VALENTINA

for REALbasic Reference

Paradigma Software Inc.
www.paradigmasoft.com
© 1998 - 2008

Contents

Valentina for REALbasic installation

To install Valentina for REALbasic you should run installer. Installer will ask you to point the plugin folder of your REALbasic.

Installer for MacOS X installs on your computer:
• V4RB_2 folder into REALbasic:plugins folder. This folder contains V4RB plugin itself and folder Examples.
• VComponents folder into /Library/CFMSupport. This folder contains several DLLs, see detailed description of VComponents folder in the Valentina Kernel.pdf.

Installer for Windows installs on your computer:
• V4RB_2 folder into REALbasic:plugins folder. This folder contains V4RB plugin itself and folder Examples.
• VComponents folder into
    C:/Program Files/Paradigma Software/VComponents_win_cw
This folder contains several DLLs, see detailed description of VComponents folder in the Valentina Kernel.pdf.
• append to the system variable PATH the path to VComponents folder, so Windows can find and load DLLs.

To Uninstall:
There is no default uninstaller, so you need manually delete installed folders.

REALbasic start:
After installation you can start REALbasic. If you have **REALbasic PRO** then in the menu "File / Add Data Sources" you can see 2 new items:
        New Valentina ...
        Select Valentina ...
which allow you work with V4RB in RBDB way.

Also V4RB adds to REALbasic set of classes and enumarated types. All Valentina classes start with letter "V". All enumarated types start with "EV". You can type for example 2 letters "va" and get from REALbasic auto-complition of word "Valentina".

If you use **REALbasic Standard** then you need disable RBDB features of V4RB, because RB Standard do not allow use them. To do this you need just create any file with name **DisableRBDB** inside of plugins folder of REALbasic. In this case items in the menu "Add Data Sources" will not present and you can use only classes of V4RB.

Where to start

You should read ValentinaKernel.pdf that contains general information about Valentina database and its features, also ValentinaSQL.pdf that contains description of SQL supported by Valentina.

Then you can read this document, which contains reference of V4RB classes, methods and constants and V4RB_Tutorial.

Also you should study examples of V4RB. It is recommended to use ExampleGuide.pdf to learn examples. This document have additional descriptions of examples and even pictures that simplify understanding.

Deployment of your application

After you have compile your application you need yet bundle it with VComponents folder. You have now 2 choices:

1) Everything is inside of single folder "MyAppFolder".

This way is the most preferable. Because you can install/uninstall application as single folder to computer of your user. For this way you need to do the following steps.

• create folder with name you need, e.g. "MyAppFolder".
• copy inside the executable which REALbasic have built.
• copy inside all files from the VComponents folder (but not folder itself).

Now you can distribute this "MyAppFolder" folder.

Note, Valentina searches first of all the application folder for "vresources" folder. If it finds it here, then Valentina assumes that all other items of Vcomponents folder also here.

2) Valentina folder is located in the system area.

This way is the same as you have it now during development. VComponents folder is located in the central place of OS where any application can find it.

This way can be choosed if you develop several small applications that all use Valentina. Using this way you have VComponents folder only in one place on a user computer.

Converting V4RB 1.x projects to 2.0

Valentina 2.0 introduce not only new API and features, but some old functions was renamed, changed number of parameters, and so on. So you need update your existed V4RB 1.x project. Here you can see list of steps which you should follow to do this work.

General issues

• If your application based on V4RB1 have do any ConvertEncoding/DefineEncoding to/from UTF8 strings then remove all this code because V4RB2 accept UTF8 strings from REALbasic self.

• remove ValentinaUtility methods

• V4RB 2.0 have enums with names EVxxx for constants.

• Find/replace a bunch more kv_ constants with the EV... enumerator equivalents. Most of the ones you need to find will be in the EVFlag domain, but also some in other places like EVOnDelete domain. Compiler will tell you which ones you need to replace since you just threw out ValentinaUtilities.

• IF you have use in the old project NOT kV_constants, but just NUMERIC values of constants then you must find all such places and replace on new Evxxxx constants. This is very very important because 2.0 can have other numeric values for constants.

• V4RB 2.0 supports a new style of errors via VException class. If you convert old 1.x project then you have to use db.LastError style. You can disable new style and return back old style. Use for this Valentina.ThrowExceptions property.

• Global methods are collected in the MODULE “Valentina”, so now:
ValentinaInit() => Valentina.Init()
ValentinaShutDown => Valentina.ShutDown()
ValentinaEscapeString => Valentina.EscapeString()
ValentinaDebugOn => Valentina.DebugLevel property
...

• ValentinaInit() renamed to Valentina.Init()
• ValentinaShutdown() renamed to Valentina.ShutDown()
• ValentinaSetExtensions() renamed to Valentina.SetExtensions()
• ValentinaVersion() renamed to Valentina.Version()

• Valentina.Init() doesn’t return a value anymore. Use the Valentina.CacheSize property to check if the engine was properly initiated.

• if in the 1.x version you have set some encoding for strings BEFORE/AFTER send strings to V4RB then now you no need to do this.
Although for now V4RB 2.0 expect to get UTF8 strings -- default encoding of V4RB

• Dcon and DbgView support depricated. instead we have now V4RB_LOG.txt file that get output from V4RB. So change

```
ValentinaDebugON(2)
```

on

```
Valentina.DebugLevel = EVDebugLevel.kLogParams.
```

It is very recommended to use log file during development. Then in case of any problems you can open V4RB_Log.txt file and monitor a lots of useful information.

Database

• VDatabase.Open() / .Create() no longer return a value. Catch exceptions or use VDatabase.ErrNumber to check errors if you choose Valentina.ThrowExceptions = false.

• If you use Classes, then in constructor of your VDatabase subclass you must add call to parent constructor of VDatabase class. See examples.

Tables

• class VBaseObject renamed to VTable, so you need to reset the super-class for all your Table classes.

• The GotoRecID() no longer exists. Use RecID and RecordExists() :

```
If RecordExists( inRec ) then
   RecID = inRec
end if
```

• The AddRecord() function now returns RecID of new record.

• The DeleteRecord() function no longer returns a value.

Cursors

• Vcursor.SqlString property deprecated.
• VCursor.CurrentPosition renamed to VCursor.Position.
• Check that db.SqlSelect() have correct parameter inReadWrite.

Fields

• Property BaseObject of class VField changed to property Table. Do find/replace.

• Change VField.Nullable to VField.IsNullable.
• Change VField.Compressed to VField.IsCompressed.
• Change VField.Indexed to VField.IsIndexed.
• Change VField.Unique to VField.IsUnique.

• Vfield.SetMethod() no longer exists.
Instead you should on creation of a field provide the method formula into the constructor of field via last parameter. IF you want change existed method, then use
Vfield.MethodText property

• property Language for Vstring/VarChar/Text removed. Also it not exists any more in the constructors of this classes. Instead VDatabse, VTable and VField classes have now new properties:
.Locale
.CollatioAttribute
.StorageEncoding

SQL

• 2.0 SQL do NOT allow you name or tables with key words of SQL. For example field with name “REFERENCES” will not work in SQL or field with name “DEFAULT”. You can find full list of keywords in the ValentinaSQL.pdf

It is recommended always to use some prefix for table/field names, e.g. tblPerson, fldName

• Replace LIKE with REGEX in case you use regex expressions or simply adopt new SQL92 LIKE syntax:

OLD	NEW	Meaning
LIKE ‘string’	LIKE ‘%string%’	Contains
LIKE ‘\Astring\Z’	LIKE ‘string’	is equal to
LIKE ‘\Astring’	LIKE ‘string%’	starts with
LIKE ‘string\Z’	LIKE ‘%string’	ends with

* Keyword NO_CASE deprecated in 2.0. For RegEx you can use instead "(?i)".
Also in 2.0 you cam use db.CollationAttribute to specify case of searches.

* Make sure that all string literals are wrapped by SINGLE QUOTES ' '.

• Look for all ‘ORDER BY’ inside ‘SELECT’ clauses. The ORDER BY column has to be included in the columns SELECTed or equal to *.

```
SELECT * FROM myTable ORDER BY myField    <- Good
SELECT myField FROM myTable ORDER BY myField    <- Good
SELECT myField FROM myTable ORDER BY myOtherField    <- Wrong!
```

Valentina Enumeration Types (Enums)

Valentina for REALbasic 2.0 introduces Enumaration Types - Enums.

REALbasic 5.5 and 6.0 do not support Enums natively. So Valentina for REALbasic does a trick using Modules of REALbasic. This yields a solution that looks exactly like the enum syntax in Java:

```
EnumName.Name
```

Each Valentina's enumeration type starts with the prefix "EV". This allows you to use the power of REALbasic auto-completion. Just type EV and you will see the list of all enumeration types of Valentina for REALbasic.

EVValueAccess

forAdd	= 1
forUpdate	= 2

EVOs

kOsDefault	= 0
kOsMac	= 1
kOsWindows	= 2
kOsUnix	= 3

EVDateFormat

kMDY	= 0
kDMY	= 1
kYMD	= 2

EVDebugLevel

kLogNothing	= 0
kLogErrors	= 1
kLogFunctions	= 2
kLogParams	= 3

EVDbMode

kDscDatBlbInd	= 1
kDsc_DatBlbInd	= 2
kDsc_DatBlb_Ind	= 3
kDsc_Dat_Blb_Ind	= 4
kDscDatBlb_Ind	= 5
kDscDat_Blb_Ind	= 6
kDscDatInd_Blb	= 7
kDsc_DatInd_Blb	= 8

EVFlag

fNone	= 0
fNullable	= 1
fIndexed	= 2
fUnique	= 4
fIndexByWords	= 8
fCompressed	= 16
fMethod	= 32
fIdentity	= 64

EVOnDelete

kNoAction	= 0
kSetNull	= 1
kCascade	= 2
kRestrict	= 3
kDefault	= 4

EVOnUpdate

kNoAction	= 0
kSetNull	= 1
kCascade	= 2
kRestrict	= 3
kDefault	= 4

EVRecursionDirection

kFromParentToChild	= 0
kFromChildToParent	= 1

EVStorageType

kDefault	= 0
kDisk	= 1
kRAM	= 2

EVTableKind

kTblPermanent	= 0
kTblTemporary	= 1

EVCursorLocation

kClientSide	= 1
kServerSide	= 2

EVLockType

kNoLocks	= 1
kReadOnly	= 2
kReadWrite	= 3

EVCursorDirection

kForwardOnly	= 1
kRandom	= 2

EVLinkType

kMany	= 0
kOne	= 1

EVSearch

kPreferIndexed	= 0
kPreferNonIndexed	= 1

EVFieldType

kTypeEmpty	= 0
kTypeEnum	= 1
kTypeBoolean	= 2
kTypeByte	= 3
kTypeShort	= 4
kTypeUShort	= 5
kTypeMedium	= 6
kTypeUMedium	= 7
kTypeLong	= 8
kTypeULong	= 9
kTypeLLong	= 10
kTypeULLong	= 11
kTypeFloat	= 12
kTypeDouble	= 13
kTypeLDouble	= 14
kTypeDecimal	= 15
kTypeDate	= 16
kTypeTime	= 17
kTypeDateTime	= 18
kTypeString	= 19
kTypeVarChar	= 20
kTypeFixedBinary	= 21
kTypeVarBinary	= 22
kTypeBLOB	= 23
kTypeText	= 24
kTypePicture	= 25
kTypeSound	= 26
kTypeMovie	= 27
kTypeRecID	= 28
kTypeOID	= 29
kTypeObjectPtr	= 30
kTypeObjectsPtr	= 31
kTypeTimeStamp	= 32

EVDumpType

kSQL	= 1
kXML	= 2

EVDataKind

kStructureOnly	= 1
kStructureAndRecords	= 2
kRecordsOnly	= 3

EVVerboseLevel

kNone	= 0
kLow	= 1
kNormal	= 2
kHigh	= 3
kVeryHigh	= 4

EVColAttribute

kFrenchCollation	= 0
kAlternateHandling	= 1
kCaseFirst	= 2
kCaseLevel	= 3
kNormalizationMode	= 4
kStrength	= 5
kHiraganaQuaternaryMode	= 6
kNumericCollation	= 7
kAttributeCount	= 8

EVColAttributeValue

kDefault	= -1
kPrimary	= 0
kSecondary	= 1
kTertiary	= 2
kDefaultStrength	= 2
kQuaternary	= 3
kIdentical	= 15
kOFF	= 16
kON	= 17
kShifted	= 20
kNonIgnorable	= 21
kLowerFirst	= 24
kUpperFirst	= 25

EVPictType

kUnknown	= 0
kMacPict	= 1
kWinDIB	= 10
kJPG	= 20
kTIFF	= 21

Valentina Module

Properties

CacheSize	as Integer	(r/o)
DebugLevel	as EVDebugLevel	
FlushEachLog	as boolean	
ThrowExceptions	as Boolean	
Version	as String	(r/o)

DatabaseCount as integer
Database(inIndex as Integer) as VDatabase

Initialisation Methods

```
Init(
        inCacheSize as Integer,
        inMacSerialNumber as String = "",
        inWinSerialNumber as String = "" )

InitClient()

ShutDown()
ShutDownClient()

Convert_1_2(
        inOldDb_Version1 as FolderItem,
        inNewDb_Version2 as FolderItem,
        inLoadRecords as Boolean,
        inDb1Key as String = "",
        inDb1StructureKey as String = "",
        inNewSegmentSize as integer = 0 )
```

Utility Methods

```
SetExtensions( inDesc as string, inDat as String, inBlb as String, inInd as String)
EscapeString( inStr as string, inForRegEx as Boolean = false ) As String

GetDatabaseFormatVersion( inVdbFile as FolderItem ) as Integer
GetCurrentFormatVersion() as Integer

GetSchemaVersion( inVdbFile as FolderItem ) as Integer
GetDatabaseMode( inVdbFile as FolderItem ) as Integer

GetIsStructureEncrypted( inVdbFile as FolderItem ) as Boolean

LocateBonjourService( inType As String, inDomain As String ) As VStringArray
```

Properties Description

CacheSize as Integer (r/o)

The current size of Valentina cache in bytes. You should assign the cache size when calling the Valentina.Init() method. There is no way to change this parameter at runtime.

Example:

```
size = Valentina.CacheSize
```

DebugLevel as EVDebugLevel

This allows you to set the debug level in Valentina for REALbasic.

Any debug level above 0 will create a file which outputs the results. The file will be named “V4RB_Log.txt”. It will be created in the same directory as the project. The only exception is for Mach-O builds in Mac OS X where it will be created one level inside the executable.

The valid values are:

```
kLogNothing   = 0 - no debug messages.
kLogErrors    = 1 - log a message only when an error occurs.
kLogFunctions = 2 - log every function.
kLogParams    = 3 - log every function and its parameters.
```

Example:

```
Valentina.Init( 3 * 1024 * 1024 )
#if DebugBuild
	Valentina.DebugLevel = EVDebugLevel.kLogParams
#endif
```

Note: Do not forget to set the debugging level to zero for your final product release.

FlushEachLog as Boolean

If this property is TRUE then Valentina will flush the disk log file after each message. This slow down work significantly. But is very useful if your application crashes.

TIP: You can wrap the problematic code only.

Example:

```
Valentina.FlushEachLog = true
	// some debugged code
Valentina.FlushEachLog = false
```

ThrowExceptions as Boolean

If this property is TRUE (default value) then Valentina for REALbasic 2.0 or new will throw REALbasic exceptions. Otherwise Valentina 2.0 will not throw exceptions and you need check the property VDatabase.errNumber to see if a Valentina call was successfull.

Example:

```
Valentina.ThrowExceptions = FALSE
```

Version as String (r/o)

Returns the version of the Valentina engine.

Example:

```
ver = Valentina.Version
```

DatabaseCount as integer

Returns: integer

Returns the count of databases that was instantinated in your application. The result counts both opened and closed databases. The result counts both local and remote databases.

Example:

```
res = Valentina.DatabaseCount
```

Database(inIndex as integer) as VDatabase

Returns: integer

Returns a database from the array of databases by an index.

