
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

ReadLog$ allow read all comments saved in a LOG file and return in string fromat.

It requires to define the following variables:

PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:**

	PulsarPath$ = “c:\pulsar\ficheros” or **PulsarPath$ = “c:\pulsar\ficheros\”**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = “LogFile”**

Example english function:

```
PulsarPath$=”c:\pulsar”
PulsarFile$=”LogFile”

CallPulsar$= ReadLog$(PulsarPath$,PulsarFile$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'If error occur, function return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

ELSE

      'If comments are returned, CallPulsar$ variable contain its
      MSGBOX "ALL COMMENTS SAVED:"+CallPulsar$

END IF
```

Example spanish function:

```
Ubicacion$=”c:\pulsar”
Archivo$=”LogFile”

CallPulsar$= LeeLog$(Ubicacion$,Archivo$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'si ocurre algun error, la funcion retorna una cadena con el codigo de
    ‘error y su descripcion
    "'ERROR CODE:xxx...." (xxx es el codigo de error- vea el manual-)
    MSGBOX "OPERATION NO FUE COMPLETADA: "+CallPulsar$

ELSE

      'Si los comentarios son leídos, son retornados dentro de la variable CallPulsar$
      MSGBOX "VALOR ACTUAL:"+CallPulsar$

END IF
```

10.2.19 DELETE LOG FILE

English function

DeleteLog$
Format a$= DeleteLog$(PulsarPath$,PulsarFile$)
Return: “ok” if not error occur
If error occur, function return error code in string format (see error section)

Spanish function

BorraLog$
Formato: a$= BorraLog$ (Ubicacion$,Archivo$)
Retorna: “ok” si no ocurre ningun error
Si algun error ocurre, retorna una cadena conteniendo el dodigo y la descripción del error (ver la seccion codigos de error)

DeleteLog$ allow kill the log file specified.

It requires to define the following variables:

PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:** **PulsarPath$ = “c:\pulsar\ficheros” or** **PulsarPath$ = “c:\pulsar\ficheros\”**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = “LOGFILE”**

Example english function:

```
PulsarPath$=”c:\pulsar”
PulsarFile$=”LogFile”

CallPulsar$= DeleteLog$(PulsarPath$,PulsarFile$)

IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'If error occur, function return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    MSGBOX "OPERATION IS NOT COMPLETED: "+CallPulsar$

ELSE

      'if log file is deleted, CallPulsar$ variable contain “ok”
      MSGBOX "LOG FILE IS DELETED:"+CallPulsar$

END IF
```

Example spanish function:

```
Ubicacion$=”c:\pulsar”
Archivo$=”LOGFILE”

CallPulsar$= BorraLog$(Ubicacion$,Archivo$)
```

```
IF LEFT$(CallPulsar$,5)=”ERROR” THEN

    'si ocurre algun error, la funcion retorna una cadena con el codigo de
    ‘error y su descripcion
    "'ERROR CODE:xxx...." (xxx es el codigo de error- vea el manual-)
    MSGBOX "OPERATION NO FUE COMPLETADA: "+CallPulsar$

ELSE

     'Si el archivo fue borrado, la variable CallPulsar$ contiene “ok”
     MSGBOX "ARCHIVO LOG FUE BORRADO:"+CallPulsar$

END IF
```

11. POWERBASIC “HEART” PROGRAMMERS SECTION

This section describe how operate ***Púlsar*** heart in **PowerBasic 6.x and 7.x** compilers through the “classic form” whit numeric command through representatives variables. If you desire program with the new packed function, please go to **10 SECTION** that describe how to do.

GENERALITIES

The classic form to operate ***Púlsar*** heart is to invoke ***Púlsar*** exportable function in DLL from compiler and send him a numeric operation command. To work with numeric commands can be hard, for that reason replace numeric codes is better. We recommend to replace them for descriptive variables that are easier of remembering.

We recommend to declare as global these operation variables in the following way:

```
GLOBAL PLS_CMD_GetEngine                AS INTEGER
GLOBAL PLS_CMD_CreateNewDatabase        AS INTEGER
GLOBAL PLS_CMD_GetDatabaseVersion       AS INTEGER
GLOBAL PLS_CMD_SaveRecordWithoutIndex   AS INTEGER
GLOBAL PLS_CMD_SaveRecordAndIndex       AS INTEGER
GLOBAL PLS_CMD_IndexDataBase            AS INTEGER
GLOBAL PLS_CMD_ModifyRecordSaved        AS INTEGER
GLOBAL PLS_CMD_GetQuantityRecordSaved   AS INTEGER
GLOBAL PLS_CMD_GetIndexRecordnumber     AS INTEGER
GLOBAL PLS_CMD_GetByteOfRecord          AS INTEGER
GLOBAL PLS_CMD_GetIndexIfExist          AS INTEGER
GLOBAL PLS_CMD_GetByteIfExist           AS INTEGER
GLOBAL PLS_CMD_DeleteIndexRecord        AS INTEGER
GLOBAL PLS_CMD_GetDataBaseStructure     AS INTEGER
GLOBAL PLS_CMD_GetCounter               AS INTEGER
GLOBAL PLS_CMD_SetCounter               AS INTEGER
```

Once declared, is necessary to assign them the operation codes in this way:

```
PLS_CMD_GetEngine%              = 0
PLS_CMD_CreateNewDatabase%      = 1
PLS_CMD_GetDatabaseVersion%     = 2
PLS_CMD_SaveRecordWithoutIndex% = 5
PLS_CMD_SaveRecordAndIndex%     = 3
PLS_CMD_IndexDataBase%          = 4
PLS_CMD_ModifyRecordSaved%      = 15
PLS_CMD_GetQuantityRecordSaved% = 10
PLS_CMD_GetIndexRecordnumber%   = 11
PLS_CMD_GetByteOfRecord%        = 12
PLS_CMD_GetIndexIfExist%        = 6
PLS_CMD_GetByteIfExist%         = 7
```

```
PLS_CMD_DeleteIndexRecord%      = 8
PLS_CMD_GetDataBaseStructure%   = 13
PLS_CMD_GetCounter%             = 9
PLS_CMD_SetCounter%             = 14
```

These variables, when being declared as global, will be able to use in any sub or function in you program. This step is necessary because the variables represent the available commands to send a order to ***Púlsar***.

Finished this, you must to declare Púlsar exportable function that will allow him to work with their databases

```
DECLARE FUNCTION Pulsar LIB "pulsar.dll"(_
        PLS_command AS INTEGER, _          'Púlsar command
        PLS_path AS STRING, _              'database path
        PLS_file AS STRING, _              'databalse file
        PLS_param AS STRING, _             'command parameters
        PLS_indexfield AS STRING, _        'index field of database
        PLS_return AS STRING ) AS LONG     'return
```

The **PULSAR.DLL** file is the manager itself, and it should be located in the **Windows SYSTEM** directory. When you copies your applications (or configure the install program), remember that the **.DLL** file should be in that Windows's folder (and not other). If he doesn't make it in that way, the compiler will give him an error message when not finding the **DLL**.

THE NEW IN THIS VERSION

The ***Púlsar*** core operates in similar form to previous versions, but the **PLS_filenumBDD%** and **PLS_filenumIndex%** variables are not required now. Neither user's identification is required through the **PLS_USER$** variable. This modifications must be implement in their source code that operates under old ***Púlsar*** versions.

Although these modifications can be tedious, the new ***Púlsar*** architecture has forced us to implement them. The benefit, potentiality and versatility that means the packed functions implementation, are highly positive and benefit directly to **PowerBasic** programmers.

DATABASES FIELDS FORMAT

Púlsar operates the fields into database records as ***text fields***. You will transform the numeric information to save to string format with the purpose of the field is not rejected by the database manager. In the same way, when reading the record information, remember to transform the content of a string to numeric format.

PÚLSAR VARIABLES

As he can see, ***Púlsar*** is declared like a function that return a long integer. The variables in the function allowing to operate the principal function programming it. These variables allowing you send parameters and information into ***Púlsar*** DLL. In the same way that he can send information, ***Púlsar*** will also send return information.

Part of Information returned from ***Púlsar*** comes in through ***long integer*** form, and part can come in into **PLS_return$** variable that you can see, is a ***string***.

A correct ***Púlsar*** call is:

```
A& = Pulsar ( PLS_command%, PLS_path$, PLS_archivo$, PLS_param$, _
              PLS_indexfield$, PLS_return$ )
```

Where PLS_command% acquires a value and represent a numeric command. The others variables can acquire different values, according to the invoked command. The function will return a long integer (in this case is captured into the A& variable) and diverse messages that can be read through the PLS_return$ variable.

To interpret the *Púlsar* return codes, go to **ERROR CODES SECTION** in this manual, that describes the long interger returned and the corresponding PLS_return$ message.

We will see now a brief description of each *Púlsar* variables:

PLS_command% (INTERGER)

Represent the numeric command to operate *Púlsar*. Aquire a valid value between 1 and 15 and indicate to *Púlsar* a specific operation: create a new database, read, delete or modify a record, etc.

For example, when assigning him the 1 value, *Púlsar* will try to create a new database. The assignment can be made this way:

```
PLS_command% = 1
```

If you prefer to operate through representatives variables, can use this:

REPRESENTATIVE VARIABLE	NUMERIC COMMAND
PLS_CMD_GetEngine%.	Return the *Púlsar* version engine
PLS_CMD_CreateNewDatabase%	Create a new Conventional DataBase and initializes the manager to operate with the files.
PLS_CMD_GetDatabaseVersion%	Allow obtain the current *Púlsar* version of DLL installed in the SYSTEM Windows directory
PLS_CMD_SaveRecordWithoutIndex%	Allow saving a record in databases files whithout indexation.
PLS_CMD_SaveRecordAndIndex%	Allow saving a record in databases files and order the records according the index field.
PLS_CMD_IndexDatabase%	Index the records saved ordering according the index field.
PLS_CMD_GetIndexIfExist%	Search and return the first number record which field index's information are coincident with the data entered for the user.
PLS_CMD_ModifyRecordSaved%	Modify a record saved in database and ordering the database information reordering according the index field.
PLS_CMD_GetByteIfExist%	Search a record witch index's information are coincident with the data entered for the user and return the initial byte of database binary file witch contains the record's database.
PLS_CMD_GetQuantityRecordSaved%	Allow get the quantity of record saved in a *Púlsar* database
PLS_CMD_GetIndexRecordNumber%	Allow read a record saved in database.
PLS_CMD_GetByteOfRecord%	Search the record number saved in database and return the initial byte witch contains it in the binary file.
PLS_CMD_DeleteIndexRecord%	Allow delete a record saved in a database and proceed to automatic reindexation and compactation to order the index table.

With this variables, for example, instead to indicate create a new database in this way:

```
PLS_command% = 1
```

He makes it in this way:

```
PLS_command% = PLS_CMD_CreateNewDatabase%
```

This is a simple way to operate their database and avoids the human errors.

EXAMPLES:

```
PLS_command% = 14              or
PLS_command% = PLS_CMD_SetCounter%
```

PLS_path$ (STRING)

This variable will contain a valid name of path in which the file will be created. Indistinctly the path can finish or not with the " \ " character.If during the command execution the path defined by user doesn't exist, it is created automatically by ***Púlsar***.

EXAMPLES:

```
PLS_path$="c:\pulsar\databases\"
PLS_path$="c:\pulsar\databases"
PLS_path$=""
```

PLS_file$ (STRING)

The PLS_file$ variable will have a valid name for create a database file. Absolutely all commands use this variable and ***Púlsar*** ***doesn't assign a defect name***. ***Púlsar*** will assign automatically to generated files a defined extension and he will prepare the files to receive information.
This manager divides the database information in three files, one contains the information itself, and it is identified easily by the extension **.PLS**, other, identified by the extension **.IDX**, contains the information referred to the database structure and save in his interior the ordination table (or index table) hisself, finally, the memo fields are save in a file identified by **.PLM** estension.
The .IDX file is the main structure of ***Púlsar*** database, being constituted in the file that contains the structure, initialization data, tables and order method of its database. But at the same time, all information is recorded in the .PLS and .PLM files, in the .IDX file destruction event, you data will be preserved by the system.

EXAMPLES:

```
PLS_path$="c:\pulsar\databases\"
PLS_file$="clientes"
```

In this example, Púlsar opers with the "c:\pulsar\databases\clientes" database.

PLS_param$ (STRING)

This variable is used by some commands, and according to command used will have a content or another. In case a command doesn't require it, ***Púlsar*** ignores their content.

EXAMPLE:

```
PLS_param$="PEREZ ROBERTO"
PLS_param$=""
```

PLS_indexfield$ (STRING)

This variable is used by some commands, and according to command used will have a content or another. In case a command doesn't require it, ***Púlsar*** ignores their content.

EXAMPLES:

```
PLS_indexfield$="SURNAME"
PLS_indexfield$=""
```

PLS_return$ (STRING)

This variable is used in Púlsar to return operative information. It returns error messages, registration information, etc.

PÚLSAR'S CALL

A correct ***Púlsar*** call will define the necessary variables to proceed to operate with database, according to specific requeriments of each command:

```
PLS_command%        = PLS_CMD_GetQuantityRecordSaved%
PLS_path$           ="c:\pulsar\databases"
PLS_file$           ="clientes"
PLS_parame$         =""
PLS_indexfield$     =""
PLS_return$         =""

A&= Pulsar ( PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
             PLS_indexfield$, PLS_return$ )
```

In this example, ***Púlsar*** will return the quantity of records saved in the **"clientes"** database. The quantity of records saved is returned in the **PLS_return$** variable, in string format. If the **A&** variable return the zero value, the operation was completed without errors and **PLS_return$** variable contains the quantity of records in database.

USER IDENTIFICATION

Since 3.0 Lite and Full versions, user identification is not required.

USING USER DEFINED TYPES (UDT) IN YOUR PROGRAMS

To save and to read records ***Púlsar*** operate only with strings variables. When create a database, ***Púlsar*** verifies the exact correspondence among the bytes defined in record and the bytes entered by the user during the operations. If there is not an exact correspondence between the size of the string and the size of defined record, ***Púlsar*** rejects the operation to avoid data damages.

This way to work the information demands precision when entering the characters and a very special care when working with variables that don't contain the quantity defined characters for each field, for what you must implement verification routines and stuffing the data with empty spaces. A more precise and effective form to works is to use **U**sers **D**efined **T**ypes variables (**UDT**), in such way is the compiler the one that assumes the control tasks on the quantity of characters assigned for each field.