See also:

Valentina.DatabaseCount()

Example:

```
db = Valentina.Database( i )
```

Initialisation Methods

```
Init(
        inCacheSize as Integer,
        inMacSerialNumber as String = "",
        inWinSerialNumber as String = "" )
```

Parameter:	Description:
inCacheSize	The size of the database cache in bytes.
inMacSerialNumber	The serial number for use under Mac OS.
inWinSerialNumber	The serial number for use under Windows.

To improve disk access, Valentina uses a cache mechanism. Using the Valentina.Init() method, you must define the size of the cache. It should be 1MB if the database is tiny, or it can be several megabytes if the database is large.

Tip: By default, it is a good idea to allocate half of available memory to the cache.

Only registered users are allowed to build and deploy Valentina-based applications, except for testing purposes. If you are a registered user, you can specify either the MacOS or the Windows OS serial number, or both. If Valentina receives an empty string, it will work in the time limited, demonstration mode. After ten minutes in demonstration mode, any request to the database will be ignored and Valentina will respond with three beeps.

Note: You must use your own security methods to ensure that you do not expose your serial numbers in your built applications.

Example:

```
        err = Valentina.Init( 5 * 1024 * 1024  )  // demo
```

```
InitClient()
```

Initializes the Valentina kernel for work in the client/server mode.

Example:

```
        Valentina.InitClient()
```

ShutDown()

When you finish working with Valentina, you should shut down it.This method closes all open databases and destroys the cache.

Example:

```
Valentina.Init( 5 * 1024 * 1024, “” , “”  )
        .....// some work here
Valentina.ShutDown()
```

ShutDownClient()

Executes clean up and finalization of work in the client/server mode.

Пример:

```
Valentina.ShutDownClient()
```

Convert_1_2(

```
        inOldDb_Version1 as FolderItem,
        inNewDb_Version2 as FolderItem,
        inLoadRecords as Boolean,
        inDb1Key as String = "",
        inDb1StructureKey as String = "",
        inNewSegmentSize as integer = 0 )
```

Parameter:	**Description:**
inOldDb_Version1	location of database in 1.x format.
inNewDb_Version2	Location for new database of 2.0 format.
inLoadRecords	If TRUE then records are copied to new database.
inDb1Key	Encryption Key of DB1.
inDb1StructureKey	Structure Encryption Key of DB1.
inNewSegmentSize	Allows to change db.SegmentSize.

Convert database of 1.x format into database of 2.0 format. The old Database must be closed before use of this method.

Note: This function do not change the old Database.

Example:

```
db.Convert_1_2( oldDB, newDB, true )
```

Utility Methods

```
SetExtensions(
        inDesc as String,
        inDat as String,
        inBlb as String,
        inInd as String)
```

Parameter:	**Description:**
inDesc	Extension for description file (.vdb)
inDat	Extension for data file (.dat)
inBlb	Extension for BLOB file (.blb)
inInd	Extension for indexes file (.ind)

You can call this function before opening or creating a database to inform the Valentina kernel which extensions it must use for database files. If you do not explicitly call this method, then the standard four extensions are used by default. If you do use this method, you must explicitly include all extensions that you want supported in your database application.

Note: The four standard file types of a Valentina database are explained in full in the ValentinaKernel.pdf.

The first example shows explicitly setting the standard extensions in a four file database.

The second example shows a database in which two files are created:
* the description database file using its standard extension;
* the index file with a custom file type of .tre instead of its standard extension, .ind.

Example(s):

```
Valentina.SetExtensions( "vdb", "dat", "blb", "ind" )

Valentina.SetExtensions( "vdb", "", "", "tre" )
```

```
EscapeString(
        inStr as String,
        inForRegEx as Boolean = false ) as String
```

Parameter:	**Description:**
inStr	The string to be escaped.
inForRegEx	TRUE if you are preparing string for a REGEX search.

This utility function is used if you build a string out of an SQL query which may use the single quote escape character. This allows you to escape a string (usually from user input) before you concatenate that string into a SQL query.

If you set inForRegEx to TRUE, then the string is treated as a regular expression and before If the inForRegEx parameter is FALSE then only a single quote character is treated by this function.

Example(s):

```
        res = Valentina.EscapeString( "Valentina's (day)", 0 )
        // res is "Valentina\'s (day)"

        res = Valentina.EscapeString( "Valentina's day", 1 )
        // res is "Valentina\'s \(day\)"

       query = "SELECT * FROM T WHERE f1 LIKE '" + s1 + "' OR f2 REGEX '" + s2 "'"
```

```
GetDatabaseVersion( inVdbFile as FolderItem ) as Integer
```

Parameter:	**Description:**
inVdbFile	Path to the database file.

Returns the version of the database file format. It can work even with a closed database.

Example:

```
        dim fi as FolderItem
        dim vers as integer

        fi = GetFolderItem( "MyDatbase.vdb" )
        vers = Valentina.GetDatabaseVersion( fi )
```

```
GetCurrentFormatVersion() as Integer
```

Returns the current format version of database file.

Example:

```
        vers = Valentina.GetCurrentFormatVersion
```

GetSchemaVersion(inVdbFile as FolderItem) as Integer

Parameter:	Description:
inVdbFile	Path to the database file.

Returns the version of database schema. It can work even with a closed database.

Example:

```
dim fi as FolderItem
dim SchemaVersion as integer

fi = GetFolderItem( "MyDatbase.vdb" )
SchemaVersion = Valentina.GetSchemaVersion( fi )
```

GetDatabaseMode(inVdbFile as FolderItem) as Integer

Parameter:	Description:
inVdbFile	Path to the database file.

Returns the database mode. It can work even with a closed database.

Example:

```
dim fi as FolderItem
dim dbMode as integer

fi = GetFolderItem( "MyDatbase.vdb" )
dbMode = Valentina.GetDatabaseMode( fi )
```

GetIsStructureEncrypted(inVdbFile as FolderItem) as Boolean

Parameter:	Description:
inVdbFile	Path to the database file.

Returns TRUE if database structure is encrypted. It can work even with a closed database.

Example:

```
dim fi as FolderItem
dim isEncrypted as integer

fi = GetFolderItem( "MyDatbase.vdb" )
isEncrypted = Valentina.GetStructureEncrypted( fi )
```

LocateBonjourService(inType As String, inDomain As String) as VStringArray

Parameter:	**Description:**
inType	A service name.
inDomain	A domain name. Pass here empty string currently.

This method allow you discover a specified service using Bonjour on the network. When you call this method you need specify the name of Bonjour service you want to find. For Valentina Server this is "_valentina._tcp". As result you get an array of strings that contain Bonjour service description. If not found any such service then a nil is returned. You can show strings of this array in GUI, so user can choose what service he want to connect. To establish connection using bonjour string, simply pass it to VConnection() constructor in the place of inHost parameter.

Example:

```
Dim resArray As VStringArray
Dim count, i As Integer
Dim item As String

resArray = Valentina.LocateBonjourService( "_valentina._tcp", "" )

if resArray <> nil then
        count = resArray.count

        for i = 1 to count
                item = resArray.GetItemAt(i)
                ' Do something with item
         next
end if
```

The Class Hierarchy

Because of performance considerations, Valentina for REALbasic is implemented as a set of classes, bypassing REALbasic's internal database plugin API.

However, you may also use REALbasic's database plugin API. REALbasic's database API allows developers to leverage the internal methods and functions of the REALbasic evironment, just like REAL Software's own database plugins. Using the REALbasic database API requires the Pro version of REALbasic. In order to disable using this, you must put a file into the REALbasic plugin folder with the name "DisableRBDB". This file can be completely blank/
The following are the Valentina for REALbasic classes. To learn more about how classes work in REALbasic, consult the REALbasic Developer's Guide.
Important: You should not mix using the Valentina API and REALbasic database API method to access a Valentina dabase in your application.

- class VDataBase
- class VTable
- class VLink
  - class VLink2
    - class VBinaryLink
- class **VField**
  - class VBoolean
  - class VByte
  - class VShort
  - class VUShort
  - class VMedium
  - class VUMedium
  - class VLong
  - class VULong
  - class VFloat
  - class VDouble
  - class VDate
  - class VTime
  - class VDateTime
  - class VString
  - class VVarChar
  - class VFixedBinary
  - class VVarBinary
  - class VBLOB
    - class VText
    - class VPicture
  - class VObjectPtr
- class VCursor
- class VSet
  - class VArraySet
  - class BitSet
- class VSetIterator

Note: The class **VField** is an abstract class. You cannot create it by using the operator NEW. Only its subclasses can be created and used explicitly.

Class VDataBase

Properties

CenturyBound	as Integer	// default 20.	
Creator	as String	// Mac creator signature	

CollationAttribute(inColAttribute as EVColAttribute) as EVColAttributeValue
CollationAttribute(inColAttribute as EVColAttribute, inColAttributeValue as EVColAttributeValue)

DateFormat	as EVDateFormat	// specifies the format of date.	
DateSep	as String	// separator for date, e.g. '/'	
ErrNumber	as Integer (r/o)	// Number of the last error, 0 if OK.	[DEPRECATED]
ErrString	as String(r/o)	// String description of error.	[DEPRECATED]
IndexCount	as Integer (r/o)		
IsEncrypted	as Boolean (r/o)	// TRUE if the database is encrypted.	
IsOpen	as Boolean (r/o)		
IsReadOnly	as Boolean (r/o)		
IsRemote	as Boolean (r/o)		
LastInsertedRecID	as Integer (r/o)		
LinkCount	as Integer (r/o)		
LocaleName	as String		
Mode	as EVDbMode		
Name	as String (r/o)		
Path	as String (r/o)		
SchemaVersion	as Integer	// Version of db Schema	
SegmentSize	as Integer		
StorageEncoding	as String		
TableCount	as Integer (r/o)		
TimeSep	as String	// separator for time, e.g. ':'	

// for CLIENT only:

ResponseTimeout	as Integer	// default 60 seconds.

ConnectionVariable(inConnVariable as EVConnectionVariable) as EVConnectionVariableValue
ConnectionVariable(inConnVariable as EVConnectionVariable, inValue as EVConnectionVariableValue)

Constructor

VDatabase(inStorageType as EVStorageType = kDefault)

```
VDatabase(
        inConnection As VConnection )
```

Disk Methods

```
Create(
        inLocation as FolderItem,
        inMode as EVDbMode = kDsc_Dat_Blb_Ind,
        inSegmentSize as integer = 32768,
        inNativeOS as EVOs = kOsDefault )

Open( inLocation as FolderItem )
Close()
ThrowOut()
Flush()
SetMacTypes(
        inDescType as String,
        inDatType as String,
        inBlbType as String,
        inIndType as String )

Clone( inTargetDb as FolderItem, inLoadRecords as Boolean = true, inDoLog as Boolean = false )
Clone( inTargetDb as VDatabase, inLoadRecords as Boolean = true, inDoLog as Boolean = false )
```

Structura methods

```
CreateTable(
        inName as String,
        inTableKind as EVTableKind = kTblPermanent,
        inStorageType as EVStorageType = kDefault ) as VTable

DropTable(inTable as VTable)

CreateForeignKeyLink(
        inName as String,
        inKeyField as VField,
        inPtrField as VField,
        inOnDelete as EVOnDelete = kSetNull,
        inOnUpdate as EVOnUpdate = kCascade,
        inTemporary as Boolean = FALSE ) as VLink

CreateBinaryLink(
        inName as String,
        inLeftTable as VTable,
        inRightTable as VTable,
        inLeftPower as EVLinkType = kOne,
        inRightPower as EVLinkType = Many,
        inOnDelete as EVOnDelete = kSetNull,
        inStorageType as EVStorageType = kDefault,
        inTemporary as Boolean = false) as VBinaryLink

DropLink( inLink as VLink )
```

Table methods

Table(inIndex as Integer) as VTable
Table(inName as String) as VTable

Link methods

Link(inIndex as Integer) as VLink
Link(inName as String) as VLink

IndexStyle methods

CreateIndexStyle(inName as String) as VIndexStyle
DropIndexStyle(inStyle as VIndexStyle)
IndexStyle(inName as String) as VIndexStyle

SQL methods

```
SqlExecute(
        inQuery as String,
        inBinds() as String = nil ) as Integer

SqlSelect(
        inQuery as String,
        inCursorLocation as EVCursorLocation = kClientSide,
        inLockType as EVLockType = kReadOnly,
        inCursorDirection as EVCursorDirection = kForwardOnly
        inBinds() as String = nil ) as VCursor
```

Encryption method

```
ChangeEncryptionKey(
        inOldKey as String
        inNewKey as String
        inForData as Integer = EVDataKind.kRecordsOnly )

Encrypt(
        inKey as String,
        inForData as Integer = EVDataKind.kRecordsOnly )

Decrypt(
        inKey as String,
        inForData as Integer = EVDataKind.kRecordsOnly )

RequiresEncryptionKey()

UseEncryptionKey(
        inKey as String,
        inForData as Integer = EVDataKind.kRecordsOnly )
```

Dump methods

```
Dump(
        inDumpFile as FolderItem,
        inDumpType as Integer,
        inDumpData as EVDataKind = kStructureAndRecords,
        inFormatDump as Boolean = false,
        inEncoding as String = UTF-16 )

LoadDump(
        inDumpFile as FolderItem,
        inNewDb as FolderItem,
        inDumpType as Integer,
        inEncoding as String = UTF-16 )
```

Utility methods

```
Diagnose(
        inVerboseLevel as EVVerboseLevel = kNone,
        inFile as FolderItem = nil ) as Boolean
```

Class Description

This class manages a database. Valentina can have multiple open databases.
Each database has an unique (case insensitive) name. Each database must have at least one table.

Properties Description

CenturyBound as Integer

This property specifies how Valentina automatically corrects dates that contains a 2 digit year value, e.g.

```
"20/04/89" -> "20/04/1989"
"20/04/04" -> "20/04/2004"
```

The default is 20.

Example:

```
cntb = db.CenturyBound
```

CollationAttribute(

```
CollationAttribute(
        inColAttribute as EVColAttribute ) as EVColAttributeValue

CollationAttribute(
        inColAttribute as EVColAttribute,
        inColAttributeValue as EVColAttributeValue )
```

Set/Get the value of the specified collation attribute for this database.

Example:

```
dim v as integer

v = database.CollationAttribute( EVColAttribute.kStrength )

database.CollationAttribute( EVColAttribute.kStrength ) =
        EVColAttributeValue.kPrimary
```

```
ConnectionVariable(
        inConnVariableName as EVConnectionVariable ) as Integer

ConnectionVariable(
        inConnVariableName as EVConnectionVariable,
        inConnVariableNameValue as EVConnectionVariableValue )
```

Get/Set the value of the connection variable by its name.

Example:

```
        dim i as integer

        i = database.ConnectionVariable(EVConnectionVariable.kFilesTransferMode)

        database.ConnectionVariable(EVConnectionVariable.kFilesTransferMode) =
                EVConnectionVariableValue.kNetwork
```

```
Creator as String
```

With MacOS applications, you can specify the creator’s signature for database files. This allows you to design an icon suite for your application.

Example:

```
        creator = db.Creator
```

DateFormat as EVDateFormat

Specify the date format for strings that contains date values. You can set format to the one of the following values: kYMD(Year, Month, Day), kDMY(Day, Month, Year), kMDY(Month, Day, Year).

Example:

```
dtf = db.DateFormat
```

DateSep as String

The character that is used as a separator in the date string. The default is "/".

Example:

```
dts = db.DateSep
```

ErrNumber as Integer [DEPRECATED]

You cam examine this property to see if the last operation was successful. Since this is a property of the database, each open database has its own “last error” number.

There are 2 kind of errors: OS-relative errors and Valentina-specific errors. OS-based errors are negative numbers. You can find their description in your OS documentation. Valentina specific errors are positive numbers.

Example:

```
errnumber = db.ErrNumber
```

ErrString as String [DEPRECATED]

Returns the string that describes the last error.

Example:

```
errstr = db.ErrString
```

IndexCount as Integer (r\o)

Returns the count of indexes in all tables of this database.

Example:

```
count = db.IndexCount
```

IsEncrypted as Boolean (r\o)

Returns TRUE if this database is encrypted.

Example:

```
encrypted = db.isEncrypted
```

IsReadOnly as Boolean (r/o)

Returns TRUE if this database is read only, i.e. it is located on the locked volume or files of databases are marked as read only.