In this manual we will use in all examples a database that contains records formed by these fields:

FIELD NAME	*SIZE*
SURNAME	30 characters
NAME	40 characters
E-MAIL	20 characters
WEB	50 characters

It would be convenient to define the following UDT structure for our database:

```
TYPE Fields
   Surname     AS STRING * 30   'last name field
   Name        AS STRING * 40   'first name field
   Email       AS STRING * 20   'e-mail field
   Web         AS STRING * 50   'web field
END TYPE

GLOBAL DatabaseRecord AS Fields        'DatabaseRecord global variable allow
```

```
                                        'operate with database records
```

This way simplifies vastly the database records's operations and avoids human errors.

CONVENTIONS FOR COMMANDS DESCRIPTION

For a better commands understanding avaiables for this manager, we will indicate for each one:

Command: Integer variable (%) that represents ta valid opetarion command

Number code: Numeric code that represents a valid operation operation and that is assigned tovariable that represents it.

Required variables: Variables whose value will be assigned by programmer and that they are required in an obligatory way. Without the appropriate values assignment, ***Púlsar*** will reject the operation.

Optional variables: Variables whose assignment is not obligatory to operate the invoked command. If they remain without assignment, ***Púlsar*** will assign them defect values. If they are assigned value, manager will work with them.

Not required variables: Variables that are ignored by the manager during command operation

Return variable: Variable used by ***Púlsar*** to return data after operation.

When during the description of an command operation the variable content is not described, and don't remember its functionality, appeal to the PULSAR FUNCTION VARIABLES in this section to obtain a detailed description.

11.1 RETURN PÚLSAR ENGINE VERSION

Commad: PLS_CMD_GetEngine%
Number code: 0

Required variables: Nothing
Return variable: PLS_return$ - Return the engine version

COMMAND

This command return into the **PLS_return$** variable (in string format), the ***Púlsar*** engine version installed in the system. In other words, return the DLL version installed in the Windows system directory. Allows to programmers prevent conflicts with yours applications that requires a specific Púlsar version. It doesn't generate screens neither interactions of any type, contrary to the **PLS_CMD_GetDatabaseVersion%** (command 2) that presents a dialog screen.

EXAMPLE

```
PLS_command%      = PLS_CMD_GetEngine%
PLS_path$         =""      'not required
PLS_file$         =""      'not required
PLS_param$        =""      'not required
PLS_indexfield$   =""      'not required
PLS_return$       =""      'not required

A&= Pulsar ( PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
             PLS_indexfield$, PLS_return$ )

IF a&=0 THEN
```

```
   MSG$="PLS_return$ return the Púlsar Engine version "
ELSE
   'operation is not completed ( return error description )
   MSG$= PLS_retorno$
END IF
```

11.2 CREATE NEW DATABASE

Create a new database and initializes to operate with the files.

Commad:	**PLS_CMD_CreateNewDatabase%**
Number code:	**1**
Required variables:	**PLS_command%** **PLS_file$** **PLS_param$**
Optional variables:	**PLS_path$** **PLS_FieldIndex$**
Not required variables:	**PLS_return$**
Return variable:	**PLS_return$ - RETURN TEXT MESSAGE**

COMMAND

This operation is the most basic and indispensable, because ***Púlsar*** won't allow him to operate with any file that has not been created previously through this command.
To create a database use **PSL_command%** variable in this way

PLS_command% = PLS_CMD_CreateNewDatabase% or
PLS_command% = 1

DATABASE FILE

The **PLS_file$** variable will have a valid name for create a database file:

PLS_file$ = "CLIENTES"

Púlsar will assign automatically to generated files a defined extension and he will prepare the files to receive information (view **9 SECTION** – **Púlsar Files**).

DATABASES PATH

The **PLS_path$** variable will contain a valid name of path in which the file will be created:

PLS_file$ = "c:\pulsar\ficheros" or
PLS_file$ = "c:\pulsar\ficheros\"

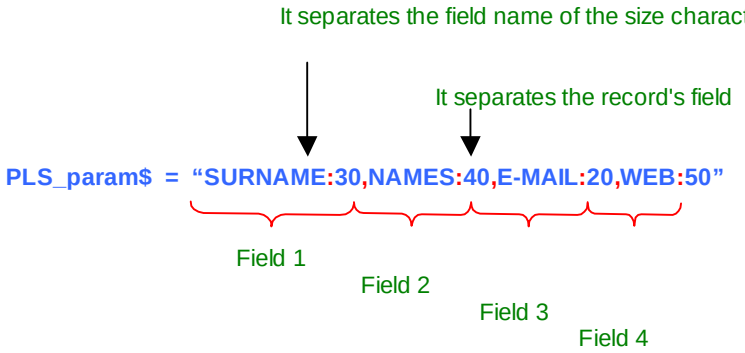
NEW DATABASES PARAMETERS

To create a database you must indicate the format of their database through the **PLS_param$** variable. For example, if he wants to create a new database that contains the following fields:

NAME'S FIELD	*SIZE*
SURNAME	30 characters
NAME	40 characters
E-MAIL	20 characters
WEB	50 characters

You ill indicate through this variable in appropriate format each field name and size, separated by the two points character (":"). Among each field definition, it should interpose a comma character (",").

Let us see:



Observe the separater chars.

During the database creation, the **PLS_param$** variable will contain the field definition at least with a minimum of 1 char. They are not allowed name of repeated fields inside oneself database. ***Púlsar*** distinguishes between upper and lowercase, in such way the field names like "**telephone**" and "**TELEPHONE**" are coincident, and the manager will reject them.

The **names** which you identify each record field can only have up to **30 chars**, but their content can arrive to **999 characters**. If you define a field name with more than 30 chars, the manager will proceed to cut it and he will only read the first 30 chars. If you tries to assign to field's content more than 999 characters, ***Púlsar*** will assign him for defect the allowed maximum value (999).

NEW DATABASE'S INDEX

Púlsar orders the information in database through an index table (or index). This index is ordered according the content of one field. The field that defines the ordination in database will be defined through the **PLS_indexfield$** variable.

To define an index field in database you will enter in this variable the name field. Following our example, if he wants that the database is ordered through the E-MAIL field, you will indicate through the **PLS_indexfield$** assignment the database index field name:

PLS_indexfield$ = "E-MAIL"

He must make sure that the index field defined in this variable coincides exactly with one of the fields defined in database creation, or otherwise, you will receive an error message.
The index field definition is not obligatory. In case the **PLS_indexfield$** variable remains empty during the database inicializacion, ***Púlsar*** will assign as defect index field the first field defined in the **PLS_param$** variable, in our case, the defect index field will be made through the field **SURNAME**.

EXAMPLE: NEW DATABASE CREATION

```
PLS_command%        = PLS_CMD_CreatenewDatabase%

'Required params to identify the CLIENTES database
PLS_path$           ="c:\pulsar\databases"
PLS_file$           ="clientes"

'Field's structure of record in database (FIELD_NAME : LONG)
PLS_param$          = "SURNAME:30,NAMES:40,E-MAIL:20,WEB:50"

PLS_indexfield$     =""             'default index (surname)
PLS_return$         =""
```

```
A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
              PLS_indexfield$, PLS_return$  )

IF a&=0 THEN
   MSG$="OK. Púlsar create new Database"
ELSE
    ' error messages returned in PLS_retorno$
   MSG$= PLS_retorno$
END IF

MSGBOX MSG$          'show the message
```

11.3 GET THE CURRENT PÚLSAR VERSION

Allow obtain the current Púlsar DLL version installed in you system.

Command:	**PLS_CMD_GetDatabaseVersion%**
Number code:	**2**
Not required variables:	**PLS_command%** **PLS_file$** **PLS_param$** **PLS_path$** **PLS_filenumIndex%** **PLS_return$**
Return variable:	**PLS_return$ - RETURN TEXT MESSAGE**

COMMAND

To obtain the ***Púlsar*** current version you must program the **PSL_command%** variable in this way:

```
PLS_command% = PLS_CMD_GetDatabaseVersion%  or
PLS_command% = 2
```

The manager will return the ***Púlsar*** current version in a string format into the **PLS_return$** variable and present a screen.

EXAMPLE

```
PLS_command%      = PLS_CMD_GetDatabaseVersion%
PLS_path$         =""
PLS_file$         =""
PLS_param$        = ""
PLS_indexfield$   =""
PLS_return$       =""

A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
              PLS_indexfield$, PLS_return$  )

IF a&=0 THEN
   MSG$= PLS_return$
ELSE
    ' error messages is returned in PLS_return$
   MSG$= "ERROR: " + PLS_return$
END IF

MSGBOX MSG$          'show message
```

11.4 SAVING A RECORD IN DATABASE WHITHOUT INDEXATION

Allow saving a record in databases files whithout reindexation.

Command:	**PLS_CMD_SaveRecordWithoutIndex%**
Number code:	**5**
Required variables:	**PLS_command%** **PLS_file$** **PLS_param$**
Optional variables:	**PLS_path$**
Not required variables:	**PLS_indexfield$** **PLS_return$**
Return variable:	**PLS_return$ - RETURN TEXT MESSAGE**

COMMAND

To save a record, you must program the **PSL_command%** variable in this way:

```
PLS_command% = PLS_CMD_SaveRecordWithoutIndex%  or
PLS_command% = 5
```

RECORD TO SAVE

PLS_param$ variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use UDT structures.

In this example:

```
TYPE Fields
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
```

This way simplifies vastly the saving's operations, when assigning into the **PLS_param$** the record to save:

```
PLS_param$ = Record
```

SAVING THE RECORD

You remember that **PLS_CMD_SaveRecord%** command will save the record still when the index information is duplicated in another record. To verify the existence (or not) of a similar record saved previously, you will make use the **PLS_CMD_ReturnIndexIfExist%** or **PLS_CMD_ReturnByteIfExist%** commands, especialized commands that search record into you database. Other mode is to use **PLS_CMD_SaveOnlyIfNotExist%** command, that save a record only if not exist previously in you database.

WARNING: Use this command only for massive entry data that not require search duplicated records in database. Inmediately end the data entry, call the **PLS_CMD_IndexDataBase%** command to force the file reindexation

EXAMPLE

```
TYPE Fields
```

```
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
. . .

. . .
Record.Surname          =”Rosas”
Record.Names            =”Juan Alberto”
Record.Email            =”ja_rosas@supernet.com.ar”
Record.Web              =”http://www.jar.com.ar”

PLS_command%            = PLS_CMD_SaveRecordWithoutIndex%

‘Required params to identify the CLIENTES database
PLS_path$               =”c:\pulsar\databases”
PLS_file$               =”clientes”

PLS_param$              = Record        ‘assign the data fields record into param variable
PLS_indefield$          =””
PLS_return$             =””

        A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                      PLS_indexfield$, PLS_return$  )

        IF a&=0 THEN
           MSG$=”OK. Record saved”
        ELSE
            ‘error messages return in PLS_return$
           MSG$= PLS_retorno$
        END IF

        MSGBOX MSG$             ‘show the message
```

11.5 SAVING A RECORD IN DATABASE WITH REINDEXATION

Allow saving a record in databases files and reindex the records saved in database

Command:	**PLS_CMD_SaveRecordAndIndex%**
Number code:	**3**
Required variables:	**PLS_command%** **PLS_file$** **PLS_param$**
Optional variables:	**PLS_path$**
Not required:	**PLS_indexfield$** **PLS_return$**
Return variable:	**PLS_return$ - RETURN TEXT MESSAGE**

COMMAND

To save a record, you must program the **PSL_command%** variable in this way:

PLS_command% = PLS_CMD_SaveRecordAndIndex% or

```
PLS_command% = 3
```

RECORD TO SAVE

To save a record, enter the data using the PLS_param$ variable. Use UDT structures to avoid human errors.

```
TYPE Fields
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
```

This way simplifies the operations, assigning into the PLS_param$ the record to save:

```
PLS_param$ = Record
```

SAVING THE RECORD

You remember that PLS_CMD_SaveRecordAndIndex% command will save the record still when the index information is duplicated in another record. To verify the existence (or not) of a similar record saved previously, you will make use the PLS_CMD_ReturnIndexIfExist% or PLS_CMD_ReturnByteIfExist% commands, especialized commands that search record into you database. Other mode is to use PLS_CMD_SaveOnlyIfNotExist% command, that save a record only if not exist previously in you database.
Remember that this command save the record whit the automatic database ordering (indexation).

EXAMPLE

```
TYPE Fields
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
. . .

. . .
Record.Surname          =”Rosas”
Record.Names            =”Juan Alberto”
Record.Email            =”ja_rosas@supernet.com.ar”
Record.Web              =”http://www.jar.com.ar”

PLS_command%            = PLS_CMD_SaveRecordAndIndex%

‘Required params to identify the CLIENTES database
PLS_path$               =”c:\pulsar\databases”
PLS_file$               =”clientes”

PLS_param$              = Record        ‘assign the data fields record into parameter variable
PLS_indexfield$         =””             ‘not required
PLS_return$             =””

        A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                      PLS_indexfield$, PLS_return$  )

        IF a&=0 THEN
```

```
   MSG$="OK. Record saved and database indexed"
ELSE
    ' error message returned in PLS_return$
   MSG$= PLS_retorno$
END IF

MSGBOX MSG$          'show the message
```

11.6 FORCED DATABASE REINDEXATION

This command force the database reindexation.

Command:	**PLS_CMD_IndexDatabase%**
Number code:	**4**
Required variables:	**PLS_command%** **PLS_file$**
Not required variables:	**PLS_indexfield$** **PLS_param$** **PLS_path$** **PLS_return$**
Return variable:	**PLS_return$ - RETURN TEXT MESSAGE**

COMMAND

To forced the database reindexation, you must program the **PSL_command%** variable in this way:

```
PLS_command% = PLS_CMD_ IndexDatabase%  ó
PLS_command% = 4
```

EXAMPLE

```
PLS_command%= PLS_CMD_IndexDatabase%
'Required params to identify the CLIENTES database
PLS_path$="c:\pulsar\databases"
PLS_file$="clientes"

PLS_param$= ""              'Not required
PLS_indexfield$=""          'Not required
PLS_return$=""              'Not required

   A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                 PLS_indexfield$, PLS_return$  )

   IF a&=0 THEN
      MSG$="OK. Database re-indexed"
   ELSE
       ' error messages returned in PLS_return$
      MSG$= PLS_return$
   END IF

   MSGBOX MSG$          'presenta el mensaje
```

11.7 SEARCH A RECORD IN YOU DATABASE

Search and return the first number record which field index information are coincident with the user entered data.

Command:	**PLS_CMD_GetIndexIfExist%**
Number code:	**6**
Required variables:	**PLS_command%** **PLS_file$** **PLS_param$**
Optional variables:	**PLS_path$**
Not required:	**PLS_return$** **PLS_indexfield$**
Return variable:	**PLS_return$ - RETURN THE NUMBER RECORD**

COMMAND

To search information into your database, ***Púlsar*** must receive into the **PSL_command%** variable this order

PLS_command% = PLS_CMD_GetIndexIfExist% or
PLS_command% = 6

REQUIRED PARAMETERS TO BEGIN THE SEARCH

To search information inside database, you should enter through the **PLS_param$** variable string to look for. The search is perfomed into the index field that order you database and the system tries to find the first coincidence with the searched string.
Remember that the search only is efficient when you database is ordered (indexed). If previously you has entered data through the save without index command (**PLS_CMD_SaveRecord%**), you should call the reindexacion command before serach data

You can define the search in these ways:

```
Record.Surname    ="R"        'SEARCH the first record that begins with R
Record.Names      =""         'Not required
Record.Email      =""         'Not required
Record.Web        =""         'Not required
PLS_param$        =Record     'Assign the record to search
PLS_USER$         ="FREE"     ' user ID (freeware version)

PLS_command%      = PLS_CMD_ReturnIndexIfExist%

                  or . . .

Record.Surname    ="Ra"       'SEARCH the first record that begins with Ra
Record.Names      =""         'Not required
Record.Email      =""         'Not required
Record.Web        =""         'Not required
PLS_param$        =Record     'Assign the record to search
PLS_USER$         ="FREE"     ' user ID (freeware version)

PLS_command%      = PLS_CMD_ReturnIndexIfExist%
```

If Púlsar finds a coincidence, will return into the **PLS_Return$** variable the record number (in **string format**) that contains the first coincidence.

PÚLSAR RETURNED MESSAGES

When finish search, ***Púlsar*** will return into the PLS_return$ variable in **string format** the record number that contains the first coincidence. If NOT FINDING any coincidence, the "**no**" word in lowercase is return into the PLS_Return$ variable..

EXAMPLE

```
TYPE Fields
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
. . .

. . .
'Search the ROSAS surname in database

'Search the first record number in database which contain "ROSAS" data

Record.Surname      ="Rosas"        'the database is indexed by SURNAME field
Record.Names        =""             'not required to search in database
Record.Email        =""             'not required to search in database
Record.Web          =""             'not required to search in database

'Required params to identify the CLIENTES database
PLS_path$           ="c:\pulsar\databases"
PLS_file$           ="clientes"

PLS_param$          = Record        'Assign the Record to serch to PLS_param$ variable

PLS_command%        = PLS_CMD_GetIndexIfExist%

PLS_indexfield$     =""             'Not required
PLS_retorno$        =""             'Not required

        A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                      PLS_indexfield$, PLS_return$  )

        IF a&=0 THEN

          'search end without errors

          IF  PLS_return$="no" then
              MSG$="Púlsar not find the data solicit"
          ELSE
              MSG$="Data find in Record N°  " + PLS_return$
          END IF

        ELSE

            ' Error occur. Returned in PLS_return$

           MSG$= PLS_return$

        END IF

        MSGBOX MSG$    'Show the message
```

11.8 MODIFY A RECORD SAVED IN DATABASE

Modify a record saved in you database and reordering if data modified affect index field.

Command:	**PLS_CMD_ModifyRecordSaved%**
Number code:	**15**
Required variables:	**PLS_command%** **PLS_file$** **PLS_param$** **PLS_indexfield$**
Optional variables:	**PLS_path$**
Not required:	**PLS_return$**
Return variable:	**PLS_return$ - RETURN TEXT MESSAGE**

COMMAND

To modify a saved record, is necessary instruct the **PLS_command%** variable with the 15 command.

```
PLS_command% = PLS_CMD_ModifyRecordSaved%  or
PLS_command% = 15
```

RECORD TO MODIFY

To modify a record saved, ***Púlsar*** needs to receive a valid record number that must be especified into the **PLS_indexfield$** variable in this way:

```
PLS_indexfield$ = "14"
```

In this example, ***Púlsar*** will modify the 14º record.

DATA TO ENTER IN A RECORD MODIFICATION

When modifying a ***Púlsar*** record, the new record must be entered through the **PLS_param$** variable. Use UDT structure foe avoid human errors.

```
TYPE Fields
        Surname      AS  STRING * 30
        Names        AS  STRING * 40
        Email        AS  STRING * 20
        Web          AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
```

In this way they are vastly simplified the operations, when assigning to **PLS_param$** variable the information to modify:

```
PLS_param$ = Record
```

AUTOMATIC INDEXATION FOR INTELLIGENT RASTER

Púlsar, when modifying a recrord saved will verify automatically if the index field (that determines the database information order) has been modified. When the new record contains modifications in the field index, the manager will proceed immediately to reorder the database, without necessity that user order the reindexation.

EXAMPLE

```
TYPE Fields
        Surname      AS  STRING * 30
```

```
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
. . .

. . .
'Modify the ROSAS record

'Search the ROSAS record in database
Record.Surname      ="Rosas"        'the database is indexed by surname field
Record.Names        =""             'not required to search in database
Record.Email        =""             'not required to search in database
Record.Web          =""             'not required to search in database

'Required params to identify the CLIENTES database
PLS_path$           ="c:\pulsar\databases"
PLS_file$           ="clientes"

'Assign for SEARCH and MODIFY routines
PLS_return$         =""             'not required

PLS_param$          = Record        'Assign the record to search to param variable

PLS_command%        = PLS_CMD_ReturnIndexIfExist%

        'SEARCH THE "ROSAS" RECORD  . . .

        A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                      PLS_indexfield$, PLS_return$  )

IF PLS_retorno$ = "no" THEN

        'RECORD NOT FOUND
        MSGBOX "Record is not found in database"      'show message

ELSE

        'RECORD FOUND... modifing the data saved in "ROSAS" record...
        'The PLS_return$ variable contains  the record number which "ROSAS" data is saved
        '¡¡¡ The PLS_indexfield$ must be contain the record number to modify !!!

        PLS_indexfield$ = PLS_return$
        PLS_command%= PLS_CMD_ModifyRecordSaved%

        'MODIFING THE DATA RECORD. . .

        Record.Surname="Rosas"
        Record.Names="Juan Alberto XX"        'modify the NAMES field of record
        Record.Email="ja_rosas@supernet.com.ar"
        Record.Web="http://www.jar.com.ar"

        'SAVING THE MODIFICATION. . .

        A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                      PLS_indexfield$, PLS_return$  )

        IF a&=0 THEN

           'RECORD MODIFY AND SAVED
```

```
            MSG$="OK. Record is saved an modified. Database is reindexed."      'show the
message

        ELSE

            'PULSAR is not able to modify the record. The error message is returned in
PLS_return$
            MSG$= PLS_return$

        END IF

END IF
```

11.8 SEARCH A RECORD AND RETURN THE INITIAL BYTE IN PÚLSAR BINARY FILE

Search a record witch index field information are coincident with the data entered and return the database binary file initial byte witch contains the record searched.

If for any cause you need manipulate the data file that contain database records obviating ***Púlsar*** intervention, use this command to obtain the record location in the data file.

¡Warning!: The ***Púlsar*** record manipulation contemplate the network permissions access. Through this method you expose the data at corruption or loss

Command:	**PLS_CMD_GetBytelfExist%**
Number code:	**7**
Required variables:	**PLS_command%** **PLS_file$** **PLS_param$**
Optional variables:	**PLS_path$**
Not required variables:	**PLS_return$** **PLS_indexfield$**
Return variable:	**PLS_return$ - RETURN THE BYTE IN BINARY FILE**

COMMAND

The search is ordered through the **PSL_comand%** variable:

```
PLS_command% = PLS_CMD_GetBytelfExist %  ó
PLS_command% = 7
```

NECESSARY PARAMETERS IN SEARCH

The manager orders his information through the index field.

In a defined record:

```
TYPE Registro
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
```

```
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Registro
```

You can define the search in this way:

```
Record.Surname      =”Ra”       ‘SEARCH the first record that begins with RA
Record.Names        =””         ‘not required
Record.Email        =””         ‘not required
Record.Web          =””         ‘not required
PLS_param$          =Record     ‘Assignment the data search

PLS_command%        = PLS_CMD_GetByteIfExist %
```

If ***Púlsar*** finds a coincidence, will return into the PLS_return$ variable the byte number (in **string format**) in which the record is located in the data file. This byte number can be used by the programmers to manipulate the record opening data file in binary mode and to proceed to the direct data manipulation.

RETURNED MESSAGES

When finish the search, ***Púlsar*** will return into PLS_return$ variable the byte (in string format) in which the record begins. If ***Púlsar*** NOT FINDING any coincidence, the "**no**" word in lowercase is returned.

EXAMPLE

```
TYPE Fields
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
. . .

. . .
‘Search the ROSAS surname in database

‘Search the initial byte of ROSAS record in database binary file

Record.Surname      =”Rosas”    ‘database is indexed by the SURNAME field
Record.Names        =””         ‘not required to search in database
Record.Email        =””         ‘not required to search in database
Record.Web          =””         ‘not required to search in database

‘Required parameters to identify the CLIENTES database
PLS_path$           =”c:\pulsar\databases”
PLS_file$           =”clientes”

PLS_param$          = Record    ‘Assign to PLS_param$ variable the record to search

PLS_command%        = PLS_CMD_GetByteIfExist%

PLS_indexfield$     =””         ‘Not required
PLS_return$         =””         ‘Not required
```

```
A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
              PLS_indexfield$, PLS_return$  )

IF a&=0 THEN

  'Search end without errors

  IF  PLS_return$="no" then

     'RECORD NOT FOUND
     MSG$="Púlsar is unable to find the record"          'show message

  ELSE

     'RECORD FOUND  IN DATABASE BINARY FILE
     MSG$="Record search in Byte Nº  " + PLS_return$

     'OPEN THE CLIENTES BINARY FILE WITCH CONTAINS THE RECORD

     'PÚLSAR'S FILES:
     '      .PLS:    Binary file, witch contains the records of conventional database
     '      .IDX:    Index and structure file ( conventional databases )

    DatabaseFILE$= PLS_path$ + "\" + PLS_file$ + ".PLS"          'Púlsar database binary file

    OPEN DatabaseFILE$ FOR BINARY ACCESS READ WRITE LOCK READ WRITE AS #75
    GET #75, VAL (PLS_retorno$), Record          'read the record directly of the binary file
                                                 'and assign it to RECORD
     CLOSE #75
     'the information saved in record is contained in RECORD users's variable

     'this technique allow manipulate the records in database directly of Púlsar binary files
    'whitout intermediation of database manager

  END IF

ELSE

    ' Error message generated by search rutine

   MSG$= PLS_return$

END IF

MSGBOX MSG$          'show message
```

11.9 GET THE QUANTITY OF RECORD SAVED IN DATABASE

Allow get the quantity of record saved in a database.

Command:	**PLS_CMD_GetQuantityRecordSaved%**
Number code:	**10**
Required variables:	**PLS_command%** **PLS_file$**
Not required variables:	**PLS_path$** **PLS_param$** **PLS_filenumIndex%** **PLS_return$**

Return variable: **PLS_return$ - RETURN QUANTITY RECORDS SAVED**

COMMAND

To obtain the quantity of records saved in a database you must program the **PLS_command%** variable with the 10 command:

```
PLS_command% = PLS_CMD_GetQuantityRecordSaved %  or
PLS_command% = 10
```

The manager will return the quantity of records saved into the **PLS_return$** variable.

EXAMPLE

```
PLS_path$          =""          'default PATH
PLS_file$          ="clientes"  'CLIENTES database

PLS_command%       = PLS_CMD_GetQuantityRecordSaved%

PLS_param$         = ""         'not required
PLS_indexfile$     =""          'not required
PLS_return$        =""          'not required

    A&= Pulsar ( PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                 PLS_indexfield$, PLS_return$ )

    IF a&=0 THEN

      'No errors
      MSG$= "Quantity of record saved: "+PLS_return$

    ELSE

      'error messages are assigned to PLS_return$ variable
      MSG$= "ERROR. UNABLE TO OBTAIN THE QUANTITY OF RECORDS: "
      +PLS_return$

    END IF

    MSGBOX MSG$          'show the message
```

11.10 READ A RECORD IN DATABASE

Allow read a record saved in a database.

Command: **PLS_CMD_GetIndexRecordNumber%**
Number code: **11**

Required variables: **PLS_command%**
PLS_file$
PLS_param$

Not required variables: **PLS_path$**
PLS_indexfield$
PLS_return$

Return Variable: **PLS_return$ - RETURN SI o NO**

COMMAND

To read a record saved you must program the PLS_command% variable with the 11 command:

```
PLS_command% = PLS_CMD_GetIndexRecordNumber% or
PLS_command% = 11
```

The manager will return the record into the PLS_return$ variable.

EXAMPLE

```
'SEARCH A RECORD AND READ IT IF EXIST

TYPE Fields
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
. . .

. . .
'Search the ROSAS surname in database

'Search the record number of "ROSAS" data in database

Record.Surname          ="Rosas"        'the database is indexed by the SURNAME field
Record.Names            =""             'not required to search
Record.Email            =""             'not required to search
Record.Web              =""             'not required to search

'Parameters required to identify the CLIENTES database
PLS_path$               ="c:\pulsar\databases"
PLS_file$               ="clientes"

PLS_param$              = Record        'Assign the data to search to PLS_param$ variable

PLS_command%            = PLS_CMD_GetIdexIfExist%

PLS_indexfield$         =""             'not required
PLS_return$             =""             'not required

        A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                      PLS_indexfield$, PLS_return$  )

        IF a&=0 THEN

          'search whitout errors

          IF  PLS_return$="no" then

              MSG$="Púlsar is not able to search the record"

          ELSE

              'RECORD FOUND, the record number is assigned to PLS_return$

              PLS_param$= PLS_return$           'this variable contains the record number
              PLS_command%= PLS_CMD_GetIndexRecordNumber%

             'READ THE RECORD FIND
```

```
A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
              PLS_indexfield$, PLS_return$  )

 IF a&=0 THEN

    'record readed, the PLS_return$ variable contains the data

    'assign the PLS_return$ contain to RECORD variable
    POKE$ VARPTR(Record),PLS_return$

   MSG$="Púlsar find the record and assign it to RECORD"

  ELSE

    MSG$="Púlsar find the record, but is not able to read it:"+ PLS_return$

  END IF

 END IF

ELSE

  ' errors messages are returned in PLS_return$ variable

 MSG$= PLS_return$

END IF

MSGBOX MSG$          'show message
```

11.11 RETURN RECORD INITIAL BYTE

Return the recrord saved initial byte.