Example:

```
res = db.IsReadOnly
```

IsRemote as Boolean (r/o)

Returns TRUE if this database is remote.

Example:

```
res = db.IsRemote
```

IsOpen as Boolean (r\o)

Returns TRUE if this database is open now.

Example:

```
res = db.IsOpen
```

LastInsertedRecID as integer (r/o)

Returns: integer

Returns the last inserted RecID in the database. Returns 0 as invalid RecID, for example if there was no any INSERTs.

This function is useful mainly if you execute

```
db.SqlExecute( "INSERT INTO T ..." )
```

because it allows you to get RecID of just inserted record. You should call this function right after SqlExecute() call. Actually any other INSERT into this database will change the result of this function.

Function VTable.AddRecord() also affects the result of this function.

Note, that if you use this function with Valentina Server then its result does not depend on work of other users.

Example:

```
recid = db.LastInsertedRecID
```

LinkCount as Integer (r\o)

Returns the count of links in the database. This property is indirectly changed when you create/drop a link, or when you establish a FOREIGN KEY constraint, or when you create an ObjectPtr field.

Example:

```
count = db.LinkCount
```

LocaleName as String

Defines the locale name for this database. Tables and fields of this database will inherit this parameter.

Example:

```
localeName = db.LocaleName
db.LocaleName = "en_US"
```

Mode as EVDbMode (r\o)

Returns the mode of this database. Using this you can define how many files hold the information in the database.

Example:

```
mode = db.Mode
```

Name as String (r\o)

The name of database.

Example:

```
name = db.Name
```

Path as String (r\o)

The full path to this database.

Example:

```
path = db.Path
```

SchemaVersion as Integer

The of version number of a database schema. Initial value is 1. It can be used if you want to change a database structure in the new version of your application.

Example:

```
ver = db.SchemaVersion
```

SegmentSize as Integer (r\o)

Returns the segment size (in bytes) of a database.

Example:

```
seg = db.SegmentSize
```

StorageEncoding as String

Defines how strings will be stored on disk. By default it is UTF-16. You can change it to any other encoding.

IMPORTANT: you can assign an encoding to a VDatabase object only before calling the Vdatabase.Create() function. You cannot change the encoding of existing db files using this property.

Example:

```
encoding = db.StorageEncoding
```

TableCount as Integer (r\o)

Returns the count of custom tables in the database (i.e. it does not count the system tables). This property is indirectly changed when you create/drop a Table.

Example:

```
count = db.TableCount
```

TimeSep as String

The character that is used as a separator for time values. The default is ":".

Example:

```
tms = db.TimeSep
```

ResponseTimeOut as Integer

This property affects only Valentina Client. It is specifies the time (in seconds) which the client will wait for a response from the server on a query. If during this time the server does not respond then the client disconnects.

By default this property is 60 seconds. You may wish set this value larger if you have some complex query and you know that the server will take a long time to resolve it.

Example:

```
db.ResponseTimeOut = 100
```

VDatabase: Local vs Remote Creation

The VDatabase class constructor has two forms. The first is for a LOCAL database and the second for a CLIENT database.

```
VDatabase( inStorageType as EVStorageType = kDefault )
```

Parameter	**Description**
inStorageType	Storage type for this database

You should use the first form, if you create a database object that will work with a local database.

The parameter inStorageType specifies if the database will be created on the DISK or in RAM. By default the database is disk-based.

Example:

```
db = new VDatabase

db = new VDatabase( EVStorageType.kRAM )
```

```
VDatabase(
        inConnection As VConnection )
```

Parameter	**Description**
inConnection	VConnection object.

You need the second form to create a VDatabase object to access a remote database. It does not establish a connection, but just stores parameters that will be used later. The connection is established on a call of either Open() or Create().

Example:

```
remote_db = new VDatabase( inConnection )
```

Disk Methods

```
Create(
        inLocation as FolderItem,
        inMode as EVDbMode =  kDsc_Dat_Blb_Ind,
        inSegmentSize as Integer = 32768,
        inNativeOS as EVOs = kOsDefault )
```

Parameter:	**Description:**
inLocation	The path to the database on the disk.
inMode	How many files for databases will be used, range 1-8; default 4.
inSegmentSize	The size of one cluster in the database file; default 32KB.
inNativeOS	The byte order for the database.

Creates a new, empty database on disk.

Note: After creation, the database is already open.

As the Mode parameter you can specify one of the following:

kDscDatBlbInd	// (description,data,BLOB,indexes)
kDsc_DatBlbInd	// description + (data,BLOB,indexes)
kDsc_DatBlb_Ind	// description + (data,BLOB) + indexes
kDsc_Dat_Blb_Ind	// description + data + BLOB + indexes
kDscDatBlb_Ind	// (description,data,BLOB) + indexes
kDscDat_Blb_Ind	// (description,data) + BLOB + indexes
kDscDatInd_Blb	// (description,data,indexes) + BLOB
kDsc_DatInd_Blb	// description + (data,indexes) + BLOB

Example:

```
db.Create( file, kDscDatBlb_Ind, 32 * 1024 )
```

Example:

```
// For a remote database, you need to specify only
// the name of the database that is registered with Valentina Server.

f = GetFolderItem("My Database1")
remote_db.Create( file, kDscDatBlb_Ind, 32 * 1024 )
```

Open(inLocation as FolderItem)

Parameter:	**Description:**
inLocation	The path to the database on the disk.

Opens an existing database at the specified location.

Example:

```
db.Open( file )
```

Example:

```
// For a remote database, you need specify just
// the name of the database that is registered with Valentina Server.

f = GetFolderItem("My Database1")
remote_db.Open( file )
```

Close()

Closes the database.

Example:

```
db.Open()
        ....
db.Close()
```

ThrowOut()

Deletes all database files from disk. This database must be closed.

Example:

```
db.Close()
db.ThrowOut()
```

Flush()

Flushes all unsaved information of this database from cache to disk.

Example:

```
db.Flush()
```

IsRemote (r/o)

Each database (never mind - local or remote) has been registered to the single array of databases. So we should be able to check it.

Example:

```
db.IsRemote
```

SetMacTypes(

```
SetMacTypes(
    inDescType as String,
    inDatType as String,
    inBlbType as String,
    inIndType as String )
```

Parameter:	**Description:**
inDescType	Mac Type of the ".vdb" file of the database.
inDatType	Mac Type of the ".dat" file of the database.
inBlbType	Mac Type of the ".blb" file of the database.
inIndType	Mac Type of the ".ind" file of the database.

This function allows you to assign own file types for database files. This is required on MacOS to correctly show custom icons.

Example:

```
db.SetMacTypes( "Mdsc",  "Mdat", "Mblb", "Mind" )
```

```
Clone(
        inTargetDb as FolderItem,
        inLoadRecords as Boolean = true,
        inDoLog as Boolean = false )
```

Parameter:	**Description:**
inTargetDb	The Path for a new database.
inLoadRecords	If TRUE then records are copied into the cloned database.
inDoLog	If TRUE then this method produce log file.

This function creates a new database which is a logical clone of this database. We say logical because physically it is not identical. For example the space used with deleted records will not be copied. This means that the cloned database can be smaller of original.

On default records also are copied into the cloned database. You can specify inLoadRecords to be FALSE to clone only the Database Structure. See details in the ValentinaKernel.pdf.

If Parameter inDoLog is TRUE then it produces a log file in the folder of database. This log file will contains information only about corrupted fields/records if any. This allows to user explicitly see where he can lost changed during cloning of database.

Example:

```
        newDbLocation = GetOpenFolderItem()
        db.Clone( newDbLocation )
```

```
Clone(
        inTargetDb as VDatabase,
        inLoadRecords as Boolean = true,
        inDoLog as Boolean = false )
```

The same as above except that first parameter is not disc location, but already existent VDatabase object.

This form allows you to create a new empty VDatabase and specify some parameters of VDatabase, e.g. Mode, SegmentSize. Later the Clone() method will copy rest of the structure and records into this database.

Example:

```
        newDbLocation = GetOpenFolderItem()

        dbCloned = new VDatabase
        dbCloned.Create( newDbLocation, kDscDatBlb_Ind, 8 * 1024 )

        db.Clone( dbCloned )
```

Database Structure Methods

```
CreateTable(
        inName as String,
        inTableKind as EVTableKind = kTblPermanent,
        inStorageType as EVStorageType = kDefault ) as VTable
```

Parameter:	**Description:**
inName	The Name of a new Table.
inTableKind	The kind of Table
inStorageType	Storage type for this database

Creates a new empty Table in the database.

The parameter inTableKind allows you to choose between permanent and temporary tables.

The parameter inStorageType allows for the creation of Tables in RAM.

Note: This only applies to a DISK-based database. It is obvious that for a RAM-based database that you cannot create a disk-based table.

Note: You need to add columns to a new table using the VTable.CreateField() method.

Example:

```
dim tbl as VTable

tbl = db.CreateTable( "Person" )
```

```
DropTable( inTable as VTable )
```

Parameter:	**Description:**
inTable	The reference of Table to delete.

Removes the specified Table from the database. This operation is undoable.

Example:

```
db.DropTable( tbl )
```

```
CreateBinaryLink(
        inName as String,
        inLeftTable as VTable,
        inRightTable as VTable,
        inLeftPower as EVLinkType = kOne,
        inRightPower as EVLinkType = kMany,
        inOnDelete as EVOnDelete = kSetNull,
        inStorageType as EVStorageType = kDefault
        inTemporary as Boolean = false ) as VBinaryLink
```

Parameter:	**Description:**
inName	The name of the link.
inLeftTable	Pointer to the Left Table.
inRightTable	Pointer to the Right Table.
inLeftPower	Link type for the Left Table.
inRightPower	Link type for the Right Table.
inOnDelete	The behavior on deletion of record-owner.
inStorageType	Storage type of the link.
inTemporary	TRUE if the link is temporary.

Creates a new Binary Link between 2 tables of this database.

To specify a link you need to define the following:

• A name for the link, unique in the scope of the database.

• Pointers to 2 tables. One table is named Left, the other is named Right.

• The type of link, i.e. if it is 1 : 1 or 1 : M or M : M.

• The behavior of the link on deletion of a record in the Table-Owner.
- In the case of a 1 : M link, the ONE table is the owner table
- In the other cases (1:1 and M:M) the developer can assign which table is to be the owner.

• The storage type for the link. Can be Disk-based or RAM-based.

A BinaryLink creates files on disk to keep information about linked records. This is why we need to specify StorageType.

You can specify the same table in the parameters inLeftTable and inRightTable. In this case you get a recursive link (or self-pointer).

Example:

```
        linkPersonPhone = db.CreateBinaryLink(
                "PersonPhone", tblPerson, tblPhone,
                EVLinkType.kMany, EVLinkType.kMany )
```

```
CreateForeignKeyLink(
        inName as String,
        inKeyField as VField,
        inPtrField as VField,
        inOnDelete as EVOnDelete = kSetNull,
        inOnUpdate as EVOnUpdate = kCascade,
        inTemporary as Boolean = false ) as VLink
```

Parameter:	**Description:**
inName	The name of link.
inKeyField	The PRIMARY KEY field of ONE Table.
inPtrField	The PTR field in the MANY Table.
inOnDelete	The behavior on deletion of record-owner.
inOnUpdate	The behavior on update of record-owner.
inTemporary	TRUE if link is temprary.

Creates a Link between 2 tables of this database using the FOREIGN KEY abstraction of the relational model. This link does not create on disk any new structures. It just establishes logical links between records using their values in the KEY and PTR fields. This function is 100% the analog of the FOREIGN KEY constraint in SQL of a RDBMS. Valentina allows a way to establish a relational link without the use of SQL.

To specify a foreign key link you need to define the following:
• A name for the link, unique in the scope of the database.
• The KEY field of the Parent table (ONE table).
• The PTR field of the Child table (MANY table).
• The behavior of the link on deletion of a record in the Parent Table.
• The behavior of the link on update of a KEY field value in the Parent Table.

Example:

```
        linkPersonPhone = db.CreateForeignKeyLink(
                "PersonPhone", tblPerson.fldID, tblPhone.PersonPtr )
```

```
DropLink( inLink as VLink )
```

Parameter:	**Description:**
inLink	The reference of Link to delete.

Removes the specified Link from the database. This operation is undoable.

Example:

```
        db.DropLink( lnk )
```

Table Methods

Table(inIndex as Integer) as VTable

Parameter:	**Description:**
inIndex	The index of a Table in a database, start from 1.

Returns a Table by an numeric index.

Example:

```
Table = db.Table( i )
```

Table(inName as String) as VTable

Parameter:	**Description:**
inName	The Name of a Table.

Returns a Table by name.

Note: The parameter inName is case insensitive.

Example:

```
Table = db.Table( "Person" )
```

Link Methods

Link(inIndex as Integer) as VLink

Parameter:	**Description:**
inIndex	The index of a Link in a database, start from 1.

Returns a Link based on numeric index.

Example:

```
Link = db.Link( i )
```

Link(inName as String) as VLink

Parameter:	**Description:**
inName	The Name of a Link.

Returns a Link by name.

Note: The parameter inName is case insensitive.

Example:

```
Link = db.Link( "Person" )
```

SQL Methods

```
SqlSelect(
        inQuery as String,
        inCursorLocation as EVCursorLocation = kClientSide,
        inLockType as EVLockType = kReadOnly,
        inCursorDirection as EVCursorDirection = kForwardOnly
        inBinds() as String = nil ) As VCursor
```

Parameter:	Description:
inQuery	The SQL string of a query.
inCursorLocation	The location of cusror.
inLockType	The lock type for records of a cursor.
inCursorDirection	The direction of a cursor.
inBinds	The array of binded parameters

Valentina uses SQL for database searches. This is documented separately in ValentinaSQL.pdf.

SqlSelect() method gets an SQL query as the string parameter, resolves it, then returns the resulting table as a cursor of type VCursor.

Note: When finished with a cursor, you must assign it the value nil to destroy it and free memory.

The optional parameters inCursorLocation, inLockType, inCursorDirection allow you to control the behavior of the cursor. See the documentation on Valentina Kernel.and VServer for more details.

You can set the following parameters with these values:

inCursorLocation:	kClientSide = 1,	kServerSide = 2,	kServerSideBulk = 3
inLockType:	kNoLocks = 1,	kReadOnly = 2,	kReadWrite = 3
inCursorDirection:	kForwardOnly = 1, kRandom = 2		

By default these parameters get the following values:
kClientSide, kReadOnly, kForwardOnly

For the SELECT command you can define an array of binded parameters. This is an array of strings for V4RB. See ValentinaSQL.pdf for details.

Example:

```
dim curs as VCursor
curs = db.SqlSelect( "SELECT * FROM T " )
```

Example:

```
curs = db.SqlSelect(   "SELECT * FROM T ",
                       EVCursorLocation.kServerSide,
                       EVLockType.kReadWrite,
                       EVCursorDirection.kRandom )
```

Example:

```
curs = db.SqlSelect(   "SELECT * FROM T WHERE f1 = :1, f2 > :2",
                       EVCursorLocation.kServerSide,
                       EVLockType.kReadWrite,
                       EVCursorDirection.kRandom,
                       Array("john", "25" ) )
```

```
SqlExecute(
        inQuery as String,
        inBinds() as String ) as Integer
```

Parameter:	**Description:**
inQuery	The SQL string of a query.
inBinds	The array of binded parameters

You can use this function to execute any SQL command supported by Valentina except for a command that returns a cursor as a result (e.g. SELECT). This is fully covered in the documentation on ValentinaSQL.

This returns the number of affect rows.

For commands that have an EXPR (expression) clause in the syntax, you can define an array of binded parameters. This is an array of strings for V4RB. See ValentinaSQL.pdf for details.

Note: such commands usually are INSERT, DELETE, UPDATE.

Example:

```
        recCount = db.SQLExecute( "UPDATE person SET name = 'john'
                                        WHERE name = 'jehn'" )
```

Example:

```
        dim Binds(-1) as String

        Binds.append 'john'
        Binds.append 'jehn'

        recCount = db.SQLExecute(
                        "UPDATE person SET name = :1 WHERE name = :2", Binds )
```

Example:

```
        // the same as above but more concise
        recCount = db.SQLExecute(
                        "UPDATE person SET name = :1 WHERE name = :2",
                        Array( "john", "jehn" ) )
```

IndexStyle Methods

CreateIndexStyle(inName as String) as VIndexStyle

Parameter:	**Description:**
inName	The name of an index style.

Creates a new Index Style in the database.

Example:

```
dim indStyle1 as VIndexStyle
IndexStyle1 = db.CreateIndexStyle( "myStyle" )
```

DropIndexStyle(inStyle as VIndexStyle)

Parameter:	**Description:**
inStyle	The index style to be deleted.

Deletes the specified index style from the database.

Example:

```
db.DropIndexStyle( IndexStyle1 )
```

IndexStyle(inName as String) as VIndexStyle

Parameter:	**Description:**
inName	The Name of a IndexStyle.

Returns an IndexStyle by name.

Note: The parameter Name is case insensitive.

Example:

```
IndexStyle1 = db.IndexStyle( "IndexStyle1" )
```

Encryption Methods

The VDataBase class has encryption methods that allows you to encrypt data of database as well as the structure of a database.

Encryption of the structure allows you to deny opening of your database files using any other programs based on the Valentina database.

Usually you will use one of the encryption methods of the database, though it is posible to merge both of them.

```
Encrypt(
        inKey as String
        inForData as EVDataKind = kRecordsOnly )
```

Parameter:	**Description:**
inKey	The key of encryption.
inForData	Specifies what data are encrypted.

Allows you to encrypt the database.

Using the inForData parameter you can specify what data must be encrypted.
inForData may accept following values:

kRecordsOnly - records of the database are encrypted.
kStructureOnly - the structure of the database (.vdb file) is encrypted.
kRecordsandStructure - records and the structure are encrypted with the same password.

When the function completes the work, you get an encrypted database on the disc. To future work with this database you need to assign the encryption key using the UseEncryptionKey() function.

Working time of the function is directly as the size of the database.

ATTENTION: If the key is lost there is no posibility to decrypt data.

Note:

• The database must be open.
• You can encrypt either an empty database or the database that already has records.
• All new tables/fields added in the database will be encrypted the same way.
• All new records added in the database will be encrypted.

Example:

```
db.Open()
db.Encrypt ( "key12345" )
```

Example:

```
db.Open()
db.Encrypt ( "key12345", kStructureOnly )
```

```
Decrypt(
        inKey as String
        inForData as EVDataKind = kRecordsOnly )
```

Parameter:	**Description:**
inKey	The encription key.
inForData	Specifies what data are encrypted.

Allows to decrypt the database.

If the database already has records then they are encrypted on the disc. When the function completes the work, you get the decrypted database which does not need the encryption key for access.

Working time of this function is directly as the size of the database.

Example:

```
db.Open()
db.Decrypt ( "key12345" )
```

Example:

```
db.Open()
db.Decrypt ( "key12345", kStructureOnly )
```

```
ChangeEncryptionKey(
                inOldKey as String
                inNewKey as String
                inForData as EVDataKind = kRecordsOnly )
```

Parameter:	**Description:**
inOldKey	Old encryption key.
inNewKey	New encryption key.
inForData	Specifies what data are encrypted.

Allows you to change the encryption key for the database.

Working time of this function is directly as the size of the database.

Example:

```
res =  db.ChangeEncryptionKey( "key12345", "key54321"  )
```

Example:

```
res =  db.ChangeEncryptionKey( "key12345", "key54321", kStructureOnly )
```

RequiresEncryptionKey() as boolean

Returns True if the database is encrypted, otherwise returns False.

This function can be used with programs such as Valentina Studio to check wether it is necessary to show an user the dialog for password entry.

Example:

```
res =  db.RequiresEncryptionKey()
```

UseEncryptionKey(

```
UseEncryptionKey(
        inKey as String
        inForData as EVDataKind = kRecordsOnly )
```

Parameter:	**Description:**
inKey	The encryption key.
inForData	Specifies what data are encrypted.

Informs the database what key must be used for data encryption.

Returns an error "wrong key", if you specify a wrong key of encryption.

Example:

```
db.UseEncryptionKey( "key12345" )
db.Open()
```

Examp;e:

```
db.UseEncryptionKey( "key12345", kStructureOnly )
db.Open()
```

Dump Methods

```
Dump(
        inDumpFile as FolderItem,
        inDumpType as Integer,
        inDumpData as EVDataKind  = kStructureAndRecords,
        inFormatDump as Boolean = false,
        inEncoding = "UTF-16" )
```

Parameter:	**Description:**
inDumpFile	The location of dump file.
inDumpType	The Type of dump.
inDumpData	Specify which information to dump.
inFormatDump	If TRUE then formats the dump file for human read.
inEncoding	Encoding of dump file.

Dumps all possible information about a database into a dump file.

Tip: You can use this file to recreate a database into a different location.

DumpType can be one of the following:
kSQL dump. A Text file that contains a set of INSERT commands.
kXML dump. A Text file that contains the database information in XML format.

XML dump is very useful as it allows you to safely dump a database with ObjectPtr fields. On loading this information into a new database, Valentina will automatically correct values of ObjectPtr fields in related tables. You can also use XML dump and load to compact your database.

Example:

```
dim db as VDatabase
...
db.Dump( fiXML, EVDumpType.kXML )
```

```
LoadDump(
        inDumpFile as FolderItem,
        inNewDb as FolderItem,
        inDumpType as Integer,
        inEncoding = "UTF-16" )
```

Parameter:	**Description:**
inDumpFile	The location of a dump file.
inNewDb	The location for a new database.
inDumpType	Type of a dump.
inEncoding	Encoding of dump file.

Loads the dump file into a new fresh database. This function is similar to the db.Create() function.

Note: You must use a variable of type VDatabase, but not your subclass of VDatabase! After the loading is complete, you will need to close the VDatabase and open it again as your subclass.

Example:

```
        dim db as VDatabase
        ...
        db.LoadDump(fiXML,fiNewDb, EVDumpType.kXML )
```

Utility methods

```
Diagnose(
        inVerboseLevel as EVVerboseLevel = kNone
        inFile as FolderItem = nil ) as Boolean
```

Parameter:	**Description:**
inVerboseLevel	Specify how many information to write into diagnose.
inFile	Location on disk of diagnose file.

Execute diagnose of an open database. Returns TRUE if the database is fine.

To produce a diagnose file you can specify its location on the disk.

Parameter inVerboseLevel can accept the following values:

kNone	= 0
kLow	= 1
kNormal	= 2
kHigh	= 3
kVeryHigh	= 4

Example:

```
res = db.Diagnose( kVeryHigh )
```

Class VTable

Properties

```
BOF                    as Boolean
EOF                    as Boolean
CollationAttribute( inColAttribute as EVColAttribute )
                       as EVColAttributeValue
DataBase               as VDataBase (r/o)   // Database of this BaseObject.
FieldCount             as Integer (r/o)     // (r/o) number of fields in this BaseObject
ID                     as Integer (r/o)
Name                   as String
IsEncrypted            as Boolean (r/o)
LinkCount              as Integer (r/o)
LocaleName             as String
PhysicalRecordCount as Integer (r/o)
RecID                  as Integer
RecordCount            as Integer (r/o)     // (r/o) number of logical records in this BaseObject.
StorageEncoding        as String
```

Field Methods

```
Field( inIndex as Integer ) as VField
Field( inName as String ) as VField
```

Link Methods

```
Link( inIndex as Integer ) as VLink
Link( inName as String ) as VLink
```

Record Methods

```
SetBlank(inAccess as EvValueAccess = forUpdate) // Clears a memory buffer of a BaseObject,
                                 // set nullable fields to NULL
AddRecord() as Integer           // Adds a new record with the current value of fields
DeleteRecord()                   // Deletes the current record
DeleteAllRecords( inSet as VSet = nil )     // Makes table empty, very fast.
UpdateRecord()                   // Updates an existing record with new values
UpdateAllRecords( inSet as VSet = nil)
```

Cach Methods

```
Flush()                          // Saves information of this BaseObject on disk only.
```

Navigation Methods

```
FirstRecord() as Boolean
LastRecord() as Boolean
PrevRecord() as Boolean
NextRecord() as Boolean

RecordExists(inRecID as Integer) as Boolean
```

```
// Set of handy CreateXXXField()

CreateBooleanField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VBoolean

CreateByteField( inName as String, inFlags as EVFlag = fNone,    inMethod as String = "" ) as VByte

CreateShortField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "" ) as VShort

CreateUShortField( inName as String, inFlags as EVFlag = fNone,  inMethod as String = "" ) as VUShort

CreateMediumField( inName as String, inFlags as EVFlag = fNone,  inMethod as String = "" ) as VMedium

CreateUMediumField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VUMedium

CreateLongField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VLong

CreateULongField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "" ) as VULong

CreateLLongField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "" ) as VLLong

CreateULLongField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VULLong

CreateFloatField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VFloat

CreateDoubleField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VDouble

CreateDateField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VDate

CreateTimeField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VTime

CreateDateTimeField( inName as String, inFlags as EVFlag = fNone, inMethod as String = "") as VDateTime
```

```
CreateStringField(
        inName as String,
        inMaxLength as Integer,
        inFlags as EVFlag = fNone, inMethod as String = ”") as VString

CreateVarCharField(
        inName as String,
        inMaxLength as Integer,
        inFlags as EVFlag = fNone,
        inMethod as String = ”") as VarChar


CreateFixedBinaryField(
        inName as String, inMaxLength as Integer) as VFixedBinary

CreateVarBinaryField(
        inName as String, inMaxLength as Integer) as VVarBinary


CreateBLOBField(
        inName as String, inSegmentSize as Integer) as VBLOB

CreateTextField(
        inName as String,
        inSegmentSize as Integer,
        inFlags as EVFlag = fNone,
        inMethod as String = ”") as VText

CreatePictureField(
        inName as String, inSegmentSize as Integer) as VPicture

CreateObjectPtrField(
        inName as String,
        inTarget as VTable,
        inOnDeletion as Integer = kCascade,
        inFlags as EVFlag = fNone,
        inLinkName as String  = "" ) as VObjectPtr
```

Structure Methods

```
DropField( inFld as VField )

ChangeType(
        inFld as VField,
        inNewType as EVFieldType,
        inParam1 as Integer ) as VField
```

Encryption Methods

```
UseEncryptionKey( inKey as String )
RequiresEncryptionKey() as Boolean
Encrypt( inKey as String )
Decrypt( inKey as String )

ChangeEncryptionKey(
        inOldKey as String
        inNewKey as String)
```

Dump Methods

```
Dump(
        inDumpFile as FolderItem,
        inDumpType as EVDumpType,
        inDumpData as EVDataKind,
        inFormatDump as Boolean )

LoadDump (
        inDumpFile as FolderItem,
        inDumpType as EVDumpType )
```

Selection Methods

```
SelectAllRecords as VBitSet,
SelectNoneRecords as VBitSet,

Sort(   inSet as VSet,
        inField as VField,
        inAcending as Boolean = true ) as VArraySet

Sort(   inSet as VSet,
        s1 as VSortItem,
        s2 as VSortItem = nil,
        s3 as VSortItem = nil,
        s4 as VSortItem = nil ) as VArraySet
```

Class Description

Each VTable manages a table of your database. Each VTable must have at least one field but is limited to no more than 65,535 fields.

Properties Description

BOF as Boolean

Returns TRUE if this is the first record of the Table.

Note: This property provides way used to ODBC API. Using this method you can navigate records of a Table using a the DO WHILE loop.

Example:

```
DO
        ...
        tbl.PrevRecord()
WHILE( not tbl.BOF )
```

EOF as Boolean

Returns TRUE if this is the last record of the Table.

Note: This property provides a way used to ODBC API. Using this method you can navigate records of aTable using a DO WHILE loop.

Example:

```
DO
        ...
        tbl.NextRecord()
WHILE( not tbl.EOF )
```

CollationAttribute(
inColAttribute as EVColAttribute) as EVColAttributeValue

CollationAttribute(
inColAttribute as EVColAttribute,
inColAttributeValue as EVColAttributeValue)

Set/Get the value of the specified collation attribute for this table.

Example:

```
dim v as integer
v = table.CollationAttribute( EVColAttribute.kStrength )

table.CollationAttribute( EVColAttribute.kStrength ) =
        EVColAttributeValue.kPrimary
```

Database as VDataBase (r/o)

Returns the parent database of this table.

Example:

```
db = table.Database
```

FieldCount as Integer (r/o)

Returns the number of custom fields in the table.

Example:

```
fldCount = table.FieldCount
```

ID as Integer (r/o)

Returns the unique identifer of the table.

Example:

```
id = table.ID
```

Name as String

The name of the table.

Example:

```
dim sname as string

sname = table.Name
table.Name = "NewName"
```

IsEncrypted as Boolean (r/o)

Returns TRUE if the database is encrypted.

Example:

```
encrypted = table.IsEncrypted
```

LinkCount as Integer (r/o)

Returns the number of links in the table.

Example:

```
dim LinkCount as Integer
LinkCount = table.LinkCount
```

LocaleName as String

Specifies for this table the locale name.

Example:

```
dim LocaleName as String
LocaleName = table.Locale

table.LocaleName = "en_US"
table.LocaleName = "jp_JP"
```

PhysicalRecordCount as Integer (r/o)

Returns the number of physical records in the table.

Example:

```
physRecCount = table.PhysicalRecordCount
```

RecID as Integer

Returns the unique automatically generated RecID of the current record. Range of values is 1..N, 0 - if the current record is undefined. Also you can use this property to change the current record of the Table. In case you try move to a non-existant record the current record will not be changed.

Example:

```
recID = Table.RecID

Table.RecID = RecID                // move to specific record
Table.RecID = 54                   // move to specific record

Table.RecID = Table.recID + 1      // move to next record
Table.RecID = Table.recID - 1      // move to prev record
```

RecordCount as Integer (r/o)

Returns the number of logical records in the table.

Example:

```
rcdCount = table.RecordCount
```

StorageEncoding as String

Specifies for this table the string encoding stored on disk.

Example:

```
dim Encoding as String
Encoding = table.StorageEncoding

table.StorageEncoding = "UTF-16"
```

Field Methods

Field(inIndex as Integer) as VField

Parameter:	**Description:**
inIndex	The index of the field. Starts from 1.

This method allows you to access fields of a Table by index. If the field with the specified index doesn't exist then it returns nil.

Example:

```
fld = Table.Field("LastName")
```

Field(inName as String) as VField

Parameter:	**Description:**
inName	The name of the field.

This method allows you to access fields of a Table by name. If the field with the specified index or name doesn't exist then it returns nil.

Example:

To get access to all the properties of a field you need to perform type casting:

```
dim fld as VField
dim fldString as VString

fld = boPerson.Field(1)
if( fld.type = kTypeString ) then
        fldString = VString( fld )
        // now you can access properties of VString field:
        // MaxLength, Language,...  using fldString
end if
```

This fragment of code can also be written using the REALbasic operator isA:

```
fld = boPerson.Field(1)
if( fld isA VString  ) then
        fldString = VString( fld )
end if
```

Link Methods

Link(inIndex as Integer) as VLink

Parameter:	**Description:**
inIndex	The index of a link.

Returns a link of this table by numeric index.

Example:

```
link = tbl.Link( i )
```

Link(inName as String) as VLink

Parameter:	**Description:**
inName	The name of a link.

Returns a link of this table by name.

Example:

```
link = tbl.Link( "link1" )
```

Record Methods

SetBlank(inAccess as EvValueAccess = forUpdate)

Parameter	**Description**
inAccess	Specify if you do SetBlank for add or for update of record.

Each VTable has a memory buffer in RAM for field values of the current record. This buffer can be cleared by the SetBlank() method, i.e. all numeric fields become zero, all string fields get an empty string. If any fields are nullable then they get a NULL value.

Parameter inAccess can be used to speed up SetBlank() if you add records. In this case you can specify its value forAdd, so Valentina will not save copies of previouse field values. In the same time you can always use the default value forUpdate and everyhting will work correctly.

Example:

```
Table.SetBlank()
```

AddRecord() as Integer

Adds a new record to the table with the current values in the memory buffer of this Table. Returns the RecID of the new record.

Note: You need to assign values to the fields for the new record, then call AddRecord().