Command:	**PLS_CMD_GetByteOfRecord%**
Number code:	**12**
Required variable:	**PLS_command%** **PLS_file$** **PLS_param$**
Optional variables:	**PLS_path$**
Not required variables:	**PLS_return$** **PLS_indexfield$**
Return variable:	**PLS_return$ - RETURN THE INITIAL BYTE**

COMMAND

To search information , you must program the **PLS_command%** variable in this way:

```
PLS_command% = PLS_CMD_GetByteOfRecord%  or
PLS_command% = 12
```

NECESSARY PARAMETERS

The manager orders his information through the index field and facilitates him the search inside the database defining the record number to search through the **PLS_param$** variable.
This variable must not contain a record number bigger than the last record saved in your database.

You can define the search in this way:

```
Record.Surname      =""          'to return the initial byte of record this field is irrelevant
Record.Names        =""          'to return the initial byte of record this field is irrelevant
Record.Email        =""          'to return the initial byte of record this field is irrelevant
Record.Web          =""          'to return the initial byte of record this field is irrelevant
PLS_param$          ="12"        'Search the initial byte of 12 record

PLS_command%        = PLS_CMD_GetByteOfRecord%
```

RETURNED MESSAGES

When finish the search, ***Púlsar*** will return into **PLS_return$** variable the byte (in string format) in which the record begins. If record is not encountered the "**no**" word in lowercase is returned.

EXAMPLE

```
TYPE Fields
        Surname     AS  STRING * 30
        Names       AS  STRING * 40
        Email       AS  STRING * 20
        Web         AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
. . .

. . .
'Search the initial byte of 12º record

'Previously is convenient to secure it the quantity of record saved in database
'using the PLS_CMD_GetQuantityRecordSaved%

Record.Surname      =""          'not required
Record .Names       =""          'not required
Record.Email        =""          'not required
Record.Web          =""          'not required

'Required parameters to identify CLIENTES database
PLS_path$           ="c:\pulsar\databases"
PLS_file$           ="clientes"

PLS_Param$          ="12"        'the number record to search is 12

PLS_command%        = PLS_CMD_GetByteOfRecord%


PLS_indexfield$     =""          'Not required
PLS_retorno$        =""          'Not required

        A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                      PLS_indexfield$, PLS_return$  )

        IF a&=0 THEN

            'RECORD FOUND
             ' the initial byte of 12º record is returned in PLS_return$ variable

            MSG$= "The initial byte of 12º  record is returned in PLS_return$"

            'OPEN THE CLIENTES BINARY FILE WITCH CONTAINS THE RECORD
```

```
'PÚLSAR'S FILES:
'      .PLS:    Binary file, witch contains the records of conventional database
'      .IDX:    Index and structure file of conventional database

DatabaseFILE$= PLS_path$ + "\" + PLS_file$ + ".PLS"        'Púlsar database
                                                           'binary file

OPEN DatabaseFILE$ FOR BINARY ACCESS READ WRITE LOCK READ_
WRITE AS #75

GET #75, VAL (PLS_retorno$), Record          'read the record directly of the
                                             'binary file
                                             'and assign it to RECORD
 CLOSE #75
 'the information saved in record is contained in RECORD users's variable

MSG$= MSG$+chr$(13)
MSG$= MSG$+"The information saved in record is contained in RECORD user's variable"

 'this technique allow manipulate the records in database directly of Púlsar binary files
'whitout intermediation of database manager

ELSE

   'ERROR OCCURS. PÚLSAR IS NOT ABLE TO READ THE INITIAL BYTE."

  MSG$= PLS_return$

END IF

MSGBOX MSG$
```

11.12 DELETE A RECORD IN DATABASE

Allow delete a record saved in database and proceed to automatic reindexation and compactation to order the index table.

Command:	**PLS_CMD_DeleteIndexRecord%**
Number code:	**8**
Required variables:	**PLS_command%** **PLS_file$** **PLS_param$**
Optional variables:	**PLS_path$**
Not required variables:	**PLS_return$** **PLS_indexfield$**
Return variable:	**PLS_return$ - RETURN "SI" OR ERROR MESSAGE**

COMMAND

To delete a saved record a database you must program the **PLS_command%** variable in this way:

```
PLS_command% = PLS_CMD_DeleteIndexRecord% or
PLS_command% = 8
```

NECESSARY PARAMETERS

The PLS_param$ variable should contain a valid record number not bigger than the total records saved in you database.

RETURNED MESSAGES

When finishes the action, ***Púlsar*** will return the message "ok" in PLS_return$ variable and ***Púlsar*** function return zero value. If cannot delete the recod, an error message is received.

EXAMPLE

```
'SEARCH A RECORD AND DELETE IT

TYPE Fields
        Surname         AS  STRING * 30
        Names           AS  STRING * 40
        Email           AS  STRING * 20
        Web             AS  STRING * 50
END TYPE

GLOBAL   Record AS Fields
. . .

. . .
'Search the ROSAS record in database to delete it

Record.Surname          ="Rosas"        'the database is indexed by surname field
Record.Names            =""             'not required
Record.Email            =""             'not required
Record.Web              =""                     'not required

'Required parameters to identify the CLIENTES database
PLS_path$               ="c:\pulsar\databases"
PLS_file$               ="clientes"

PLS_param$              = Record                'Assign the record to PLS_param$ variable

PLS_command%            = PLS_CMD_GetIndexIfExist%

PLS_indexfield$         =""             'Not required
PLS_return$             =""             'Not required

        A&= Pulsar (  PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
                      PLS_indexfield$, PLS_return$  )

        IF a&=0 THEN

          'NO ERRORS. Púlsar function return zero value

          IF  PLS_return$="no" then

              'RECORD NOT FOUND

              MSG$="Púlsar is not able to find the record"

          ELSE

              'RECORD FOUND

              'The PLS_return$ variable contains the ROSAS number record
              PLS_param$= PLS_return$
              PLS_command%= PLS_CMD_DeleteIndexRecord%
```

```
'DELETE THE DATABASE RECORD

 A&= Pulsar ( PLS_command%, PLS_path$, PLS_file$, PLS_param$, _
              PLS_indexfield$, PLS_return$ )

 IF a&=0 THEN
    MSG$="Púlsar found the record and delete it"
 ELSE
    MSG$="Púlsar found the record but is unable to delete it: "+ PLS_return$
 END IF

END IF

ELSE

  ' messages returned by Púlsar
 MSG$= PLS_return$            'show message

END IF

MSGBOX MSG$
```

12. RAPID-Q PROGRAMMERS SECTION

The correct way to communicate with ***Púlsar*** is through the variables that requires each packed sub. The **RetFunction$** variable is the one that will return from Púlsar the information that you need. You must indicate to RapidQ that a variable is sent to a DLL but that the new value that DLL assigns to variable will be read by RapidQ. For this, you must add the @ char during call. This @ char indicates to RapidQ that the variable is shared between you program and Púlsar DLL.

RapidQ is a great compiler, but is unestable when he call a DLL This unestability should be anticipate by programmer following some simple rules. To call any packed sub, programmer should be careful to assign a character at least to all variables used in the subroutine. Assign a space at least to avoid a Windows error message. For example, if assign a variable in this way:

RecordToDelete $=""

you will give an error message. A correct way is this:

RecordToDelete $="_" (1 space char)

Another tip that you will keep in mind is that RapidQ is not flexible when assigning space to variables. ***Púlsar*** manages variables dynamically, but it is not the case in RapidQ. If you wait a ***Púlsar*** message through variables sendeds from RapidQ, ***Púlsar*** will return all the characters awaiting, but RapidQ will read only the quantity of characters that you assign to the variable. For example, if you want read a record of 100 bytes, but under RapidQ assigns it less chars:

RetFunction $= space$(5)

RapidQ will only read the first 5 chars that ***Púlsar*** sends him. Be generous when assigning variables that return him information from manager. Always assign enough characters. You will see that in all sample programs I assign 160 characters to **RetFunction$**. You can assign more or less, according you desire.

12.1 OPERATING PÚLSAR THROUGH PACKED SUBS

GENERALITIES

Púlsar is a DLL specially designed to operate with **PowerBasic**, although in fact, he can make it with any language that support DLL's as **Rapid-Q**.

First step consist in declare this packed subs into you program

```
DECLARE SUB RQ_PulsarEngine LIB "pulsar.dll" ALIAS "RQ_PulsarEngine"(RetFunction$)
DECLARE SUB RQ_PulsarBuild LIB "pulsar.dll" ALIAS "RQ_PulsarBuild"(RetFunction$)
DECLARE SUB RQ_NewConventionalDB LIB "pulsar.dll" ALIAS _
        "RQ_NewConventionalDB"(RetFunction$,Structure$,IndexField$,PulsarPath$,PulsarFile$)
DECLARE SUB RQ_SaveIndexConvDB LIB "pulsar.dll" ALIAS_
        "RQ_SaveIndexConvDB"(RetFunction$,PulsarPath$,PulsarFile$,RecordToSave$)
DECLARE SUB RQ_GetQuantityOfRecords LIB "pulsar.dll" ALIAS _
        "RQ_GetQuantityOfRecords"(RetFunction$,PulsarPath$,PulsarFile$)
DECLARE SUB RQ_GetRecordConvDB LIB "pulsar.dll" ALIAS _
        "RQ_GetRecordConvDB"(RetFunction$,PulsarPath$,PulsarFile$,RecordNumber$)
DECLARE SUB RQ_SINEConvDB LIB "pulsar.dll" ALIAS _
        "RQ_SINEConvDB"(RetFunction$,PulsarPath$,PulsarFile$,RecordToSave$)
DECLARE SUB RQ_SaveConvDB LIB "pulsar.dll" ALIAS _
        "RQ_SaveConvDB"(RetFunction$,PulsarPath$,PulsarFile$,RecordToSave$)
DECLARE SUB RQ_ReindexConvDB LIB "pulsar.dll" ALIAS _
        "RQ_ReindexConvDB"(RetFunction$,PulsarPath$,PulsarFile$)
DECLARE SUB RQ_ReturnRecordByte LIB "pulsar.dll" ALIAS _
        "RQ_ReturnRecordByte"(RetFunction$,PulsarPath$,PulsarFile$,RecordNumber$)
DECLARE SUB RQ_SARIndexConDB LIB "pulsar.dll" ALIAS _
        "RQ_SARIndexConDB"(RetFunction$,PulsarPath$,PulsarFile$,DataToSearch$)
DECLARE SUB RQ_SARByteConDB LIB "pulsar.dll" ALIAS _
        "RQ_SARByteConDB"(RetFunction$,PulsarPath$,PulsarFile$,DataToSearch$)
DECLARE SUB RQ_DeleteRecordConvDB LIB "pulsar.dll" ALIAS _
        "RQ_DeleteRecordConvDB"(RetFunction$,PulsarPath$,PulsarFile$,RecordNumber$)
DECLARE SUB RQ_ModifyRecordConvDB LIB "pulsar.dll" ALIAS _
        "RQ_ModifyRecordConvDB"(RetFunction$,PulsarPath$,PulsarFile$,NewRecordInDB$,_
        RecordToModify$)
DECLARE SUB RQ_GetCounter LIB "pulsar.dll" ALIAS "RQ_GetCounter"(RetFunction$,PulsarPath$,PulsarFile$)
DECLARE SUB RQ_SetCounter LIB "pulsar.dll" ALIAS _
        "RQ_SetCounter"(RetFunction$,PulsarPath$,PulsarFile$,NumberToSet$)
DECLARE SUB RQ_DeleteLog LIB "pulsar.dll" ALIAS "RQ_DeleteLog"(RetFunction$,PulsarPath$,PulsarFile$)
DECLARE SUB RQ_SaveInLog LIB "pulsar.dll" ALIAS _
        "RQ_SaveInLog"(RetFunction$,PulsarPath$,PulsarFile$,TextToSave$)
DECLARE SUB RQ_ReadLog LIB "pulsar.dll" ALIAS "RQ_ReadLog"(RetFunction$,PulsarPath$,PulsarFile$)
```

These declarations can see them in the included samples programs (view Pulsar30Lite.Inc file)

OPERATING PÚLSAR THROUGH STRINGS VARIABLES

As you can see, all subs are declared as strings, as well as all required variables. ***Púlsar*** operates only with string variables. If you tries to operate with numeric codes without to transform them before to string format, their compiler will return an error code.

The functions return different messages types (always in string format) when being invoked. If error occur, function return error code in this string format:

"ERROR CODE:xxx.Error Description"

Where **xxx** is a numeric error code and **Error Description** is the error description in corresponding language (see **ERROR CODES SECTION**).

DATABASES FIELDS FORMAT

Púlsar operates the fields into database records as ***text fields***. You will transform the numeric information to save to string format with the purpose of the field is not rejected by the database manager. In the same way, when reading the record information, remember to transform the content of a string to numeric format.

USING USER DEFINED TYPES (UDT) IN YOUR PROGRAMS

To save and to read records ***Púlsar*** operate only with strings variables. When create a database, ***Púlsar*** verifies the exact correspondence among the bytes defined in record and the bytes entered by the user during the operations. If there is not an exact correspondence between the size of the string and the size of defined record, ***Púlsar*** rejects the operation to avoid data damages.

This way to work the information demands precision when entering the characters and a very special care when working with variables that don't contain the quantity defined characters for each field, for what you must implement verification routines and stuffing the data with empty spaces. A more precise and effective form to works is to use **U**sers **D**efined **T**ypes variables (**UDT**), in such way is the compiler the one that assumes the control tasks on the quantity of characters assigned for each field.

In this manual we will use in all examples a database that contains records formed by these fields:

FIELD NAME	*SIZE*
APELLIDO	30 characters
NOMBRES	40 characters
E-MAIL	20 characters
WEB	50 characters

It would be convenient to define the following UDT structure for our database:

```
TYPE Fields
   Apellido     AS STRING * 30   'last name field
   Nombres      AS STRING * 40   'first name field
   Email        AS STRING * 20   'e-mail field
   Web          AS STRING * 50   'web field
END TYPE

GLOBAL DatabaseRecord AS Fields      'DatabaseRecord global variable allow
                                     'operate with database records
```

This way simplifies vastly the database records's operations and avoids human errors.

12.2 THE PACKED SUBROUTINES

12.2.1 PULSARBUILD

This sub return the ***Púlsar*** Build version installed in you system. In other words, return the DLL Build version installed in the Windows system directory. It allows to programmers prevent conflicts with yours applications that require a specific Púlsar version. It doesn't generate screens neither interactions of any type. This version return the “3011112002L” string.

Example:

```
'The RetFunction$ variable return in all pulsar subs the data
'required by programmers

RetFunction$=SPACE$(11) 'in RQ_PulsarBuild call the RetFunction
                        'variable must be defined as this chars lenght

RQ_PulsarBuild @RetFunction$

Msg$="Púlsar Build: "+RetFunction$+"."
MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
```

```
END
```

12.2.2 PULSARENGINE

This sub return the ***Púlsar*** Engine version installed in you system. This sub prevent conflicts with yours applications that require a specific Púlsar compiled version. This version return the "3.0L" string.

Example:

```
'The RetFunction$ variable return in all pulsar subs the data
'required by programmers

RetFunction$=SPACE$(4) 'in RQ_PulsarEngine call the RetFunction
                       'variable must be defined as 4 chars lenght

RQ_PulsarEngine @RetFunction$

IF RetFunction$ <>"3.0L" THEN
  MESSAGEBOX("This program require Púlsar 3.0 Lite installed in you system.", "Púlsar 3.0 Lite", mbOK)
  END
END IF

Msg$="Púlsar Engine: "+RetFunction$+"."
MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)

END
```

12.2.3 CREATE NEW DATABASE

RQ_NewConventionalDB

Format:

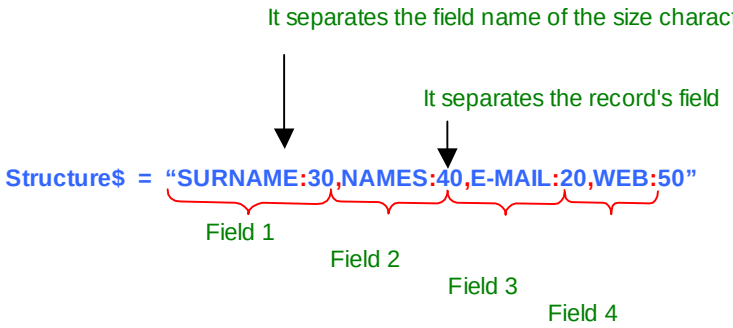
```
RQ_NewConventionalDB$ @RetFunction$,Structure$,IndexField$,_
                      DataBasePath$,DataBaseFile$
```

RetFunction$ return: "ok" if not error occur, If error occur, sub return error code in string format (see error section)

RQ_NewConventionalDB packed sub allow create a new database file in your disk.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
Structure$	Use Structure$ variable to define the structure coincident with the UDT definded in you personal database. In this example we use this UDT: TYPE Fields Apellido AS STRING * 30 'last name field Nombres AS STRING * 40 'first name field Email AS STRING * 20 'e-mail field Web AS STRING * 50 'web field END TYPE GLOBAL DatabaseRecord AS Fields

	You ill indicate through this variable in appropriate format each field name and size, separated by the two points character (":"). Among each field definition, it should interpose a comma character (","). Let us see:  Observe the separater chars. During the database creation, the Structure$ variable will contain the field definition at least with a minimum of 1 char. They are not allowed name of repeated fields inside oneself database. ***Púlsar*** distinguishes between upper and lowercase, in such way the field names like "**telephone**" and "**TELEPHONE**" are coincident, and the manager will reject them. The **names** which you identify each record field can only have up to **30 chars**, but their content can arrive to **999 characters**. If you define a field name with more than 30 chars, the manager will proceed to cut it and he will only read the first 30 chars. If you tries to assign to field's content more than 999 characters, ***Púlsar*** will assign him for defect the allowed maximum value (999). **Example:** **Structure$="APELLIDO:30,NOMBRES:40,E-MAIL:20,WEB:50"**
IndexField$	***Púlsar*** orders the information in database through an index table (or index). This index is ordered according the content of one field. The field that defines the ordination in database will be defined through the IndexField$ variable. To define an index field in database you will enter in this variable the name field. Following our example, if he wants that the database is ordered through the E-MAIL field, you will indicate through the IndexField$ assignment the database index field name: IndexField$ = "E-MAIL" He must make sure that the index field defined in this variable coincides exactly with one of the fields defined in Structure$ variable, or otherwise, you will receive an error message. The index field definition is not obligatory. In case the IndexField$ variable remains empty during the database inicializacion, ***Púlsar*** will assign as defect index field the first field defined in the Structure$ variable, in our case, the defect index field will be made through the field APELLIDO. **Example:** **IndexField$ = "APELLIDO"**

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. Indistinctly the path can finish or not with the " \ " character. If during the command execution the path defined by user doesn't exist, it is created automatically by ***Púlsar***. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. ***Púlsar*** will assign automatically to generated files a defined extension and he will prepare the files to receive information. This manager divides the database information in three files, one contains the information itself, and it is identified easily by the extension **.PLS**, other, identified by the extension **.IDX**, contains the information referred to the database structure and save in his interior the ordination table (or index table) hisself, finally, the memo fields are save in a file identified by **.PLM** estension. The .IDX file is the main structure of ***Púlsar*** database, being constituted in the file that contains the structure, initialization data, tables and order method of its database. But at the same time, all information is recorded in the .PLS and .PLM files, in the .IDX file destruction event, you data will be preserved by the system. **Example:** **DataBaseFile$ = "CLIENTES"**

Example:

```
Structure$="APELLIDO:30,NOMBRES:40,E-MAIL:20,WEB:50"
IndexField$="APELLIDO"

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_NewConventionalDB @RetFunction$,Structure$,IndexField$,PulsarPath$,PulsarFile$

IF LEFT$(RetFunction$,2)<>"ok" THEN

   'If database file is created, the packed sub return "ok".
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
   END 'End program

ELSE

   'database is created
   MESSAGEBOX("OPERACION OK.", "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

12.2.4 SAVE A RECORD AND REINDEX YOU DATABASE

RQ_SaveIndexConvDB

Format:
RQ_SaveIndexConvDB @RetFunction$,DataBasePath$,DataBaseFile$,RecordToSave$

RetFunction$ return: "ok" if not error occur, If error occur, sub return error code in string format (see error section)

RQ_SaveIndexConvDB packed sub allow save a record into you database and reorder the index table automatically. You remember that ***Púlsar*** will save the record still when the index information is duplicated in another record. To verify the existence (or not) of a similar record saved previously, you will make use the SARIndexConDB$ or SARByteConDB$ subs, especialized subs that search record into you database. Other mode is to use SINEConvDB$ sub, that save a record only if not exist previously in you database.
Remember that this sub save the record whit the automatic database ordering (indexation).

It requires to define the following variables:

RetFuntion$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordToSave$	RecordToSave$ variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use UDT structures. **Example:** DatabaseRecord.Apellido = Ponce 'Surname DatabaseRecord.Nombres = Richard 'Names DatabaseRecord.Email = rponce@lanet.com.ar DatabaseRecord.Web = http://www.hepika.com.ar RecordToSave$ = DatabaseRecord 'assign the record fields saved in 'DatabaseRecord UDT into 'RecordToSave$ variable for call 'packed sub

Example:

```
DatabaseRecord.Apellido = Ponce      'Surname
```

```
DatabaseRecord.Nombres = Richard   'Names
DatabaseRecord.Email = rponce@lanet.com.ar
DatabaseRecord.Web = http://www.hepika.com.ar

'assign the record fields saved in DatabaseRecord UDT
'into RecordToSave$ variable for call packed function

RecordToSave$=DatabaseRecord.Apellido
RecordToSave$=RecordToSave$+DatabaseRecord.Nombres
RecordToSave$=RecordToSave$+DatabaseRecord.Email
RecordToSave$=RecordToSave$+DatabaseRecord.Web

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers

RetFunction$=SPACE$(160)

'RQ_SaveIndexConvDB packed sub allow save a record into you database
'and reordering the index table automatically

RQ_SaveIndexConvDB @RetFunction$,PulsarPath$,PulsarFile$,RecordToSave$

'if not error occur, function return "ok", if error occur, return
'"ERROR CODE:XXX.Error_Description" string into CallPulsar$ variable

IF LEFT$(RetFunction$,2)<>"ok" THEN

  Msg$="OPERATION IS NOT COMPLETED IN RECORD:"+Record$+_
       " "+LTRIM$(RTRIM$(RetFunction$))
  MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
  END

END IF
```

12.2.5 SAVING A RECORD IN DATABASE WHITHOUT INDEXATION

RQ_SaveConvDB

Format:
RQ_SaveConvDB$ @RetFunction$,DataBasePath$,DataBaseFile$,RecordToSave$

RetFunction$ return: “ok” if not error occur,If error occur, sub return error code in string format (see error section)

RQ_SaveConvDB packed sub allow save a record into you database **without** reorder the index table. You remember that ***Púlsar*** will save the record still when the index information is duplicated in another record. To verify the existence (or not) of a similar record saved previously, you will make use the SARIndexConDB$ or SARByteConDB$ subs, especialized subs that search record into you database. Other mode is to use SINEConvDB$ sub, that save a record only if not exist previously in you database.

WARNING: Use this sub only for massive entry data that not require search duplicated records in database. Inmediately end the data entry, call the ReindexConvDB$ sub to force the file reindexation

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers

DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordToSave$	**RecordToSave$** variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use **UDT** structures. **Example:** DatabaseRecord.Apellido = Ponce 'Surname DatabaseRecord.Nombres = Richard 'Names DatabaseRecord.Email = rponce@lanet.com.ar DatabaseRecord.Web = http://www.hepika.com.ar RecordToSave$ = DatabaseRecord 'assign the record fields saved in 'DatabaseRecord UDT into 'RecordToSave$ variable for call 'packed sub

Example:

```
DatabaseRecord.Apellido="BBB - ultimo"
DatabaseRecord.Nombres ="bebe"
DatabaseRecord.Email   ="bebe@rapidq.com.ar
DatabaseRecord.Web     ="www.bebe.com.ar"

'assign the UDT to RecordToSave$ variable to call
'the packed sub

RecordToSave$=DatabaseRecord.Apellido
RecordToSave$=RecordToSave$+DatabaseRecord.Nombres
RecordToSave$=RecordToSave$+DatabaseRecord.Email
RecordToSave$=RecordToSave$+DatabaseRecord.Web

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_SaveConvDB @RetFunction$,PulsarPath$,PulsarFile$,RecordToSave$

IF LEFT$(RetFunction$,5)="ERROR" THEN

   'If record is saved, the packed function return "ok".
   'If error occur, function return error code in string
```

```
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
    MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
    END

  ELSE

    'record is saved
    MESSAGEBOX("OPERACION OK.", "Púlsar 3.0 in RapidQ", mbOK)

  END IF
```

12.2.6 FORCED DATABASE REINDEXATION

RQ_ReindexConvDB

Format
RQ_ReindexConvDB$ @RetFunction$, DataBasePath$,DataBaseFile$

RetFunction$ return: “ok” if not error occur, If error occur, sub return error code in string format (see error section)

RQ_ReindexConvDB packed sub forced the database reindexation. This sub must used after SaveConvDB$ (save a record in database an NOT reindex it)

It requires to define the following variables:

RetFuntion$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = “c:\pulsar\ficheros” or** **DataBasePath$ = “c:\pulsar\ficheros\”**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = “CLIENTES”**

Example:

```
  'The RetFunction$ variable return in all pulsar subs the data
  'required by programmers
  RetFunction$=SPACE$(160)

  RQ_ReindexConvDB @RetFunction$,PulsarPath$,PulsarFile$

  IF LEFT$(RetFunction$,5)="ERROR" THEN
    'If databae is reindexed, the packed sub return "ok".
    'If error occur, function return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
```

```
    Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
    MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
    END

  ELSE

    MESSAGEBOX("OPERACION OK.", "Púlsar 3.0 in RapidQ", mbOK)

  END IF
```

12.2.7 GET THE QUANTITY OF RECORD SAVED IN DATABASE

RQ_GetQuantityOfRecords

Format:
RQ_GetQuantityOfRecords$ @RetFunction$,DataBasePath$,DataBaseFile$

RetFunction$ return: if not error occur, return a number in string format, If error occur, sub return error code in string format (see error section)

RQ_GetQuantityOfRecords sub read the quantity of records saved in you database

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**

Example:

```
'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_GetQuantityOfRecords @RetFunction$,PulsarPath$,PulsarFile$

IF LEFT$(RetFunction$,5)="ERROR" THEN

  'If not error occur, function return the quantity or records saved in
  'you database in string format
  'If error occur, return "ERROR CODE:XXX.Error_Description" string into
  'RetFunction$ variable
```

```
    Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
    MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
    END

  ELSE

    Msg$="This database are “+LTRIM$(RTRIM$(RetFunction$))+” records.”
    MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)

  END IF
```

12.2.8 READ A RECORD IN DATABASE

RQ_GetRecordConvDB

Format
RQ_GetRecordConvDB$ @RetFunction$,DataBasePath$,DataBaseFile$,_
RecordNumberToRead$

RetFunction$ return: if not error occur, return a complete record saved, if error occur, sub return error code in string format (see error section)

RQ_GetRecordConvDB allow read a record saved into you database.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = “c:\pulsar\ficheros” or** **DataBasePath$ = “c:\pulsar\ficheros\”**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = “CLIENTES”**
RecordNumberToRead$	RecordNumberToRead$ variable must contain a valid record number to read into database. This record number must not be greater than the last record saved neither smaller than 1 **Example:** RecordNumberToRead$ =”25”

Example:

```
RQ_GetRecordConvDB @RetFunction$,PulsarPath$,PulsarFile$,_
                        LTRIM$(RTRIM$(STR$(Reading%)))

IF LCASE$(LEFT$(RetFunction$,5))="error" THEN

   'If not error occur, function return into the RecordSaved$ variable
   'the record readed.
   'If error occur, return "ERROR CODE:XXX.Error_Description" string into
   'the RetFunction$ variable
   Msg$="Error reading record number:"+ LTRIM$ ( RTRIM$ ( STR$ ( Reading% ) ) )
   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
   END