Example:

```
thePerson.SetBlank
thePerson.FirstName.Value = “John”
thePerson.LastName.Value = “Roberts”
NewRecID = thePerson.AddRecord()
```

DeleteRecord()

Deletes the current record of a Table.

After deletion, the next record becomes the current one if it exists. Otherwise the previous record becomes current. If a Cursor becomes empty then the current record will be undefined.

Example:

```
Table.DeleteRecord()
```

DeleteAllRecords(inSet as VSet = nil)

Parameter	**Description**
inSet	The selection of records.

Deletes all records in a Table if inSet is nil. Otherwise deletes only the specified selection of records.

Example:

```
Table.DeleteAllRecords()
```

UpdateRecord()

This method stores new modified values of fields of the current record of the Table.

Example:

```
thePerson.RecID = SomeRecID
thePerson.FirstName.Value = “Brian”
thePerson.LastName.Value = “Blood”
thePerson.UpdateRecord()
```

UpdateAllRecords(inSet as VSet = nil)

Parameter	**Description**
inSet	The selection of records.

Updates all records in a Table if inSet is nil. Otherwise updates only the specified selection of records.

Example:

```
Table.UpdateAllRecords()
```

Cache Methods

Flush()

This method flushes all unsaved information of the Table from the cache to disk.

Note:This can be faster than VDataBase.Flush() because it affects data from only one Table.

Example:

```
Table.Flush()
```

Navigation Methods

FirstRecord() as Boolean

Goes to the first logical record of a Table. Reads the record from disk to the memory buffer of a Table.
Returns TRUE if the first record is found.
Returns FALSE if the current record already was the first or the Table is empty.

Example:

```
res = Table.FirstRecord()
```

LastRecord() as Boolean

Goes to the last logical record of a Table. Reads a record from disk to the memory buffer of a Table.
Returns TRUE if the last record is found.
Returns FALSE if the current record already was the last or the Table is empty.

Example:

```
res = Table.LastRecord()
```

PrevRecord() as Boolean

Goes to the previous logical record of a Table. Reads a record from disk to the memory buffer of a Table.
Returns TRUE if the previous record is found.
Returns FALSE if the current record was the first or the Table is empty.

Example:

```
res = Table.PrevRecord()
```

NextRecord() as Boolean

Goes to the next logical record of a Table.
Reads a record from disk to the memory buffer of a Table.
This returns TRUE if the next record is found, or FALSE if the current record was the last or the Table is empty.

Example:

```
res = Table.NextRecord()
```

RecordExists(inRecID as Integer) as Boolean

Parameter	**Description**
inRecID	RecID of a record.

Returns TRUE if the record with the specified RecID exists in the table.

Example:

```
res = Table.RecordExists( RecID )
```

Working with Database Structure

The Valentina API for REALbasic lets you not only create or work with static database structures but also exposes you to methods for creating dynamic database structures. This is also very useful for when you upgrade your database application and need dynamically update the database structure to support new features in your application.

Valentina for REALbasic provides the set of methods to create fields. There exists several groups of methods which have similar parameters. So we will describe the groups of these methods.

Methods to create numeric fields

```
CreateShortField(
        inName as string,
        inFlags as EVFlag = fNone,
        inMethod as String = ”" ) as VShort
```

Parameter:	**Description:**
inName	The name of the field.
inFlags	The flags of the field.
inMethod	The text of the method for a calculation field.

Create a numeric field of the corresponding type. The full list of methods you can see in the section describing the VTable Class.

• To create a field you should specify its name.
• You can specify flags for a field to modify its behavior.
• If you want to create a calculated field then you should specify the method text.

Example:

```
        fldAge = tblPerson.CreateShortField(
                "age",  EVFlags.fNullable + EVFlags.fIndexed )
```

Methods to create string/varchar fields

```
CreateStringField(
        inName as String,
        inMaxLength as Integer,
        inFlags as EVFlag = fNone,
        inMethod as String = ””) as VString

CreateVarCharField(
        inName as String,
        inMaxLength as Integer,
        inFlags as EVFlag = fNone,
        inMethod as String = ””) as VVarChar
```

Parameter:	**Description:**
inName	The name of the field.
inMaxLength	The maximum length (in characters)
inFlags	The flags of the field.
inMethod	The text of the method for a calculation field.

Creates a String or VarChar field.
• You need to specify the maximum length in characters. In the case of UTF16 encoding, then 2 bytes per char will be used. If you use a single byte encoding, then one byte per character will be used. You can specify flags for a field to modify its behavior.
• You can specify flags for a field to modify its behavior.
• If you want to create a calculated field then you should specify the method text.

Example

```
        fldAge = tblPerson.CreateStringField(
                    "name", 40, EVFlags.fNullable + EVFlags.fIndexed )
```

Methods to create fixed/var binary fields

```
CreateFixedBinaryField(
        inName as String,
        inMaxLength as Integer) as VFixedBinary

CreateVarBinaryField(
        inName as String,
        inMaxLength as Integer) as VVarBinary
```

Parameter:	**Description:**
inName	The name of the field.
inMaxLength	The maximum length (in bytes)

Create a fixed or variable size binary field.
• You need to specify the maximum length in bytes.

Example

```
        fldAge = tblPerson.FixedBinaryField(
                    "nameStile", 40,  EVFlags.fNullable + EVFlags.fIndexed )
```

Method to create BLOB fields.

```
CreateBLOBField(
        inName as String,
        inSegmentSize as Integer) as VBLOB
```

Parameter:	**Description:**
inName	The name of the field.
inSegmentSize	The segment size of the BLOB field.

Create a BLOB (Binary Large Object) field.

• You need to specify the segment size in bytes.

Example

```
        fldAge = tblPerson.CreateBLOBField(
                        "notesStyle",  256 )
```

Method to create TEXT fields.

```
CreateTextField(
        inName as String,
        inSegmentSize as Integer,
        inFlags as EVFlag = fNone,
        inMethod as String = ””) as VText
```

Parameter:	**Description:**
inName	The name of the field.
inSegmentSize	The segment size of the BLOB field.
inFlags	The flags of the field.
inMethod	The text of the method for a calculation field.

Create a Text field.

• You need to specify the segment size in bytes.
• You can specify flags for a field to modify its behavior.
• If you want to create a calculated field then you should specify the method text.

Example

```
        fldAge = tblPerson.CreateTextField(
                        "notes", 256, EVFlags.fNullable + EVFlags.fIndexed )
```

Method to create Picture fields.

```
CreatePictureField(
        inName as String,
        inSegmentSize as Integer) as VPicture
```

Parameter:	**Description:**
inName	The name of the field.
inSegmentSize	The segment size of the field.

Create a picture field. You need to specify the segment size in bytes.

Example

```
        fldAge = tblPerson.CreatePictureField(
                        "foto", 256, EVFlags.fNullable + EVFlags.fIndexed )
```

Method to create ObjectPtr fields.

```
CreateObjectPtrField(
        inName as String,
        inTarget as VTable,
        inOnDeletion as Integer = 2,
        inFlags as EVFlag = fNone,
        inLinkName as String = "" ) as VObjectPtr
```

Parameter:	**Description:**
inName	The name of the field.
inTarget	The target table.
inOnDeletion	The behavior on deletion of the record-owner.
inFlags	The flags of the field.
inLinkName	The link name for this ObjectPtr-link.

Create an ObjectPtr field.

• You need to specify a target table and deletion control.
• You can specify flags for a field to modify its behavior.

Example

```
        fldAge = tblPerson.CreateObjectPtrField(
                        "ParentPtr",  EVFlags.fNullable + EVFlags.fIndexed )
```

DropField(inFld as VField)

Parameter:	**Description:**
inFld	The field that should be deleted.

Removes the referenced field (column) from a Table. This operation is undoable! It will occur instantaneously for a Table with any number of records.

Example:

```
Table.DropField( fld )
```

ChangeType(
inFld as VField,
inNewType as EVFieldType,
inParam1 as Integer) as VField

Parameter:	**Description:**
inFld	The field whose type should be changed.
inNewType	New type for a field.
inParam	The Additional parameter (see below).

Sometimes you may need to change the type of a field. For example, if you first made a field “Quantity” as VUShort and later you have found that in real life the quantity might be more than 65’535, you will need to change its type into VULong.

For String and VarChar fields inParam is MaxLength.
For BLOB an its subtypes (Text, Picture) in Param is SegmentSize.
For all remaining types of fields, in Param is ignored and should be zero.

Example:

```
fld = Table.ChangeType( fld, EVFieldType.kTypeString,40 )
```

VTable Encryption Methods

The VTable class has a set of functions for encryption analog to functions of the VDatabase and VField classes.

You may wish to use these functions if you want to encrypt only one or several Tables of a database. It gains speed improvements over having to encrypt an entire database.

Notice, you can not specify the own encryption key for a Table in case if its database is encrypted before.

Encrypt(inKey as String)

Parameter:	**Description:**
inKey	The encryption key.

Allows you to encrypt the Table.

When the function completes work, you get an encrypted Table on the disc. To future work with this Table you need to assign the encryption key using the UseEncryptionKey() function.

Working time of the function is directly as the size of the Table.

ATTENTION: If the key is lost there is no posibility to decrypt data.

Example:

```
tbl.Encrypt( "key12345" )
```

Decrypt(inKey as String)

Parameter:	**Description:**
inKey	The encryption key.

Allows to decrypt the Table.

If the Table already has records then they are decrypted on the disc. When the function completes the work, you get the decrypted Table which does not need the encryption key for access.

Working time of this function is directly as the size of the Table.

Example:

```
tbl.Decrypt( "key12345" )
```

```
ChangeEncryptionKey(
        inOldKey as String,
        inNewKey as String )
```

Параметр:	**Описание:**
inOldKey	The encryption key.
inNewKey	New encryption key.

Allows you to change the encryption key fot the Table.

Working time of this function is directly as the size of the Table.

Example:

```
        tbl.ChangeEncryptionKey( "key12345", "key54321"  )
```

RequiresEncryptionKey() as Boolean

Returns True if the Table is encrypted with the own encryption key, otherwise it returns False.

ATTENTION: if you encrypt the entire database than this method will return False for its Tables.

This function can be used with programs such as Valentina Studio to check wether it is necessary to show an user the dialog for password entry.

Example:

```
res = tbl.RequiresEncryptionKey()
```

UseEncryptionKey(inKey as String)

Parameter:	**Description:**
inKey	The encryption key

Informs the database what key must be used for data encryption.

Returns an error "wrong key", if you specify a wrong key of encryption.

This function must be called just if VTable.RequiresEncryptionKey() returns True for this Table.

ATTENTION: while the VDatabase.UseEncryptionKey() method must be called before opening of the database, the VTable.UseEncryptionKey() methods must be called after opening the database and before the first attempt to work with data of the Table.

Example:

```
db.UseEncryptionKey( "key12345" )
db.Open()

tbl.UseEncryptionKey( "key12345" )
```

Dump Methods

```
Dump(
        inDumpFile as FolderItem,
        inDumpType as EVDumpType,
        inDumpData as EVDataKind,
        inFormatDump as Boolean )
```

Parameter	**Description**
inDumpFile	The location of the dump file.
inDumpType	The Type of dump.
inDumpData	Specify which information to dump.
inFormatDump	If TRUE then formats the dump file to be human readable.

Dumps the table to a file in XML or SQL format.

Example:

```
dim tbl as VTable
...
tbl.Dump( fiXML, EVDumpType.kXML )
```

```
LoadDump(
        inDumpFile as FolderItem,
        inDumpType as EVDumpType )
```

Parameter	**Description**
inDumpFile	The location of the dump file.
inDumpType	The type of dump.

Loads a XML or SQL dump from the specified file into the Table.

Example:

```
dim tbl as VTable
...
tbl.loadDump(fiXML,  EVDumpType.kXML )
```

Selection Methods

SelectAllRecords as VBitSet

Returns a selection of all records of a table as a VBitSet.

Example:

```
allRecs = Table.SelectAllRecords()
```

SelectNoneRecords as VBitSet

Returns a VBitSet, which contains no records of a table. The size of the VBitSet is equal to the number of physical records in the table.

Example:

```
NoneRecs = Table.SelectNoneRecords()
```

```
Sort(
        inSet as VSet,
        inField as VField,
        inAscending as boolean = true ) as VArraySet
```

Parameter:	**Description:**
inSet	The set of records to be sorted. .
inField	The field on which to do sorting.
inAscending	The direction of sorting.

Executes sorting of the selection inSet by the field inField. The parameter inAscending specifies the order of sorting.

Returns a new sorted selection as an ArraySet.

Example:

```
        SortedSet = table.Sort( allRecs, fldName )
```

```
Sort(   inSet as VSet,
        s1 as VSortItem,
        s2 as VSortItem = nil,
        s3 as VSortItem = nil,
        s4 as VSortItem = nil ) as VArraySet
```

Parameter:	**Description:**
inSet	The set to be sorted.
s1	Description of the first sorted field.
s2	Description of the second sorted field.
s3	Description of the third sorted field.
s4	Description of the fourth sorted field.

Executes sorting of a table selection inSet on several fields (up to 4).

Example:

```
        SortedSet = table.Sort(
                allRecs, new SortItem(fldName), new SortItem(fldLastName) )
```

Class VField

Properties

CollationAttribute(inColAttribute as EVColAttribute) as EVColAttributeValue

DefaultValue	asVariant	
ID	as Integer (r/o)	
IndexStyle	as VIndexStyle	
IsEncrypted	as Boolean (r/o)	
IsIndexed	as Boolean	
IsMethod	as Boolean (r/o)	// TRUE if the field is a method.
IsNullable	as Boolean	// TRUE if the field accepts NULL values
IsNull	as Boolean	// TRUE if the current value of the field is NULL.
IsUnique	as Boolean	// TRUE the field only has unique values
LocaleName	as String	
MethodText	as String	
Name	as String	// up to 32 bytes
StorageEncoding	as String	
Table	as VTable (r/o)	
Type	as EVFieldType (r/o)	
TypeString	as String (r/o)	
Value	as Variant	

Value methods

```
SetBlank()                          // clear the value of the field.

GetString() as String               // returns a value of the Field as a String
SetString( inValue as String )      // store a String value in the Field
```

Search methods

```
ValueExists(
        inValue as Variant,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as Boolean

ValueExists(
        inValue as Variant,
        ByRef outCount as Integer,
        inSelection as VSet = nill,
        inSearchPref as EvSearch = kPreferIndexed ) as Boolean

FindValue(
        inValue as Variant,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet

FindValueAsArraySet(
        inValue as Variant,
        inSelection as VSet = nil,
        inMaxCount as integer = &hffffffff,     // ulong_max
        inSearchPref as EvSearch = kPreferIndexed ) as VArraySet
```

```
FindRange(
        inLeftInclude as Boolean,
        inLeftValue as Variant,
        inRightValue as Variant,
        inRightInclude as Boolean,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet

FindRangeAsArraySet(
        inLeftInclude as Boolean,
        inLeftValue as Variant,
        inRightValue as Variant,
        inRightInclude as Boolean,
        inSelection as VSet = nil,
        inMaxCount as integer = &hffffffff,     // ulong_max
        inSearchPref as EvSearch = kPreferIndexed ) as VArraySet

FindSingle(
        inValue as Variant,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as Integer

FindNulls(
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet

FindNotNulls(
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet

FindStartsWith(
        inValue as String,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet

FindContains(
        inValue as String,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet

FindEndsWith(
        inValue as String,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet

FindRegEx (
        inValue as String,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

```
FindLike(
        inValue as String,
        inEscapeChar as String = "\",
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet

FindDistinct(
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Encryption methods

```
UseEncryptionKey(inKey as String)
RequiresEncryptionKey() as Boolean
Encrypt(inKey as String)
Decrypt(inKey as String)

ChangeEncryptionKey(
        inOldKey as String
        inNewKey as String)
```

Class Description

This is the base abstract class for all other types of fields, so you will never create an instance of it. Each field must have an unique name (case insensitive) in the scope of a Table.

Using VTable.Field() or VCursor. Field(), you can get a reference of VField. There is no real difference between a VField of a Table and a VField of a Cursor.

If you need to get access to properties of VField subclasses, then you need to do type casting to that subclass.