 ELSE

   'error no occur, assign the RetFunction$ variable data to
   'DatabaseRecord UDT structure
   DatabaseRecord.Apellido=LEFT$(RetFunction$,30)
   DatabaseRecord.Nombres=MID$(RetFunction$,31,40)
   DatabaseRecord.Email=MID$(RetFunction$,71,20)
   DatabaseRecord.Web=MID$(RetFunction$,91,50)

   'show the record
   Msg$="APELLIDO: "+DatabaseRecord.Apellido+CHR$(10)+CHR$(13)
   Msg$=Msg$+"NOMBRES: "+DatabaseRecord.Nombres+CHR$(10)+CHR$(13)
   Msg$=Msg$+"E-MAIL: "+DatabaseRecord.Email+CHR$(10)+CHR$(13)
   Msg$=Msg$+"WEB: "+DatabaseRecord.Web

   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)

 END IF
```

12.2.9 SAVE A RECORD ONLY IF NOT EXIST PREVIOUSLY

RQ_SINEConvDB

Format
RQ_SINEConvDB$ @RetFunction$,DataBasePath$,DataBaseFile$,RecordToSave$

RetFunction$ return: “ok” if not error occur, If error occur, sub return error code in string format (see error section)

SINEConvDB is **S**ave **I**f **N**ot **E**xist previously in **Conv**encional **D**ata**B**ase

RQ_SINEConvDB sub allow save a record and assure it that this record is not saved prevously. Only save the record if the data contain into index file is not find between the saved records of you database

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:**

	DataBasePath$ = “c:\pulsar\ficheros” or **DataBasePath$ = “c:\pulsar\ficheros\”**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = “CLIENTES”**
RecordToSave$	**RecordToSave$** variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use **UDT** structures. RecordToSave$ variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use UDT structures. **Example:** DatabaseRecord.Apellido = Ponce ‘Surname DatabaseRecord.Nombres = Richard ‘Names DatabaseRecord.Email = rponce@lanet.com.ar DatabaseRecord.Web = http://www.hepika.com.ar RecordToSave$ = DatabaseRecord 'assign the record fields saved in ‘DatabaseRecord UDT into ‘RecordToSave$ variable for call ‘packed sub

Example:

```
DatabaseRecord.Apellido= "Rosas"          '¿is this record (Rosas) saved?
DatabaseRecord.Nombres = "Ezequiel Edelmiro"
DatabaseRecord.Email   = "ezequiel_ponce@rapidq.com.ar
DatabaseRecord.Web     = "www.ezequiel.com.ar"

'assign the UDT to RecordToSave$ variable to call
'the packed sub

RecordToSave$=DatabaseRecord.Apellido
RecordToSave$=RecordToSave$+DatabaseRecord.Nombres
RecordToSave$=RecordToSave$+DatabaseRecord.Email
RecordToSave$=RecordToSave$+DatabaseRecord.Web

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_SINEConvDB @RetFunction$,PulsarPath$,PulsarFile$,RecordToSave$

IF LEFT$(RetFunction$,5)="ERROR" THEN

   'If record is saved, the packed function return "ok".
```

```
'If error occur, function return error code in string
'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)

'NOTE:If record is not saved because exist in you database, SINEConvDB$
'function return the 123 code error: "CANNOT SAVE RECORD BECAUSE RECORD
'EXIST IN DATABASE"

ELSE

'record is saved
MESSAGEBOX("OPERACION OK.", "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

12.2.10 SEARCH A RECORD IN YOU DATABASE AND RETURN INDEX NUMBER

RQ_SARIndexConDB

Format
RQ_SARIndexConDB$ @RetFunction$,DataBasePath$,DataBaseFile$,DataToSearch$

RetFunction$ return: if not error occur, return a number in string format, If error occur, sub return error code in string format (see error section)

SARIndexConDB is **S**earch record **A**nd **R**eturn **Index** in **Conv**encional **D**ata**B**ase

RQ_SARIndexConDB sub allow search a record into you database. If record is find return the record number in string format. If record is not find, return error code.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
DataToSearch$	To search information inside database, you should enter through the DataToSearch$ variable the string to look for. The search is perfomed into

	the index field that order you database and the system tries to find the first coincidence with the searched string. Remember that the search only is efficient when you database is ordered (indexed). If previously you has entered data through the save without index sub, you should call the reindexacion sub before search data **Examples:** **DataToSearch $ = "Rosas"** **DataToSearch $ = "Ros"** **DataToSearch $ = "R"** If ***Púlsar*** finds a coincidence, will return the record number (in **string format**) that contains the first coincidence.

Example:

```
DataToSearch$="BBB - ultimo"   'our data to search

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_SARIndexConDB @RetFunction$,PulsarPath$,PulsarFile$,DataToSearch$

IF LEFT$(RetFunction$,5)="ERROR" THEN

   'If record is saved, the packed function return "ok".
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)

ELSE

   'record is find
   Msg$="This record contain data:"+LTRIM$(TRIM$(RetFunction$))
   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

12.2.11 SEARCH A RECORD IN YOU DATABASE AND RETURN RECORD INITIAL BYTE IN DATA FILE

RQ_SARByteConDB

Format:
RQ_SARByteConDB$ @RetFunction$,DataBasePath$,DataBaseFile$,DataToSearch$

RetFunction$ return: if not error occur, return a number in string format, If error occur, sub return error code in string format (see error section)

SARByteConDB is **S**earch record **A**nd **R**eturn initial **Byte** in **Conv**encional **D**ata**B**ase

RQ_SARByteConDB sub allow search a record into you database. If record is find return the initial byte in data file to manipulate it obviating púlsar intervention

¡Warning!: The Púlsar record manipulation contemplate the network permissions access. Through this method you expose the data at corruption or loss.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
DataToSearch$	To search information inside database, you should enter through the DataToSearch$ variable the string to look for. The search is perfomed into the index field that order you database and the system tries to find the first coincidence with the searched string. Remember that the search only is efficient when you database is ordered (indexed). If previously you has entered data through the save without index sub, you should call the reindexacion sub before search data **Examples:** **DataToSearch $ = "Rosas"** **DataToSearch $ = "Ros"** **DataToSearch $ = "R"** If ***Púlsar*** finds a coincidence, will return the the database binary file initial byte (in **string format**) witch contains the record searched.

Example:

```
DataToSearch$="BBB - ultimo"   'our data to search

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_SARByteConDB @RetFunction$,PulsarPath$,PulsarFile$,DataToSearch$

IF LEFT$(RetFunction$$,5)="ERROR" THEN
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
   END
END IF
```

```
'show the initial byte
Msg$="INITIAL BYTE IN DATA FILE: "+LTRIM$(RTRIM$(RetFunction$))
MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
```

12.2.12 RETURN RECORD INITIAL BYTE

RQ_ReturnRecordByte

Format:
RQ_ReturnRecordByte$ @RetFunction$,DataBasePath$,DataBaseFile$,RecordNumber$

RetFunction$ return: if not error occur, return a number in string format, If error occur, sub return error code in string format (see error section)

ReturnRecordByte is **Return Record** initial **Byte** in data file.

RQ_ReturnRecordByte sub allow get the record initial byte in data file to manipulate it obviating púlsar intervention. User must be enter the record number.

¡Warning!: The Púlsar record manipulation contemplate the network permissions access. Through this method you expose the data at corruption or loss.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = "c:\pulsar\ficheros" or** **DataBasePath$ = "c:\pulsar\ficheros\"**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = "CLIENTES"**
RecordNumber$	RecordNumber$ variable must contain a valid record number to read into database. This record number must not be greater than the last record saved neither smaller than 1 **Example:** RecordNumber$ ="15"

Example:

```
RecordNumber$="15"   ' search the 15 record saved in database

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_ ReturnRecordByte @RetFunction$,PulsarPath$,PulsarFile$, RecordNumber$

IF LEFT$(RetFunction$$,5)="ERROR" THEN
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
   END
END IF

'show the initial byte
Msg$="INITIAL BYTE IN DATA FILE: "+LTRIM$(RTRIM$(RetFunction$))
MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
```

NOTE: view ReadRecordDirectly.bas program

12.2.13 MODIFY A RECORD SAVED IN DATABASE

RQ_ModifyRecordConvDB

Format

```
RQ_ModifyRecordConvDB$ @RetFunction$, DataBasePath$,DataBaseFile$,_
                       NewRecordInDB$,RecordToModify$
```

RetFunction$ return: “ok” if not error occur, If error occur, sub return error code in string format (see error section)

RQ_ModifyRecordConvDB packed sub allow modify the data contain in a record saved in you database. If you modify data contain into index field, this sub reordering automatically you database.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = “c:\pulsar\ficheros” or** **DataBasePath$ = “c:\pulsar\ficheros\”**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = “CLIENTES”**

NewRecordInDB$	NewRecordInDB$ variable must contain a valid record to save into database. Púlsar control that the bytes contained in this variable maintain byte to byte correspondece with the record defined during you database creation. The best method to assign it the record to save is to use **UDT** structures. **Example:** DatabaseRecord.Apellido = Ponce ‘Surname DatabaseRecord.Nombres = Richard ‘Names DatabaseRecord.Email = rponce@lanet.com.ar DatabaseRecord.Web = http://www.hepika.com.ar NewRecordInDB$ = DatabaseRecord 'assign the record fields saved in ‘DatabaseRecord UDT into ‘NewRecordInDB$ variable for call ‘packed functio
RecordToModify$	RecordToModify$ variable must contain a valid record number to read into database. This record number must not be greater than the last record saved neither smaller than 1 **Example:** RecordNumber$ =”15”

Example:

```
DatabaseRecord.Apellido="Record Modified!"
DatabaseRecord.Nombres ="RM"
DatabaseRecord.Email   ="rm@modified.com.ar
DatabaseRecord.Web     ="www.modified.com.ar"

RecordToModify$="6"   'record number to modify, this number must not
                      'be greater than the last record saved neither
                      'smaller than 1

'assign the record fields saved in DatabaseRecord UDT
'into RecordToSave$ variable for call packed sub
NewRecordToSave$=DatabaseRecord.Apellido
NewRecordToSave$=NewRecordToSave$+DatabaseRecord.Nombres
NewRecordToSave$=NewRecordToSave$+DatabaseRecord.Email
NewRecordToSave$=NewRecordToSave$+DatabaseRecord.Web

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_ModifyRecordConvDB @RetFunction$,PulsarPath$,PulsarFile$,_
                              NewRecordToSave$,RecordToModify$

IF LEFT$(RetFunction$,2)<>"ok" THEN
   'If record is deleted, the packed sub return "ok".
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)
```

```
ELSE
   MESSAGEBOX("OPERACION OK.", "Púlsar 3.0 in RapidQ", mbOK)
END IF
```

12.2.14 DELETE A RECORD IN DATABASE

RQ_DeleteRecordConvDB

Format
RQ_DeleteRecordConvDB$ @RetFunction$,DataBasePath$,DataBaseFile$,RecordNumber$

RetFunction$ return: “ok” if not error occur, If error occur, sub return error code in string format (see error section)

RQ_DeleteRecordConvDB allow delete a record in you database and reordering automatically the index table.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
DataBasePath$	The DataBasePath$ variable will contain a valid name of path in which the file will be created. **Example:** **DataBasePath$ = “c:\pulsar\ficheros” or** **DataBasePath$ = “c:\pulsar\ficheros\”**
DataBaseFile$	The DataBaseFile$ variable will have a valid name for create a database file. **Example:** **DataBaseFile$ = “CLIENTES”**
RecordNumber$	RecordNumber$ variable must contain a valid record number to read into database. This record number must not be greater than the last record saved neither smaller than 1 **Example:** RecordNumber$ =”25”

Example:

```
RecordToDelete$="5"   'record number to delete, this number must not
                      'be greater than the last record saved neither
                      'smaller than 1 (Use the GetQuantityOfRecords$
                      'packed function to know the last record)
```

```
'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_DeleteRecordConvDB @RetFunction$,PulsarPath$,PulsarFile$,RecordToDelete$

IF LEFT$(RetFunction$,2)<>"ok" THEN
   'If record is deleted, the packed sub return "ok".
   'If error occur, function return error code in string
   'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
   Msg$="OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
   MESSAGEBOX(Msg$, "Púlsar 3.0 in RapidQ", mbOK)

ELSE

   MESSAGEBOX("OPERACION OK.", "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

12.2.15 GET NUMBER SAVED IN COUNTER FILE

RQ_GetCounter

Format
RQ_GetCounter$ @RetFunction$,PulsarPath$,PulsarFile$

RetFunction$ return: if not error occur, return a number in string format, If error occur, sub return error code in string format (see error section)

RQ_GetCounter allow obtain the actual number saved in a counter file and simultaneously increasy it in 1. If the counter file not exist, this command create it automatically and save the 1 value as actual data.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:** **PulsarPath$ = "c:\pulsar\ficheros" or** **PulsarPath$ = "c:\pulsar\ficheros\"**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = "COUNTER"**

Example:

PulsarPath$="c:\pulsar"

```
PulsarFile$=”Counter”

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_GetCounter @ RetFunction$,PulsarPath$,PulsarFile$

IF LEFT$( RetFunction$,5)=”ERROR” THEN

    'If error occur, sub return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    MESSAGEBOX("OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$)),_
                    "Púlsar 3.0 in RapidQ", mbOK)

ELSE

      'If number saved into counter file is returned, RetFunction$ variable contain it
      MESSAGEBOX("COUNTER ACTUAL:" +LTRIM$(RTRIM$(RetFunction$)),_
                      "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

12.2.16 SAVE A NUMBER IN COUNTER FILE

RQ_SetCounter

Format
RQ_SetCounter$ @RetFunction$,PulsarPath$,PulsarFile$,NumberToSet$

RetFunction$ return: “ok” if not error occur, If error occur, sub return error code in string format (see error section)

RQ_SetCounter allow save a value in counter file between 1 and 99.999.999. If counter file not exist, create it automatically and save the selected number.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:** **PulsarPath$ = “c:\pulsar\ficheros” or** **PulsarPath$ = “c:\pulsar\ficheros\”**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = “COUNTER”**

NumberToSet$	NumberToSet$ must contain a number between 1 and 99.999.999. **Example:** **NumberToSet$ = "425"**

Example:

```
PulsarPath$="c:\pulsar"
PulsarFile$="Counter"

NumberToSet$= "1001"

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_SetCounter @ RetFunction$,PulsarPath$,PulsarFile$, NumberToSet$

IF LEFT$( RetFunction$,5)="ERROR" THEN

    'If error occur, sub return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    Msg$= "OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
    MESSAGEBOX( Mgs$, "Púlsar 3.0 in RapidQ", mbOK)

ELSE

     'If number  is saved, RetFunction$ return "ok"
     MESSAGEBOX("COUNTER IS SETTED", "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

12.2.17 SAVE A COMMENT IN LOG FILE

RQ_SaveInLog

Format
RQ_SaveInLog$ @RetFunction$,PulsarPath$,PulsarFile$,CommentToSave$

RetFunction$ return: "ok" if not error occur, If error occur, sub return error code in string format (see error section)

RQ_SaveInLog allow save a comment text into LOG file.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:** **PulsarPath$ = "c:\pulsar\ficheros" or** **PulsarPath$ = "c:\pulsar\ficheros\"**

PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = "LogFile"**
CommentToSave$	CommentToSave$ must contain a comment in string format. **Example:** **CommentToSave$ = "Operation completed."**

Example:

```
PulsarPath$="c:\pulsar"
PulsarFile$="LogFile"

CommentToSave$= "This the comment saved in LOG file"

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_SaveInLog @ RetFunction$, PulsarPath$,PulsarFile$,CommentToSave$

IF LEFT$( RetFunction$,5)="ERROR" THEN

    'If error occur, sub return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    Msg$= "OPERATION IS NOT COMPLETED: "++LTRIM$(RTRIM$(RetFunction$))
    MESSAGEBOX( Mgs$, "Púlsar 3.0 in RapidQ", mbOK)

ELSE

      'If comment  is saved RetFunction$ return "ok"
      MESSAGEBOX("COMMENT SAVED ", "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

12.2.18 READ THE COMMENTS SAVED IN A LOG FILE

RQ_ReadLog

Format
RQ_ReadLog$ @RetFunction$,PulsarPath$,PulsarFile$

RetFunction$ return: if not error occur, return comments in string format
If error occur, sub return error code in string format (see error section)

RQ_ReadLog allow read all comments saved in a LOG file and return in string fromat.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers

PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:** **PulsarPath$ = "c:\pulsar\ficheros" or** **PulsarPath$ = "c:\pulsar\ficheros\"**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = "LogFile"**

Example:

```
PulsarPath$="c:\pulsar"
PulsarFile$="LogFile"

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_ReadLog @ RetFunction$,PulsarPath$,PulsarFile$

IF LEFT$( RetFunction$,5)="ERROR" THEN

    'If error occur, sub return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    Msg$= "OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$))
    MESSAGEBOX( Mgs$, "Púlsar 3.0 in RapidQ", mbOK)

ELSE

      'If comments are returned, RetFunction$ variable contain its
      MESSAGEBOX("COMMENTS SAVED: "+LTRIM$(RTRIM$(RetFunction$)),_
                  "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

12.2.19 DELETE LOG FILE

RQ_DeleteLog

Format
RQ_DeleteLog$ @RetFunction$, PulsarPath$,PulsarFile$

RetFunction$ return: "ok" if not error occur, If error occur, sub return error code in string format (see error section)

DeleteLog$ allow kill the log file specified.

It requires to define the following variables:

RetFunction$	RetFunction$ variable return in all pulsar subs the data required by programmers
PulsarPath$	The PulsarPath$ variable will contain a valid name of path in which the file will be created. **Example:** **PulsarPath$ = "c:\pulsar\ficheros" or** **PulsarPath$ = "c:\pulsar\ficheros\"**
PulsarFile$	The PulsarFile$ variable will have a valid name for create a database file. **Example:** **PulsarFile$ = "LOGFILE"**

Example:

```
PulsarPath$="c:\pulsar"
PulsarFile$="LoGFile"

'The RetFunction$ variable return in all pulsar subs the data
'required by programmers
RetFunction$=SPACE$(160)

RQ_DeleteLog @ RetFunction$,PulsarPath$,PulsarFile$

IF LEFT$( RetFunction$,5)="ERROR" THEN

    'If error occur, sub return error code in string
    'format: "ERROR CODE:xxx...." (xxx is error code -see púlsar manual-)
    MESSAGEBOX("OPERATION IS NOT COMPLETED: "+LTRIM$(RTRIM$(RetFunction$)),_
                "Púlsar 3.0 in RapidQ", mbOK)

ELSE

     'if log file is deleted, RetFunction$ variable contain "ok"
     MESSAGEBOX("LOG FILE IS DELETED: ", "Púlsar 3.0 in RapidQ", mbOK)

END IF
```

APPENDIX 1
ERROR CODES SECTION - PÚLSAR RETURNED CODES

INTERPRETATION OF PLS_RETURN VARIABLE MESSAGES
(Spanish and English Version)

All packed functions (***PowerBasic***) and subs (***RapidQ***) return different messages types (always in string format) when being invoked. If error occur, return an error code in this string format:

"ERROR CODE:xxx.Error Description"

Where **xxx** is a numeric error code and **Error Description** is the error description in corresponding language. In this section, you can see the string returned in packed functions and subs in **blue.** When programming through the "heart direct mode" (**SECTION 11**) Púlsar returned a numeric code in long integer format and a description into the PLS_return variable. This section too describe it in **green**.

COMPLETED OPERATION AND NO ERRORS OCCURS

FUNCTION = 0
PLS_return Maybe assume differents values, according to invoqued operation.
NO ERRORS OCCURS

In messages returned in packed functions and subs when no error occurs see sections 10 (PowerBasic packed functions **) and 14 (** RapidQ packed subs **)**

NETWORK OPERATIONS

Púlsar operates transparently your databases and avoids all possible crash. When you try read file information, if another terminal is operating in this file function can return two values: **118** or **70** (see more descriptions below). In both cases, the operations are not forced by ***Púlsar*** that allows him in that way to generate yours retry routines.

PÚLSAR RETURNED CODES

999.COMMAND NOT SUPPORTED IN THIS VERSION

PULSAR FUNCTION RETURN: 999
PLS_return "COMANDO NO SOPORTADO POR ESTA VERSIÓN"
"COMMAND NOT SUPPORTED IN THIS VERSION"

This message will appear when you tries to invoke a unsupported command in the Púlsar Lite version.

100.PULSAR COMMAND NOT RECOGNIZED

PULSAR FUNCTION RETURN: 100
PLS_return "COMANDO DE PULSAR INCORRECTO"
"PULSAR COMMAND NOT RECOGNIZED"

The user has tried to operate Púlsar whit a unrecognized command. This usually happens when the variable commands are invoked outside the local procedure. Declare commands variables as GLOBAL and assign hem the values inside the main procedure. Another cause is the use of integer numbers and not representatives variables. Use numeric commands throught representative variables.

101.TO CREATE A DATABASE SHOULD BE ASSIGN A NAME'S FILE

PULSAR FUNCTION RETURN: 101
PLS_return "CREAR BASE DE DATOS: DEBE ASIGNAR UN NOMBRE A LA BASE DE DATOS"
"TO CREATE A DATABASE SHOULD BE ASSIGN A NAME'S FILE"

The user has tried to create a new database without entering the name file which Púlsar should create it. Enter a file name through the variable string PLS_param$. When creating a database, is not required the path definition in PLS_path$ variable, but is commendable.

102.DEFAULT PATH NOT DETECTED TO CREATE THE DATABASE

PULSAR FUNCTION RETURN: 102
PLS_return "CREAR BASE DE DATOS: NO SE PUEDE DETECTAR PATH DE DEFECTO"
"DEFAULT PATH NOT DETECTED TO CREATE THE DATABASE"

When creating a new database, you didn't define the Path parameter through the PLS_path$ variable. Púlsar tried to determine the default Path, and for some reason it could not complete the operation. Define a path through the PLS_path$ variable, and if error persists, consider that this error can be returned as consequence of a damage in you device. A surface fail in hard disk impedes the appropriate reading of data, a FAT's problem, or problem in communication between mother board and you disk.

103.SHOULD BE ENTER A VALID COMMAND TO OPERATE PULSAR

PULSAR FUNCTION RETURN: 103
PLS_return "DEBE INGRESAR COMANDO PARA OPERAR PULSAR"
"SHOULD BE ENTER A VALID COMMAND TO OPERATE PULSAR"

The user has tried to invoke to Púlsar without specifying a valid command. This can happen when the command variables are invoked outside the procedure that has defined them. Declare the variables of commands as GLOBAL and assign them the values inside the main procedure.

104.INCORRECT PARAMETERS TO CREATE DATABASE

PULSAR FUNCTION RETURN: 104
PLS_return "PARAMETROS PARA CREACION DE BASE DE DATOS INCORRECTOS"
"INCORRECT PARAMETERS TO CREATE DATABASE"

The user has invoked new database creation command, but the parameters that define their structure (fields and sizes), is not specified. Remember that when you create a new database, must define the PSL_param$ variable contain the fields and length information.

105.NEW DATABASE SHOULD CONTAIN A FIELD AT LEAST

PULSAR FUNCTION RETURN: 105
PLS_return "BASE DE DATOS NUEVA DEBE CONTENER AL MENOS UN CAMPO"
"NEW DATABASE SHOULD CONTAIN A FIELD AT LEAST"

To create a new database, you must define at least a field in record structure. **See 104**

106. INDEX FIELD NAME NOT DEFINED IN PARAMETERS OF DATABASE

PULSAR FUNCTION RETURN: 106
PLS_return "NOMBRE DE CAMPO INDICE DEFINIDO POR USUARIO NO PARAMETRIZADO EN BASE DE DATOS"

"INDEX FIELD NAME NOT DEFINED IN PARAMETERS OF DATABASE"

The user has tried to define as index field a field not defined in the PSL_param$ variable.

107.CANNOT CREATE DATABASE INDEX BECAUSE IT ALREADY EXISTS

PULSAR FUNCTION RETURN: 107
PLS_return "NO SE PUEDE CREAR INDICE DE BASE DE DATOS PORQUE YA EXISTE"
"CANNOT CREATE DATABASE INDEX BECAUSE IT ALREADY EXISTS"

Púlsar doesn't allow to duplicate the database file name inside the same directory or folder.

108.CANNOT SAVE RECORD BECAUSE INDEX FILE IS NOT FOUND

PULSAR FUNCTION RETURN: 108
PLS_return "NO SE PUEDE GRABAR PORQUE NO SE ENCUENTRA ARCHIVO INDICE"
"CANNOT SAVE RECORD BECAUSE INDEX FILE IS NOT FOUND"

This error can be generated by different procedures that you try to access to database. It indicates that Púlsar cannot locate the .IDX file inside the directory defined in PLS_path$. This error can be presented when you tries to access a database that has not been previously created, when the file name is incorrect, when the file path is incorrect or when you has deleted the .IDX file.

109. RECORD IS NOT COINCIDENT WITH DATABASE PARAMETERS

PULSAR FUNCTION RETURN: 109
PLS_return "INFORMACION A GRABAR NO COINCIDE CON PARAMETRIZACIÓN
DE BASE DE DATOS"
"RECORD IS NOT COINCIDENT WITH DATABASE PARAMETERS"

Several procedures that you try to access a conventional database can give this error. It is produced when the user tries save or modify a record entering more or less bytes that the defined for his database. In PSL_param$ variable has more or less bytes that the lenght saved in the record configuration. To avoid this error, is recommended operate with UDT.

110.UNABLE TO INDEX THE DATABASE BECAUSE INDEX FILE IS NOT FOUND

PULSAR FUNCTION RETURN: 110
PLS_return "NO SE PUEDE REINDEXAR PORQUE NO SE ENCUENTRA ARCHIVO INDICE"
"UNABLE TO INDEX THE DATABASE BECAUSE INDEX FILE IS NOT FOUND"

See 110

111."CANNOT CONSULT DATABASE BECAUSE INDEX FILE IS NOT FOUND

PULSAR FUNCTION RETURN: 111
PLS_return "NO SE PUEDE CONSULTAR PORQUE NO SE ENCUENTRA ARCHIVO INDICE"
"CANNOT CONSULT DATABASE BECAUSE INDEX FILE IS NOT FOUND"

See 110

112.IT SHOULD SPECIFY INDEX NUMBER OF RECORD FOR CONSULT

PULSAR FUNCTION RETURN: 112
PLS_return "DEBE ESPECIFICAR UN NUMERO DE INDEXACION PARA EL REGISTRO"
"IT SHOULD SPECIFY INDEX NUMBER OF RECORD FOR CONSULT"

Four Púlsar procedures can generate this error: ***Delete***, ***Modify***, ***Read*** or ***Get The Initial Byte of Record*** in the **.PLS** file. The record number assign toPSL_param$ variable is invalid.

113.RANGE OF CONSULTS IT SHOULD BE BETWEEN 1 TO 999.999.999

PULSAR FUNCTION RETURN: 113
PLS_return "RANGO DE CONSULTA DEBE SER ENTRE REGISTRO Nº 1 AL Nº 99.999.999"
"RANGE OF CONSULTS IT SHOULD BE BETWEEN 1 TO 999.999.999"

Diverse procedures can give this error. The user has tried to access a invalid record number. Púlsar can only support databases with a maximum of 99.999.999 records.

114.NUMBER OF REQUESTED RECORD EXCEEDS THE DATABASE RECORDS SAVING

PULSAR FUNCTION RETURN: 114
PLS_return "NUMERO DE REGISTRO SOLICITADO EXCEDE A LOS GRABADOS EN BASE DE DATOS"
"NUMBER OF REQUESTED RECORD EXCEEDS THE DATABASE RECORDS SAVING"

He has even tried access a record number that doesn't exist in the database. To avoid this error, appeal to the **GetQuantityRecordSaved$** Function or command.

115.CANNOT CONSULT DATABASE BECAUSE INDEX FILE IS NOT FOUND

PULSAR FUNCTION RETURN: 115
PLS_return "NO SE PUEDE BUSCAR PORQUE NO SE ENCUENTRA ARCHIVO INDICE"
"CANNOT CONSULT DATABASE BECAUSE INDEX FILE IS NOT FOUND"

See 110

116.THE LOOKED DATA IS BIGGER SIZE THAT THE INFORMATION OF THE INDEX FIELD

PULSAR FUNCTION RETURN: 116
PLS_return "EL DATO BUSCADO TIENE UNA MAYOR TAMAÑO QUE LA INFORMACION DEL CAMPO INDICE"
"THE LOOKED DATA IS BIGGER SIZE THAT THE INFORMATION OF THE INDEX FIELD"

Púlsar has tried to search a record that chars exceeds the index field size. Make sure that the quantity of chars in PSL_param$ variable not exceeds the index field lenght. Púlsar only search information in the index field, if the data that you look exceeds these size, this error is generated.