For example, if you have the reference of a string Field and want to access the property MaxLength of the class VString:

```
dim fld as VField
dim str_fld as VString

fld = Person.Field( “Name” )
str_fld = VString( fld )
if( str_fld <> nil )
        maxLen = str_fld.MaxLength
end if
```

Properties Description

CollationAttribute(inColAttribute as EVColAttribute) as EVColAttributeValue

The value of the specified collation attribute for this table.

Example:

```
v = fld.CollationAttribute( EVColAttribute.kStrength )

fld.CollationAttribute( EVColAttribute.kStrength ) = EVColAttributeValue.kPrimary
```

DefaultValue asVariant

The default value of the field. This value is used when you INSERT a new record into the table, but do not specify a value for this field. By default this property is nil.

Example:

```
v = fld.DefaultValue
```

ID as Integer (r/o)

Return the unique identifier of the field.

Example:

```
id = fld.ID
```

IndexStyle as VIndexStyle

Specifies the index style for this field. You can use this property to assign/change the index style of a field. Also you can check the current index style of the field.

Example:

```
fld.IndexStyle = style1

currStyle = fld.IndexStyle
```

IsEncrypted as Boolean (r/o)

Returns TRUE if the database is encrypted.

Example:

```
encrypted = fld.IsEncrypted
```

IsIndexed as Boolean

If TRUE then Valentina will maintain an index for this field. This property can be changed at runtime.

Example:

```
fld.IsIndexed = FALSE
        ... // add many records for example
fld.Indexed = TRUE
```

IsMethod as Boolean (r/o)

TRUE if the field is virtual, i.e. it is a Table Method.
Read Only.

Example:

```
if( fld.IsMethod )
```

IsNullable as Boolean

If TRUE then this field can have a NULL value. In this case 1 bit per record is added.

Example:

```
fld.IsNullable = TRUE
if( fld.Nullable )
```

IsNull as Boolean

This is a record property. It is TRUE if the value of this field for the current record of the table is NULL.

NOTE: don’t confuse it with the property of isNullable! isNullable is a property of the column of a table, IsNull is a property of the current record.

Example:

```
....
curs.Position = i
if( curs.Field(1).IsNull ) then
        ....
```

IsUnique as Boolean

If TRUE then this field will not accept duplicate entries. Also, if the field is unique then it is automatically indexed.

Example:

```
fld.IsUnique = TRUE
if( fld.Unique )
```

LocaleName as String

Specifies the locale name for this field.

Example:

```
LocaleName = fld.LocaleName

fld.LocaleName = "en_US"
fld.LocaleName = "jp_JP"
```

MethodText as String

Returns the text of the field method. Also you can use this property to change the text of the field method.

Example:

```
method = fld.MethodText

fld.MethodText = “CONCAT(FirstName, ‘ ‘, LastName)”
```

Name as String

Each field has a unique name in the scope of a Table. The maximum length of the name is 32 bytes.

Example:

```
name = fld.Name
fld.Name = "last"
```

StorageEncoding as String

Specifies for this table the encoding of strings stored on disk.

Example:

```
Encoding = fld.StorageEncoding

fld.StorageEncoding = "UTF-16"
```

Table as VTable (r/o)

Returns the Table of this field.

Example:

```
t = fld.Table
```

Type as EVFieldType (r/o)

Each field has a type, which defines the context of data which can be stored in it. The type of a field is defined when you use a constructor of a subclass of VField.

Each field has several flags, which define its behavior:

Example:

```
case fld.Type
```

See also: VTable.ChangeType

TypeString as String (r/o)

Returns the type of this field as a string. This can be used in GUI tools.

Example:

```
strType = fld.TypeString
```

Value as Variant

The VField class has a property Value of the general kind called a VARINAT. This means that you can easily get/set value of any field type using this property.

Also note that each subclass of VField class has its own property Value of corresponding type. When REALbasic and Valentina know the exact type of value they work faster. So if you care about speed you should prefer to use the Value of subclasses.

Example:

```
dim f as VField
dim iv as integer

f.value = 5
iv = f.value
```

Value Methods

SetBlank()

Clears the value of a field.
- If the field has a default value then set its value to default.
- Otherwise If the field is Nullable, then set its value to NULL.
- Otherwise for a numeric field, set it to zero; for String fields, set it to an empty string.

Example:

```
fld.SetBlank()
```

GetString() as String

Returns the value of the field as a string.

Example:

```
str = fld.GetString()
```

SetString(inValue as String)

Parameter:	**Description:**
inValue	New value for the field.

Sets a field value using strings, regardless of the assigned field type. When assigning a value to a field, Valentina will convert the string into the appropriate type.

If you develop an application with a dynamic database structure, then you will use these methods instead of the Value property of the appropriate field class.

Example:

```
str = "aaaaa"
...

fld.SetString( str )
```

Search Methods

```
ValueExists(
        inValue as Variant,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as Boolean
```

Parameter:	**Description:**
inValue	The value to search.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Check if the specified value exists in the specified selection of the records. Returns TRUE if at least one record has a value equal to inValue.

If inSelection is nil then it searches all records of the table. Otherwise it searches only records in the specified selection.

Example:

```
        found = fld.ValueExists( 5 )
        found = fld.ValueExists( 5, S )
```

```
ValueExists(
        inValue as Variant,
        ByRef outCount as Integer,
        inSelection as VSet = nill,
        inSearchPref as EvSearch = kPreferIndexed ) as boolean
```

Parameter:	**Description:**
inValue	The value to search.
outCount	The count of records that match inValue.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Does the same as the above method ValueExists, but also calculates the count of records that match. So this function requires more time.

Example:

```
        dim count as integer
        found = fld.ValueExists( 5, count )
        found = fld.ValueExists( 5, count, S )
```

```
FindValue(
        inValue as Variant,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inValue	The value to search.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Finds the specified value in the selection of records. Returns a BitSet of found records.

If inSelection is nil then it searches all records of the table. Otherwise it searches only records the specified selection.

Note: You should prefer to use this function in the case where you expect a large number of found records. Otherwise it is better to use "FindValueAsArraySet()".

Example:

```
        dim s1 as VBitSet
        s1 = fld1.FindValue(5)
        s2 = fld2.FindValue( 7, s1 )
```

```
FindValueAsArraySet(
        inValue as Variant,
        inSelection as VSet = nil,
        inMaxCount as integer = &hffffffff,
        inSearchPref as EvSearch = kPreferIndexed ) as VArraySet
```

Parameter:	**Description:**
inValue	The value to search
inSelection	Selection of records.
inMaxCount	The maximum number of records to return.
inSearchPref	Specifies if the search should use index.

Does the same as the previous function but returns the selection as an ArraySet.

Note: You should prefer to use this function in the case where you expect a relatively small number of found records. Otherwise it is better to use "FindValue()". Also using parameter inMaxCount you can even reduce the number of returned records if you need.

Example:

```
        dim s1 as VArraySet
        s1 = fld1.FindValueAsArraySet(5)
        s2 = fld2.FindValueAsArraySet( 7, s1 )
```

```
FindRange(
        inLeftInclude as boolean,
        inLeftValue as Variant,
        inRightValue as Variant,
        inRightInclude as boolean,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inleftInclude	TRUE if the left value of the range must be included.
inLeftValue	The left value of the range.
inRightValue	TRUE if the right value of the range must be included.
inrRightInclude	The right value of the range.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Finds the records which have values that fit into the specified range of values. Returns a BitSet of found records.

The range of values is defined in a mathematical way, e.g. [leftValue, rightValue] or (left-Value, rightValue). Parameters LeftInclude and RightInclude specify if the end points of range should be included or excluded.

If inSelection is nil then it searches all records of the table. Otherwise it searches only records the specified selection.

Note: You should prefer to use this function in case you expect a large number of found records. Otherwise it is better to use "FindRangeAsArraySet()".

Example:

```
s1 = fld1.FindRange( true , 5, 8, true )        // [5, 8]
s1 = fld1.FindRange( false, 5, 8, true )        // (5, 8]
s1 = fld1.FindRange( true , 5, 8, false )       // [5, 8)
s1 = fld1.FindRange( false, 5, 8, false )       // (5, 8)
```

```
FindRangeAsArraySet(
        inLeftInclude as boolean,
        inLeftValue as Variant,
        inRightValue as Variant,
        inRightInclude as boolean,
        inSelection as VSet = nil,
        inMaxCount as integer = &hffffffff,
        inSearchPref as EvSearch = kPreferIndexed ) as VArraySet
```

Parameter:	**Description:**
inleftInclude	TRUE if the left value of the range must be included.
inLeftValue	The left value of the range.
inRightValue	TRUE if the right value of the range must be included.
inrRightInclude	The right value of the range.
inSelection	Selection of records.
inMaxCount	The maximum number of records to return.
inSearchPref	Specifies if the search should use index.

Does the same as the previous function but returns the selection as an ArraySet.

Note: You should prefer to use this function in the case where you expect a relatively small number of found records. Otherwise it is better to use "FindRange()". Using parameter inMaxCount you can even reduce the number of returned records if you need.

Example:

```
        s1 = fld1.FindRangeAsArraySet( true , 5, 8, true )        // [5, 8]
        s1 = fld1.FindRangeAsArraySet( false, 5, 8, true )        // (5, 8]
        s1 = fld1.FindRangeAsArraySet( true , 5, 8, false )       // [5, 8)
        s1 = fld1.FindRangeAsArraySet( false, 5, 8, false )       // (5, 8)
```

```
FindSingle(
        inValue as Variant,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as Integer
```

Parameter:	**Description:**
inValue	The value to search
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Finds the specified value in the selection of records. Returns the RecID of the first found record that matches. You should use this function only if you are sure that you will find one record. The advantage of this function is that you avoid the overhead of Sets.

If inSelection is nil then it searches all records of the table. Otherwise it searches only records the specified selection.

Example:

```
        foundRecID = fld.FindSingle( 5 )
```

```
FindDistinct(
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns selection that contains only distinct values.

If inSelection is nil then it searches all records of the table. Otherwise it searches only records of the specified selection.

Example:

```
        dim bset as VBitSet
        bset = fld.FindDistinct()
```

```
FindNulls(
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns all records of the specified selection that have NULL values.

If inSelection is nil then it searches all records of the table. Otherwise it searches only records of the specified selection.

Example:

```
        dim bset as VBitSet
        bset = fld.FindNulls()
```

```
FindNotNulls(
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns all records of the specified selection that have NOT NULL values.

If inSelection is nil then it searches all records of the table. Otherwise it searches only records of the specified selection.

Example:

```
        dim bset as VBitSet
        bset = fld.FindNotNulls()
```

String Search Methods

The following methods perform String searches on field values. These functions work for any field type that can convert its value to a String. The result of a comparison depends on the current Collation settings for this field.

All these functions have the optional parameter inSelection. If it is nil then all records of table are searched. Otherwise only records of the specified selection are searched.

```
FindStartsWith(
        inValue as String,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inValue	The search String.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns all records of the specified selection which have field value that starts with the specified String.

Note: see additional description at the start of this paragraph.

Example:

```
dim bset as VBitSet
bset = fld.FindStartsWith( "Jo" )
```

```
FindContains(
        inValue as String,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inValue	The search String.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns all records of the specified selection which have a field value that contains the specified String.

Note: see additional description at the start of this paragraph.

Example:

```
dim bset as VBitSet
bset = fld.FindContains( "Jo" )
```

```
FindEndsWith(
        inValue as String,
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inValue	The search String.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns all records of the specified selection which have a field value that ends with the specified String.

Note: see additional description at the start of this paragraph.

Example:

```
        dim bset as VBitSet
        bset = fld.FindEndsWith( "hn" )
```

```
FindLike(
        inValue as String,
        inEscapeChar as String = "\",
        inSelection as VSet = nil,
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inValue	The search String.
inEscapeChar	The character to be used as escape character.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns all records of the specified selection which have a field value that matches the SQL search WHERE fld LIKE 'str'.

Note: see additional description at the start of this paragraph.

Example:

```
        dim bset as VBitSet
        bset = fld.FindLike( "%eter" )
```

```
FindRegEx (
        inValue as String,
        inSelection as VSet = nil
        inSearchPref as EvSearch = kPreferIndexed ) as VBitSet
```

Parameter:	**Description:**
inValue	The search String.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns all records of the specified selection which have a field value that matches the SQL search WHERE fld REGEX 'str'.

Note: see additional description at the start of this paragraph.

Example:

```
        dim bset as VBitSet
        bset = fld.FindRegEx( "Pe?" )
```

VField Encryption Methods

The VField class has a set of functions for encryption analog to functions of the VDatabase and VTable classes.

You may wish to use these functions if you want to encrypt only one or several Fields of a database. It gains speed improvements over having to encrypt an entire database.

Notice, you can not specify a special encryption key for a Field in case if its database is encrypted before.

Encrypt(inKey as String)

Parameter:	**Description:**
inKey	The encryption key.

Allows you to encrypt the separate Field in the table.

When the function completes the work, you get an encrypted Field on the disc. To future work with this Field you need to assign the encryption key using the UseEncryptionKey() function.

Working time of the function is directly as the size of the Field.

ATTENTION!!! if the key is lost there is no posibility to decrypt data.

Example:

```
fld.Encrypt( "key12345" )
```

Decrypt(inKey as String)

Parameter:	**Description:**
inKey	The encryption key.

Allows to decrypt the Field in the table.

If the Field already has records then they are decrypted on the disc. When the function completes the work, you get the decrypted Field which does not need the encryption key for access.

Working time of this function is directly as the size of the Field.

Example:

```
fld.Decrypt( "key12345" )
```

```
ChangeEncryptionKey(
        inOldKey as String,
        inNewKey as String )
```

Параметр:	**Описание:**
inOldKey	The encryption key.
inNewKey	New encryption key.

Allows you to change the encryption key for the Field.

Working time of this function is directly as the size of the Field.

Example:

```
fld.ChangeEncryptionKey( "key12345", "key54321" )
```

RequiresEncryptionKey() as Boolean

Returns True if the Field is encrypted with the own encryption key, otherwise it returns False.

ATTENTION: if you encrypt the entire database than this method will return the False for its Fields.

This function can be used with programs such as Valentina Studio to check wether it is necessary to show an user the dialog for password entry.

Example:

```
res =fld.RequiresEncryptionKey()
```

UseEncryptionKey(inKey as String)

Parameter:	**Description:**
inKey	The encryption key.

Informs the database what encryption key must be used for data encryption.

Returns an error "wrong key", if you specify a wrong key of encryption.

This function must be called just if VField.RequiresEncryptionKey() returns True for this Field.

ATTENTION: while the VDatabase.UseEncryptionKey() method must be called before opening of the database, the VField.UseEncryptionKey() methods must be called after opening the database and before the first attempt to work with data of the Field.

Example:

```
db.UseEncryptionKey( "key12345" )
db.Open()

fld.UseEncryptionKey( "key12345" )
```

Numeric Fields

Valentina for REALbasic has a set of classes that represent numeric field types. All of these classes are subclasses of the VField class, so they inherit all the properties and methods of the VField class.

These classes are quite small. They have just a constructor and a property 'Value' of the corresponding type. For example, the VBoolean field returns a boolean, while a VDouble field returns a double.

Each class has a constructor where you should specify:
• the name of the field,
• the flags for this field
• the text of the method for calculated fields.

Example of constructor declarations

```
VBoolean   ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VShort     ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VUShort    ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VMedium    ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VUMedium ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VLong      ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VULong     ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VLLong     ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VULLong    ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VFloat     ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
VDouble    ( Name as a String, Flags as an Integer = fNone, inMethod as String = "" )
```

You will need to use these classes directly:
• if you want to use Classes way in your project.
• if you want to perform type casting from a VField to one of the Numeric types.

Example:

```
fld = new VByte( “byte_fld”,  EVFlags.fNone )
fld = new VByte( “byte_fld”,  EVFlags.fIndexed )
fld = new VByte( “byte_fld”,  EVFlags.fIndexed + EVFlags.fUnique )
fld = new VByte( “byte_fld”,  EVFlags.fIndexed + EVFlags.fNullable )
```

Class VDate

Properties

```
Day          as Integer                 // 1..31
Month        as Integer                 // 1.12
Year         as Integer                 // any year between -222..+222
```

Constructor

```
VDate(
        inName as String,
        inFlags as Integer = fNone,
        inMethod  as String = "" )
```

Method