117.UNABLE TO OPERATE DATABASE BECAUSE INDEX FILE IS NOT FOUND

PULSAR FUNCTION RETURN: 117
PLS_return "NO SE PUEDE OPERAR LA BASE DE DATOS PORQUE NO SE ENCUENTRA ARCHIVO INDICE"
"UNABLE TO OPERATE DATABASE BECAUSE INDEX FILE IS NOT FOUND"

See 110

118. DATABASE AFFECTED TO EXCLUSIVE PROCESS. ACCESS DENIED.

PULSAR FUNCTION RETURN: 118
PLS_return "BASE DE DATOS AFECTADA A PROCESO EXCLUSIVO. ACCESO

DENEGADO."
"DATABASE AFFECTED TO EXCLUSIVE PROCESS. ACCESS DENIED."

Púlsar cannot access to information because maintenance process is open in database. It is a normal process when several terminals works exist operating in network. If no terminal is operating and database is not being shared at this time, an energy shortcut can have been produced during the reindexation process.

During the reindexation and other processes Púlsar lock all records until concluding the maintenance. If during the process the electric power is interrupted, the index file database have been block.

In this case, use the ***Pulsar Analizer Program*** (freeware) to unblock the index file and recovery you data.

See Pulsar RUN-TIME ERROS, Nº 70

119.NAME FIELDS SHOULD NOT REPEAT

PULSAR FUNCTION RETURN: 119
PLS_return "LOS NOMBRE DE LOS CAMPOS DE LA BASE DE DATOS NO DEBEN REPETIRSE"
"NAME FIELDS SHOULD NOT REPEAT"

The field names should be only and not repeat. Púlsar won't allow him to create a database with repeated fields names.

120.COUNTER VALUE SHOULD BE DIFFERENT TO ZERO

PULSAR FUNCTION RETURN: 120
PLS_return "PARA SETEAR UN CONTADOR DEBE INDICAR UN VALOR DISTINTO DE CERO"
"COUNTER VALUE SHOULD BE DIFFERENT TO ZERO"

Púlsar doesn't allow to assign the zero value in counter file. Valid values are between 1 and 99.999.999

121.COUNTER VALUE CANNOT EXCEED 99.999.999

PULSAR FUNCTION RETURN: 121
PLS_return "EL VALOR DEL CONTADOR NO PUEDE EXCEDER 99.999.999"
"COUNTER VALUE CANNOT EXCEED 99.999.999"

Púlsar doesn't allow to assign a value bigger that 99.999.999. Valid values in counters are between 1 and 99.999.999

122. CANNOT CREATE INDEX OF TEXT'S DATABASE BECAUSE IT ALREADY EXISTS

PULSAR FUNCTION RETURN: 122
PLS_return "NO SE PUEDE CREAR INDICE DE BASE DE DATOS DE TEXTO PORQUE YA EXISTE"
"CANNOT CREATE INDEX OF TEXT'S DATABASE BECAUSE IT ALREADY EXISTS"

He has tried to create a memo database that is already created and defined. Púlsar doesn't allow to duplicate name of files. Rename the database file.

123.CANNOT SAVE BECAUSE INDEX FILE TEXT DATABASE IS NOT FOUND

PULSAR FUNCTION RETURN: 123
PLS_return "NO SE PUEDE GRABAR PORQUE NO SE ENCUENTRA ARCHIVO INDICE DE BASE DE DATOS DE TEXTO"
"CANNOT SAVE BECAUSE INDEX FILE TEXT DATABASE IS NOT FOUND "

See 110

124.THERE IS NOT DATA TO SAVE IN TEXT DATABASE

PULSAR FUNCTION RETURN: 124
PLS_return "NO HAY INFORMACION PARA GRABAR EN BASE DE DATOS DE TEXTO"
"THERE IS NOT DATA TO SAVE IN TEXT DATABASE "

He has tried to save information in a memo database, but the text to record is not in the PLS_param$ variable. Assigns the text record to save to PLS_param$ variable.

125.CANNOT CONSULT TEXT DATABASE BECAUSE INDEX FILE IS NOT FOUND

PULSAR FUNCTION RETURN: 125
PLS_return "NO SE PUEDE CONSULTAR PORQUE NO SE ENCUENTRA ARCHIVO INDICE DE BASE DE DATOS DE TEXTO"
"CANNOT CONSULT TEXT DATABASE BECAUSE INDEX FILE IS NOT FOUND"

See 110

126.SHOULD SPECIFY A RECORD NUMBER TO SEE INFORMATION

PULSAR FUNCTION RETURN: 126
PLS_return "DEBE ESPECIFICAR UN NUMERO DE REGISTRO PARA VER INFORMACIÓN"
"SHOULD SPECIFY A RECORD NUMBER TO SEE INFORMATION"

To search text in saved database, is necessary that indicates the record number. In memo databases, the manager assigns a key number to text and it returns him through the PLS_return$ variable. That same number is the one that is requested now.

127. REQUESTED EXCEEDS THE SAVED RECORDS IN TEXT DATABASE

PULSAR FUNCTION RETURN: 127
PLS_return "EL NUMERO SOLICITADO EXCEDE LOS REGISTROS GRABADOS EN LA BASE DE DATOS"
"REQUESTED EXCEEDS THE SAVED RECORDS IN TEXT DATABASE"

He has requested to search a text in database indicating a nonexistent record number.

128. CANNOT MODIFY TEXT DATABASE RECORD BECAUSE INDEX FILE IS NOT FOUND

PULSAR FUNCTION RETURN: 128
PLS_return "NO SE PUEDE MODIFICAR REGISTRO DE BASE DE DATOS DE TEXTO PORQUE NO SE ENCUENTRA ARCHIVO INDICE "
"CANNOT MODIFY TEXT DATABASE RECORD BECAUSE INDEX FILE IS NOT FOUND"

See 110

129.SHOULD SPECIFY A RECORD NUMBER TO MODIFY INFORMATION

PULSAR FUNCTION RETURN: 129
PLS_return "DEBE ESPECIFICAR UN NUMERO DE REGISTRO PARA MODIFICAR INFORMACIÓN"
"SHOULD SPECIFY A RECORD NUMBER TO MODIFY INFORMATION"

He has requested to modify information in a record saved without assign the record number to **PLS_param$** variable.

130.TO MODIFY REFERENCE'S TEXT SHOULD ENTER DATA IN PSL_return$

PULSAR FUNCTION RETURN: 130
PLS_return "PARA MODIFICAR REFERENCIA DE TEXTO DEBE INGRESARLA A TRAVES DE PSL_retorno$"
"TO MODIFY REFERENCE'S TEXT SHOULD ENTER DATA IN PSL_return$"

He has tried to modify the REFERENCE field in a text database without assign the text to save to **PLS_return$** variable.

131. TO MODIFY DATE'S TEXT RECORD SHOULD ENTER DATA IN PSL_IndexField $

PULSAR FUNCTION RETURN: 131
PLS_return "PARA MODIFICAR FECHA DE CAMPO DE TEXTO DEBE INGRESARLA A TRAVES DE PSL_campoindex$"
"TO MODIFY DATE'S TEXT RECORD SHOULD ENTER DATA IN PSL_IndexField$"

He has tried to modify the DATE field in a text database record without indicating the text to save in that field.

132.TO SAVE IN LOG'S FILES SHOULD ENTER DATA

PULSAR FUNCTION RETURN: 132
PLS_return "PARA GRABAR EN ARCHIVOS LOG DEBE INGRESAR EL TEXTO A GUARDAR"
"TO SAVE IN LOG'S FILES SHOULD ENTER DATA"

He has tried to save information in a LOG file without assigning the text to save to the **PSL_param$** variable.

RUN TIME ERRORS

005.ILLEGAL FUNCTION CALL

PULSAR FUNCTION RETURN: 5
PLS_return "ARGUMENTO INAPROPIADO DE SUBRUTINA O FUNCIÓN"
"ILLEGAL FUNCTION CALL"

This is a catch-all error related to passing an inappropriate argument to some statement or function. A few of the 101 things that can cause it follow:

007.OUT OF MEMORY

PULSAR FUNCTION RETURN: 7
PLS_return "MEMORIA AGOTADA"

"OUT OF MEMORY"

Many different situations can cause this message, including dimensioning too large an array.

009.SUBSCRIPT / POINTER OUT OF RANGE

PULSAR FUNCTION RETURN: 9
PLS_return "PUNTERO FUERA DE RANGO"
"SUBSCRIPT / POINTER OUT OF RANGE"

You attempted to use a subscript smaller than the minimum or larger than the maximum value established when the array was dimensioned. Attempting to use a null pointer will also cause this error.

024. DEVICE TIME-OUT

PULSAR FUNCTION RETURN: 24
PLS_return "TIME-OUT DE DISPOSITIVO"
"DEVICE TIME-OUT"

The specified time-out value for a UDP or TCP communications operation has expired.

051. INTERNAL ERROR

PULSAR FUNCTION RETURN: 51
PLS_return "ERROR INTERNO DE COMPILADOR"
"INTERNAL ERROR"

A malfunction occurred within the PowerBASIC run-time system. Contact the Hepika Technical Support Center.

052. BAD FILE NAME OR NUMBER

PULSAR FUNCTION RETURN: 52
PLS_return "ERROR DE NOMBRE O NUMERO DE ARCHIVO ABIERTO"
"BAD FILE NAME OR NUMBER"

The file number you gave in a file statement doesn't match one given in an OPEN statement, or the file number may be out of the range of valid file numbers.

053. FILE NOT FOUND

PULSAR FUNCTION RETURN: 53
PLS_return "ARCHIVO NO ENCONTRADO"
"FILE NOT FOUND"

The file name specified could not be found on the indicated drive.

054. BAD FILE MODE

PULSAR FUNCTION RETURN: 54
PLS_return "MODO DE ARCHIVO INCORRECTO"
"BAD FILE MODE"

You attempted a PUT or a GET (or PUT$ or GET$) on a sequential file.

055. FILE IS ALREADY OPEN

PULSAR FUNCTION RETURN: 55
PLS_return "ARCHIVO NO PUEDE REABRIRSE"
"FILE IS ALREADY OPEN"

You attempted to open a file that was already open, or you tried to delete an open file.

057. DEVICE I/O ERROR

PULSAR FUNCTION RETURN: 57
PLS_return "ERROR DE DISPOSITIVO"
"DEVICE I/O ERROR"

A hardware problem occurred when trying to carry out some command.

058. FILE ALREADY EXIST

PULSAR FUNCTION RETURN: 58
PLS_return "ARCHIVO YA EXISTE"
"FILE ALREADY EXIST"

The new name argument specified in your NAME statement already exists.

061. DISK FULL

PULSAR FUNCTION RETURN: 61
PLS_return "DISCO LLENO"
"DISK FULL"

Disk full. There isn't enough free space on the indicated or default disk to carry out a file operation. Create some more free disk space and retry your program.

062. INPUT PAST END

PULSAR FUNCTION RETURN: 62
PLS_return "INPUT SUPERA FIN DE DATOS"
"INPUT PAST END"

You tried to read more data from a file than it had to read. Use the EOF (end of file) function to avoid this problem. Trying to read from a sequential file opened for output or append can also cause this error.

063. BAD RECORD NUMBER

PULSAR FUNCTION RETURN: 63
PLS_return "NUMERO DE REGISTRO INCORRECTO"
"BAD RECORD NUMBER"

A negative number or a number larger than 2,147,483,647 was specified as the record argument to a random file PUT or GET statement.

064. BAD FILE NAME

PULSAR FUNCTION RETURN: 64
PLS_return "NOMBRE DE ARCHIVO INCORRECTO"

"BAD FILE NAME"

The file name specified in a KILL, or NAME statement contains invalid characters.

067. TOO MANY FILES

PULSAR FUNCTION RETURN: 67
PLS_return "DEMASIADOS ARCHIVOS"
"TOO MANY FILES"

This error can be caused either by trying to create too many files in a drive's root directory, or by an invalid file name that affects the performance of the Create File system call.

068. DEVICE UNAVAIABLE

PULSAR FUNCTION RETURN: 68
PLS_return "DISPOSITIVO INACTIVO"
"DEVICE UNAVAIABLE"

You tried to OPEN a device file on a machine without that device; for example, OPENing COM1 on a system without a serial adapter or modem.

070. PERMISSION DENIED

PULSAR FUNCTION RETURN: 70
PLS_return "ACCESO DENEGADO POR BLOQUEO DE RED"
"PERMISSION DENIED"

Púlsar cannot access to information because maintenance process is open in database. It is a normal process when several terminals works exist operating in network. If no terminal is operating and database is not being shared at this time, an energy shortcut can have been produced during the reindexation process.

During the reindexation and other processes Púlsar lock all records until concluding the maintenance. If during the process the electric power is interrupted, the index file database have been block.

In this case, use the ***Pulsar Analizer Program*** (freeware) to unblock the index file and recovery you data.

See Pulsar RETURNED ERROR MESSAGES, Nº 118

071. DISK NOT READY

PULSAR FUNCTION RETURN: 71
PLS_return "EL DISCO NO ESTÁ LISTO"
"DISK NOT READY"

The door of a floppy disk drive is open, or there is no disk in the indicated drive.

072. DISK MEDIA ERROR

PULSAR FUNCTION RETURN: 72
PLS_return "SECTOR DE DISCO INCORRECTO"
"DISK MEDIA ERROR"

The controller board of a floppy or hard disk indicates a hard media error in one or more sectors.

074. RENAME ACROSS DISKS

PULSAR FUNCTION RETURN: 74
PLS_return "NO RENOMBRAR ARCHIVOS"
“RENAME ACROSS DISKS”

Rename across disks - You can't rename a file across disk drives.

075. PATH/FILE ERROR

PULSAR FUNCTION RETURN: 75
PLS_return "PATH INCORRECTO"
“PATH/FILE ERROR”

During a command capable of specifying a path name (OPEN, NAME, or MKDIR, for example), you used a path inappropriately; trying to OPEN a subdirectory or to delete a directory in-use, for example.

076. PATH NOT FOUND

PULSAR FUNCTION RETURN: 76
PLS_return "PATH NO ENCONTRADO"
“PATH NOT FOUND”

The path you specified during a CHDIR, MKDIR, OPEN, etc., can't be found.

GENERAL INDEX