```
Set(
        inYear as Integer,
        inMonth as Integer,
        inDay as Integer)

SetDate( inDate as Date )
GetDate() as Date
```

VDate Methods

Set(

```
inYear as Integer,
inMonth as Integer,
inDay as Integer)
```

Parameter:	**Description:**
inYear	The Year of a new value.
inMonth	The Month of a new value.
inDay	The Day of a new value.

Set the value of date field.

Example:

```
fldDate.Set( 1972, 3, 20 )
```

SetDate(inDate as Date)

Parameter:	**Description:**
inDate	The Date of a new value.

This function set value of VDate field with help of native REALbasic date value.

Example:

```
Dim myDate As Date
myDate = New Date
fldDate.SetDate( myDate )
```

GetDate() as Date

This function get value from VDate field into native REALbasic date value.

Example:

```
Dim myDate As Date
myDate = fldDate.GetDate()
```

Class VTime

Properties

```
Hour            as Integer          // 0..23
Minute          as Integer          // 0..59
Second          as Integer          // 0..59
MilliSecond     as Integer
```

Constructor

```
VTime(
        inName as String,
        inFlags as Integer = fNone,
        inMethod as string = "" )
```

Method

```
Set(
        inHour as Integer,
        inMinute as Integer,
        inSecond as Integer)

SetTime( inDate as Date )
GetTime() as Date
```

VTime Methods

Set(

```
Set(
        inHour as Integer,
        inMinute as Integer,
        inSecond as Integer)
```

Parameter:	**Description:**
inHour	Hours of a new value.
inMinute	Minutes of a new value.
inSecond	Seconds of a new value.

The classes VDate and VTime differ from the group of numeric fields in that they have a complex “Value” represented by several properties.

Also, they have the method Set() that allows for setting all three properties in one call.

Example:

```
fldTime.Set( 7, 20, 0 )
```

SetTime(inDate as Date)

Parameter:	**Description:**
inDate	The Date of a new value.

This function set value of VTime field with help of native REALbasic date value.

Example:

```
Dim myDate As Date
myDate = New Date
fldTime.SetTime( myDate )
```

GetTime() as Date

This function get value from VTime field into native REALbasic date value.

Example:

```
Dim myDate As Date
myDate = fldTime.GetTime()
```

Class VDateTime

Properties

```
Day             as Integer          // 1..31
Hour            as Integer          // 0..23
Month           as Integer          // 1.12
Minute          as Integer          // 0..59
Second          as Integer          // 0..59
Year            as Integer          // any year between -222..+222
Millisecond     as Integer
```

Constructor

```
VDateTime(
        inName as String,
        inFlags as InInteger = fNone,
        inMethod as String = "" )
```

Methods

```
SetDate(
        inYear as Integer,
        inMonth as Integer,
        inDay as Integer)

SetTime(
        inHour as Integer,
        inMinute as Integer,
        inSecond as Integer)


SetDateTime( inDate as Date )
GetDateTime() as Date
```

VDateTime Methods

SetDate(inYear as Integer, inMonth as Integer, inDay as Integer)

Parameter:	**Description:**
inYear	The Year of a new value.
inMonth	The Month of a new value.
inDay	The Day of a new value.

Sets the day, month and year.

Example:

```
fldDataTime.SetDate( 1972, 03, 20 )
```

SetTime(inHour as Integer, inMinute as Integer, inSecond as Integer)

Parameter:	**Description:**
inHour	Hours of a new value.
inMinute	Minutes of a new value.
inSecond	Seconds of a new value.

Sets the time of day.

Example:

```
fldDataTime.SetTime( 7, 20, 00 )
```

SetDateTime(inDate as Date)

Parameter:	**Description:**
inDate	The Date of a new value.

This function set value of VDateTime field with help of native REALbasic date value.

Example:

```
Dim myDate As Date
myDate = New Date
fldDateTime.SetDateTime( myDate )
```

GetDateTime() as Date

This function get value from VDateTime field into native REALbasic date value.

Example:

```
Dim myDate As Date
myDate = fldDateTime.GetDateTime()
```

Class VString
Class VVarChar

Properties

MaxLength	as Integer	// the maximum length of a string which can be stored
IndexByWords	as Boolean	// if TRUE then each word of the string is indexed separately
Value	as String	

Constructor

```
VString / VVarChar(
        inName as String,
        inMaxLength as Integer,
        inFlags as Integerб
        inMethod as String = ””)
```

Class VString
Class VVarChar

Class Description

This type of field is used for storing strings in a database. VString and VVarChar classes have the same API, except for their constructors.

Properties Description

MaxLength as Integer

The Maximum length of a field can be in the range of values 1 .. 65535 bytes. It can be applied to VString, VVarChar, VFixedBinary, VVarBinary fields.

Note: If you change the maximum length of the field, then you also are changing a size of the table records. This means that Valentina must rebuild the table, so this operation may take a long time.

Example:

```
len = fldString.MaxLength
fldString.MaxLength = 120
```

IndexByWords as Boolean

Using this flag you can specify that a String or a VarChar field should be indexed by words.

Example:

```
fldString.IndexByWords = TRUE
```

Value as String

You should use this property to set or get the value of a String or a VarChar field.

Example:

```
FirstName.Value = “John”
LastName.Value = “Roberts”
```

Class VFixedBinary
Class VVarBinary

Class Description

This type of field is used for storing small binary data in a database. VFixedBinary and VVarBinary classes have the same API, except for their constructors.

Note: The String type of REALbasic is able to correctly handle strings that contain a zero value inside. We mirror this feature of REALbasic in these classes making, a Value of the type String.

Tip: You can use these classes to store text style.

Properties Description

MaxLength as Integer

The maximum length of a FixedBinary and a VarBinary field can be in the range of values 1 .. 65535 bytes.

Example:

```
len = fldBinary.MaxLength
fldBinary.MaxLength = 120
```

Value as String

You should use this property to set/get the value of a FixedBinary or a VarBinary field.

Example:

```
dim str as String

str = "aaa" + chr(0) + "bbb"
fldBinary.Value = str

str = fldBinary.Value
```

Styled Text

To read styled text into a database, your database table must have a field for the text and a separate field for the style info. So, you need these Table properties:

```
storedText as VString
storedStyle as VFixedBinary
```

Note: The field storedText can be VString, VVarChar or VText – whatever is appropriate for your text. For the field storedStyle, you should use VFixedBinary, VVarBinary or VBLOB correspondingly.

In your Table constructor, instantiate these fields:

```
storedText = new Vstring(”storedText_field”,1024)
storedStyle = new VFixedBinary(”storedText_style”,1024)
```

In the example below, ‘myCursor’ is a ‘VCursor’. The text is read and written to an editField called ‘editField1’.

To write the styled text to the database use the following:

```
myCursor.setBlank
myCursor.Field(”storedText_field”).setString(editField1.text)
myCursor.FixedBinaryField(”storedText_style”).value = editField1.textStyleData
myCursor.update
```

To read the styled text from the database cursor back into the editField:

```
dim temp as string
temp = myCursor.FixedBinaryField(”storedText_style”).value
editField1.setTextAndStyle(myCursor.Field(”storedText_field”), temp)
```

Tip: Do not forget that styled text under Windows and MacOS are treated differently, so you need to build code according to the target platform.

Class VBLOB

Properties

DataSize	as Integer (r/o)	
IsCompressed	as Boolean	// TRUE if this BLOB field is compressed.
SegmentSize	as Integer (r/o)	// (in bytes), N * 1024

Constructor

```
VBLOB(
        inName as String,
        inSegmentSize as Integer = 256 )
```

Methods

```
DeleteData()

ReadData as String
WriteData( inData as String )

FromFile( inLocation as Folderitem )
ToFile( inLocation as Folderitem )
```

Class Description

BLOB is a Binary Large OBject. This type of a field is intended for storing large chunks of data, such as graphics, video, text and more.

Constructors of BLOB fields do not have parameter Flags.

Properties Description

DataSize as integer (r/o)

Returns the size in bytes of the value of the current record for this BLOB field.

Example:

```
dim size as Long
size = fldBLOB.DataSize()
```

IsCompressed as Boolean

If TRUE then a BLOB field will compress its data when writing to disk.

Note: The compression method supported by Valentina is described in the Valentina kernel documentation.

Example:

```
fldBlob.IsCompressed = true
```

SegmentSize as Integer (r/o)

Returns the segment size (in bytes) of a BLOB field.

Example:

```
segment = fldBlob.SegmentSize
```

Methods

DeleteData()

Deletes BLOB data of the field.

Note: After this function you must Update() the record of a Table to store a new reference to the BLOB record in the table.

This method is useful if you want to delete BLOB data, but you do not want to delete records.

Example:

```
fldBLOB.DeleteData()
curs.UpdateRecord()
```

ReadData as String

Read value of BLOB and return it as string (note that a REALbasic String can hold binary data).

Example:

```
dim blobValue as String
blobValue = fldBLOB.readData()
```

WriteData(inData as String)

Parameter:	Description:
inData	The binary data to be stored in the BLOB field.

These methods allow you to store in the BLOB field any raw data using REALbasic String.

Example:

```
dim s1 as String
s1 = “aaaaaa”                        // 6 chars
blob_fld.WriteData( s1 )
```

FromFile(inLocation as Folderitem)

Parameter:	**Description:**
inLocation	A location of the file.

Reads the whole file into the BLOB field.

Example:

```
fldBLOB.FromFile( location )
tbl.AddRecord()
```

ToFile(inLocation as Folderitem)

Parameter:	**Description:**
inLocation	A location of the file.

Uploads the value of BLOB field of the current record into a new disk file, specified by parameter inLocation.

Example:

```
fldBLOB.ToFile( location )
```

Class VText

Properties

```
IndexByWords      as Boolean    // TRUE if indexed by each word of the string
Value             as String
```

Constructor

```
VText(
        inName as String,
        inSegmentSize as Integer = 256,
        inFlags as Integer = 0,
        inMethod as String = "")
```

Class Description

This is a special class for storing text which combines the features of VString and VBLOB.

It can be indexed like a VString but has no limit in the size of the content because it is subclass of VBLOB.

String and Text fields can be searched using regular expressions.

Class VPicture

Properties

DefQuality	as Integer	// Default quality for this Picture field.
PictureType	as EVPictureType (r/o)	

Constructor

```
VPicture(
        inName as String,
        inSegmentSize as Integer = 256 )
```

Methods

```
ReadPicture() as Picture
WritePictureAs(
        inPict as Picture,
        inPictType as EVPictureType,
        inQuality as Integer = 50 )
```

Class Description

A Picture field is a special BLOB field which can store pictures in different formats.

Note: By default it converts a Bitmap OS picture into JPEG format.

This field will get and return back a PICT handle on Mac OS and a DIB handle on Windows OS .

Methods

```
WritePictureAs(
        inPict as Picture,
        inPictType as EVPictureType = kJPG,
        inQuality as Integer = 50)
```

Parameter:	**Description:**
inPict	The Picture to be stored.
inPictType	The picture format.
inQuality	Compresion rate, 0..100, default is 50.

Stores a Picture into VPicture field using the specified format.

Parameter Quality can be in the range 0..100 and specify quality of a jpeg compression. The larger the value the better the quality. This parameter can be ignored if the picture format does not require it, e.g. TIFF.

This method expect that Picture is
DIB on Windows.
PICT on MAC.

Note, PICT with JPG compression also is accepted if you specify inPictType as kJPG.

Example:

```
fldPicture.WritePictureAs( inPict, EVPicture.kJPG, 50 )
```

```
ReadPicture() as Picture
```

Reads a picture from the VPicture field and returns it as a Picture to REALbasic. The picture in the database can be in any supported format.

Note, ReadPicture also can show pictures that was added into database using VBLOB. FromFile() method.

Example:

```
dim pict as Picture
pict = fldPicture.ReadPicture()
```

Class VObjectPtr

Properties

OnDeletion	as EVOnDelete	
Target	as VTable	
Value	as Integer	// here is stored the RecID of the target record

Constructor

```
VObjectPtr(
        inName as String,
        inTarget as VTable,
        inOnDeletion as EVOnDelete= kSetNull,
        inFlags as IEVFlag = fNone,
        inLinkName as String = ""  )
```

Method

```
ConvertFromRDB(
        inPrimaryKey as VField,
        inForeignKey as VField)
```

```
AsVLink2() as VLink2
```

The field of the type ObjectPtr is intended to establish a “many to one” relation [M:1] between two Tables by ‘direct pointer’.

Note; In SQL this is called a FOREIGN KEY

It stores references to the related parent record (“One” record). The value of an ObjectPtr field is an unsigned long number (4 bytes, ulong) and it is the physical record number of the parent table. To set the Value of this field you must get the RecID of the record in the TargetTable:

```
mObjectPtr.Value = boPerson.GetRecID
```

Sometimes you may wish to relate a record of Table B to a non-current record of Table A, in this case you can save the RecID to a variable and use it later:

```
dim RecID as Integer
RecID = TableA.GetRecID
TableA.GoToRecord( SomeOtherRecord )
...
TableB.TableA_Ptr.Value = RecID
```

• RecID is 1-based, zero is used for the ID of the undefined record.

The ObjectPtr field must know the pointed object (a parent object) and a deletion control to work correctly.

The Target must be defined when you create the field. There is no reason to change the Target at runtime.

The DeletionControl regulates a record deletion in the “Many” table when a record is deleted in the “One” table. It can be changed at runtime. This is the rule, which defines the behavior on deletion of a record. There are three methods for deleting records.

Leave related Many records:
From the database the record of the parent table is only deleted. The ObjectPtr field of the related child-records is automatically set to 0.

Delete related Many records:
The “One” and “Many “ components are all deleted. If a Many record also has some related Many records in a third Table, then they are also deleted in a cascade delete.

Can not delete if related Many:
The deletion of the One record is not allowed if there is at least one related Many record.

The ObjectPtr field can be used to establish a MANY to ONE relation, but it also can be used to establish a ONE to ONE relation. For this you should specify the ObjectPtr field as unique. Valentina can use this information to optimize a query.

Besides using the ObjectPtr field you can establish a Many to Many relation between two tables. For this you need to create an additional third table - Link as shown on the picture.

Properties Description

OnDelete as Integer

The behavior on deletion of the record-owner.

Example:

```
v = fldPtr.OnDelete
```

Target as VTable

The target table for this ObjectPtr field.

Note: Usually you will read this property. There is not much sense to change the existing target table, because in this case all values of the ObjectPtr field will become zero.

Example:

```
tbl = fldPtr.Target
```

Value as Integer

The Value of the field.

Example:

```
fldPtr.value = tblPerson.RecID
```

Example:

```
tblPerson.RecID = fldPtr.value
```

Constructor

```
VObjectPtr(
        inName as String,
        inTarget as VTable,
        inOnDeletion as  EVOnDelete= kSetNull,
        inFlags as EVFlag = fNone,
        inLinkName as String = ”” )
```

Parameter:	**Description:**
inName	The name of the field.
inTarget	The target table.
inOnDeletion	The behavior on deletion of the record-owner.
inFlags	The flags of the field.
inLinkName	The name of the link.

Constructor of ObjectPtr field.

Note: you will need this if you use the Class method of Valentina to create a database.

Example:

```
        sub tblPhone

                mfPersonPtr = new VObjectPtr(
                        "PersonPtr", tblPerson, EVOnDelete.kSetNull )

        end sub
```

```
ConvertFromRDB(
        inPrimaryKey as VField,
        inForeignKey as VField)
```

Parameter:	**Description:**
inPrimaryKey	The field of the target table that plays role of the PRIMARY KEY field.
inForeignKey	The field of the table of this ObjectPtr field that plays role of the FOREIGN KEY.

Converts a RDB-link between 2 tables into an ObjectPtr-link.

Example:

```
        fldPtr.ConvertFromRDB( fldPersonID, fldPersonPtr )
```

Class VCursor

Properties

```
DataBase            as VDataBase // (r/o) Database of this Cursor.
FieldCount          as Integer   // (r/o) number of selected fields for this Cursor.
Position            as Integer
RecordCount         as Integer   // Number of selected records, it can be reduced.
ReadOnly            as Boolean   // (r/o) TRUE if records can't be changed
                                 // i.e. you can't add/update/delete records.
```

Creation of Cursor

```
VCursor(
        inDatabase as VDatabase,
        inQuery as String,
        inCursorLocation as EVCursorLocation = kClientSide,
        inLocksType as EVLockType = kReadOnly,
        inCursorDirection as EVCursorDirection = kForwardOnly )
```

Field methods

```
Field( InIndex as Integer ) as VField
Field( InName as String ) as VField

BooleanField(inIndex as Integer) as VBoolean
BooleanField(inName as String) as VBoolean

ByteField(inIndex as Integer) as VByte
ByteField(inName as String) as VByte

ShortField(inIndex as Integer) as VShort
ShortField(inName as String) as VShort

UShortField(inIndex as Integer) as VUShort
UShortField(inName as String) as VUShort

MediumField(inIndex as Integer) as VMedium
MediumField(inName as String) as VMedium

UMediumField(inIndex as Integer) as VUMedium
UMediumField(inName as String) as VUMedium

LongField(inIndex as Integer) as VLong
LongField(inName as String) as VLong

ULongField(inIndex as Integer) as VULong
ULongField(inName as String) as VULong

LLongField(inIndex as Integer) as VLLong
LLongField(inName as String) as VLong
```

ULLongField(inIndex as Integer) as VULLong
ULLongField(inName as String) as VULLong

FloatField(inIndex as Integer) as VFloat
FloatField(inName as String) as VFloat

DoubleField(inIndex as Integer) as VDouble
DoubleField(inName as String) as VDouble

DateField(inIndex as Integer) as VDate
DateField(inName as String) as VDate

TimeField(inIndex as Integer) as VTime
TimeField(inName as String) as VTime

DateTimeField(inIndex as Integer) as VDateTime
DateTimeField(inName as String) as VDateTime

StringField(inIndex as Integer) as VString
StringField(inName as String) as VString

VarCharField(inIndex as Integer) as VVarChar
VarCharField(inName as String) as VVarChar

FixedBinaryField(inIndex as Integer) as VFixedBinary
FixedBinaryField(inName as String) as VFixedBinary

VarBinaryField(inIndex as Integer) as VVarBinary
VarBinaryField(inName as String) as VVarBinary

BlobField(inIndex as Integer) as VBlob
BlobField(inName as String) as VBlob

TextField(inIndex as Integer) as VText
TextField(inName as String) as VText

PictureField(inIndex as Integer) as VPicture
PictureField(inName as String) as VPicture

ObjectPtrField(inIndex as Integer) as VObjectPtr
ObjectPtrField(inName as String) as VObjectPtr

Navigation methods

```
FirstRecord() as Boolean
LastRecord() as Boolean
PrevRecord() as Boolean
NextRecord() as Boolean
```

Record methods

```
SetBlank()                     // blank the memory buffer of the record
AddRecord() as Integer         // adds a new record to a cursor

UpdateRecord()                 // updates the current records of the cursor
UpdateAllRecords()             // updates ALL records of a cursor with a new value.

DeleteRecord()                 // deletes the current record of a cursor
DeleteAllRecords()             // deletes all records of a cursor

DropRecord()                   // removes the current record from a cursor
                               // but don’t delete it from the original Table.
```

Import/export methods

```
ImportText(
        inFile as FolderItem,
        inFieldDelimiter as String = chr(09),
        inLineDelimiter as String = LE,
        inEncoding as String = ”UTF-16",
        inHasColumHeader as Boolean = FALSE,
        inMaxRecordsToImport as Integer = 0 )

ExportText(
        inFile as FolderItem,
        inFieldDelimiter as String = chr(09),
        inLineDelimiter as String = LE,
        inEncoding as String =”UTF-16",
        inHasColumHeader as Boolean = FALSE )
```

Conversion methods

```
ToArraySet() as VArraySet
```

Class Description

This class provides the result of the execution of a SQL SELECT statement. Valentina offers a cursor with a random access to the records.

Each cursor has an independent memory buffer, so you can have many cursors at the same time for the same BaseObject, each of which points to different records.

Properties Description

Database as VDataBase

Returns the database of this cursor.

Example:

```
db = fld.Database
```

FieldCount as Integer

Returns the number of fields of this cursor.

Example:

```
fldCount = curs.FieldCount  // get local shortcut to avoid of calling in loop
for i = 1 to fldcount
        ...
next
```

Position as Integer

The current position in the cursor. You can set or get the current position of cursor using this property.

The valid range of values is from 1 to the.

When you assign a new value to the Position, Valentina loads a record from the disk to the memory buffer.

Note: If you try to assign a wrong value then the current record is not changed.

Example:

```
for i = 1 to curs.RecordCount
    curs.Position = i
next
```

RecordCount as Integer

Returns the number of records of cursor.

Example:

```
        recCount = curs.RecordCount // store into a local variable to avoid of calling it
loop
        for i = 1 to fldcount
                ...
        next
```

ReadOnly as Boolean

Returns TRUE if the Cursor is read only, otherwise returns FALSE.

Example:

```
        if( curs.ReadOnly )
                ....
```

Creation of Cursor

```
VCursor(
        inDatabase as VDatabase,
        inQuery as String,
        inCursorLocation as EVCursorLocation = kClientSide,
        inLockType as EVLockType = kReadOnly,
        inCursorDirection as EVCursorDirection = kForwardOnly )
```

Parameter:	**Description:**
inDatabase	The reference to VDataBase object.
inQuery	The query string.
inCursorLocation	The location of the cursor.
inLocksType	The type of record locks.
inCursorDirection	The cursor direction.

This constructor provides you with the second way to create a Cursor . If you want to define a subclass of VCursor than you need to use the constructor of VCursor.

Note: The otherway to create a Cursor is by using the method VDatabase.SQLSelect().

The constructor is given a string as a parameter (as inQuery), resolves it, then returns the resulting table as a cursor of type VCursor.

Note: When finished with a cursor, you must assign it the value nil to destroy it and free memory.

The optional parameters inCursorLocation, inLockType, inCursorDirection allow you to control the behavior of the cursor. See the documentation on Valentina Kernel.and VServer for more details.

You can set the following parameters with these values:

inCursorLocation:	kClientSide = 1,	kServerSide = 2,	kServerSideBulk = 3
inLockType:	kNoLocks = 1,	kReadOnly = 2,	kReadWrite = 3
inCursorDirection:	kForwardOnly = 1, kRandom = 2		

By default these parameters get the following values:
kClientSide, kReadOnly, kForwardOnly

Example:

```
Sub myCursor( inDB as VDataBase, inSQL as String )
        VCursor(inDB, inSQL)          // init parent class.
        ...
end sub
```

This assumes that you want to create the class myCursor which is a subclass of VCursor.

Field Methods

Field(inIndex as Integer) as VField
Field(inName as String) as VField

Parameter:	Description:
inIndex	The Index of the field. Starts from 1.
inName	The Name of the field.

You can use these methods to access fields of the cursor and their values. The order of fields in the cursor is the same as the order of fields in the SELECT statement of the query.

Example:

```
	dim i, Records as Integer
	LastName as String
	dim cur as VCursor

	cur = gDataBase.SQLSelect(“select * from person where name like ‘john’ no_
case”)

	Records = cur.RecordCount
	for i = 1 to Records
		cur.Position = i
		LastName = cur.Field( “last_name” ).GetString
	next
```

Type casting Methods

After you get the field as a VField, you can use type casting to get a reference to the actual class of the field.

As described in the paragaph “VField” you may need to perform type casting:
a) to access a value of the field not as a String but as a number which is about 20 times faster.
b) to access properties of the VField subclasses.

The VCursor class has a set of methods which do this type casting for you.

BooleanField(InIndex as Integer) as VBoolean
BooleanField(InName as String) as VBoolean

ByteField(InIndex as Integer) as VByte
ByteField(InName as String) as VByte

ShortField(inIndex as Integer) as VShort
ShortField(inName as String) as VShort

UShortField(inIndex as Integer) as VUShort
UShortField(inName as String) as VUShort

MediumField(inIndex as Integer) as VMedium
MediumField(inName as String) as VMedium

UMediumField(inIndex as Integer) as VUMedium
UMediumField(inName as String) as VUMedium

LongField(inIndex as Integer) as VLong
LongField(inName as String) as VLong

ULongField(inIndex as Integer) as VULong
ULongField(inName as String) as VULong

LLongField(inIndex as Integer) as VLLong
LLongField(inName as String) as VLLong

ULLongField(inIndex as Integer) as VULLong
ULLongField(inName as String) as VULLong

FloatField(inIndex as Integer) as VFloat
FloatField(inName as String) as VFloat

DoubleField(inIndex as Integer) as VDouble
DoubleField(inName as String) as VDouble

DateField(inIndex as Integer) as VDate
DateField(inName as String) as VDate

TimeField(inIndex as Integer) as VTime
TimeField(inName as String) as VTime

DateTimeField(inIndex as Integer) as VDateTime
DateTimeField(inName as String) as VDateTime

StringField(inIndex as Integer) as VString
StringField(inName as String) as VString

VarCharField(inIndex as Integer) as VVarChar
VarCharField(inName as String) as VVarChar

FixedBinaryField(inIndex as Integer) as VFixedBinary
FixedBinaryField(inName as String) as VFixedBinary

VarBinaryField(inIndex as Integer) as VVarBinary
VarBinaryField(inName as String) as VVarBinary

BlobField(inIndex as Integer) as VBlob
BlobField(inName as String) as VBlob

TextField(inIndex as Integer) as VText
TextField(inName as String) as VText

PictureField(inIndex as Integer) as VPicture
PictureField(inName as String) as VPicture

ObjectPtrField(inIndex as Integer) as VObjectPtr
ObjectPtrField(inName as String) as VObjectPtr

You have several ways to work with fields of a cursor. Lets say you have variables defined as:

```
        dim fld as VField
        dim fldLong as VLong
```

Then you define

```
        fld = curs.Field( “long_fld” )
        VLong( fld ).value = 5
```

Here we get an instance of the VField class from the Cursor. Then, use dynamic type casting to a VLong class.

```
        VLong( curs.Field( “long_fld” ) ).value = 5
```

This is the same operation, but can be written with a single line of code:

```
        curs.LongField(“long_fld”).value = 5
```

Here we ask the cursor to return the field which is already typecasted to type VLong.

Tip: If you need to access cursor fields in a loop, it is much faster to obtain all fields before the loop, then to access them in the loop by reference.

Example:

```
        dim fLong as VLong
        dim VString as VString
        dim curs as VCursor
        dim recCount as Long

        curs = db.SQLSelect( "SELECT Number, str FROM T" )
        fLong = curs.LongField( 1 )
        fString = curs.StringField( 2 )

        recCount = curs.RecordCount
        for i = 1 to recCount
                curs.currentRecord = i
                fLong = i
                fString = str(i)
                curs.Add()
        next
```

Navigation Methods

FirstRecord() as Boolean

Go to the first logical record of a Cursor. Returns TRUE if the first record is found.

Example:

```
res = curs.FirstRecord()
```

LastRecord() as Boolean

Go to the last record of a Cursor. Returns TRUE if the last record is found

Example:

```
res = curs.LastRecord()
```

PrevRecord() as Boolean

Go to the previous record of a Cursor if it exists. Returns TRUE if the previous record is found. Otherwise, it returns FALSE and this means we are at the first logical record in the Cursor or the Cursor is empty.

Example:

```
res = curs.PrevRecord()
```

NextRecord() as Boolean

Go to the next logical record of a Cursor if it exists. Returns TRUE if the next record is found. Otherwise it returns FALSE, which means we are at the last logical record in the Cursor.

Example:

```
if( myCursor.FirstRecord() )
        Do
                // work here
        Loop    Until myCursor.NextRecord() = FALSE
end if
```

You can also do this with the ‘Position property’ in conjunction with ‘RecordCount’, but NextRecord() is more efficient.

Example:

```
if( myCursor.RecordCount > 0 )
        myCursor.Position = 1
        For i = 1 to myCursor.RecordCount  // work here
                myCursor.Position = myCursor.Position + 1
        Next
end if
```

Record Methods

SetBlank(inAccess as EvValueAccess = forUpdate)

Each Cursor has a RAM buffer for field values of the current record. This buffer can be cleared by the SetBlank() method, i.e. all numeric fields become zero, all string fields get the empty string. If a field is Nullable then it will get a NULL value.

Parameter inAccess can be used to speed up SetBlank() if you add records. In this case you can specify its value forAdd, so Valentina will not save copies of previouse field values. In the same time you can always use the default value forUpdate and everyhting will work correctly.

Example:

```
curs.SetBlank( forAdd )
        curs.LongField(1).Value = i
        curs.ShortField(2).Value = i
res = curs.AddRecord()
```

AddRecord() as Integer

Adds a new record to the Cursor with the current field values in the RAM buffer.

Returns RecID of added records.

IMPORTANT: it returns RecID of original table where record was inserted! Valentina can do this because cursor that allows adding of new records always is built on single table.

Example:

```
curs.SetBlank()
        curs.LongField(1).Value = i
        curs.ShortField(2).Value = i
newRecID = curs.AddRecord()
```

UpdateRecord()

Updates the current record of a Cursor with the values in the RAM buffer.

It throws error if a record cannot be updated, e.g. cursor is ReadOnly.

Example:

```
curs.currentRecord = i
    curs.LongField(1).Value = i + 100
    curs.ShortField(2).Value = i + 100
curs.UpdateRecord()
```

UpdateAllRecords()

Updates ALL records of a Cursor with new values. This function can update several fields of the cursor at once. Valentina will only update fields with new values (dirty fields). It is not important what record is current when you, assign new values.

This function is much faster than an iteration of the cursor records in a loop to assign new values.

It throws error if a record cannot be updated, e.g. cursor is ReadOnly.

Example:

```
curs.LongField(1).Value = 145
curs.ShortField(2).Value = 200

curs.UpdateAllRecords()
```

DeleteRecord()

Deletes the current record of a cursor. The next record becomes the current record. Otherwise the previous record becomes current. If a Cursor becomes empty then the current record is undefined.

Returns FALSE if the record cannot be deleted, e.g. it was locked or does not exist, or a cursor is read only.

Example:

```
curs.DeleteRecord()
```

DeleteAllRecords()

Deletes all records of the Cursor. The Cursor becomes empty, the current record becomes undefined.

Returns FALSE if the records cannot be deleted (e.g. cursor is ReadOnly).

Example:

```
curs.DeleteAllRecords()
```

DropRecord()

Removes the current record from a Cursor, but does not delete it from the original Table.

Example:

```
curs.DropRecord()
```

Import/Export Methods

```
ImportText(
        inFile as FolderItem,
        inFieldDelimiter as String = chr(09),
        inLineDelimiter as String = LE,
        inEncoding as String = ”UTF-16",
        inHasColumHeader as Boolean = FALSE,
        inMaxRecordsToImport as Integer = 0 )
```

Parameter:	**Description:**
inFile	File to be imported.
inFieldDelimiter	Character to be used as a field delimeter, default is a tab-chr(0x09).
inLineDelimiter	Character to be used as a record delimeter, default is the OS Line Ending.
inEncoding	Encoding of the imported file.
inHasColumnHeader	TRUE if the import file has a column header line.
inMaxRecordsToImport	The maximum number of records to import.

Imports the specified text file into the fields of the Cursor.

Note: The Cursor must have the flag CanBeUpdated set to TRUE.

The parameters FieldDelimiter and LineDelimiter are optional, i.e. you may specify one of them or both . By default they are TAB (09) and the OS Line Ending correspondingly.

If the cursor represents a subset of the table-fields, then the omitted fields will be filled with NULL values if the field in NULLABLE or blank values otherwise.

Importing text to a Cursor works for a single Table only.

Example:

```
        curs.ImportText( fileToImport, chr(09), chr(13) )
```

```
ExportText(
        inFile as FolderItem,
        inFieldDelimiter as String = chr(09),
        inLineDelimiter as String = LE,
        inEncoding as String =”UTF-16",
        inHasColumHeader as Boolean = FALSE )
```

Parameter:	**Description:**
inFile	The file to be imported.
inFieldDelimiter	The character to be used as a field delimeter, default is tab-chr(0x09).
inLineDelimiter	The character to be used as a record delimeter, default is the OS Line Ending.
inEncoding	Encoding of the imported file.
inHasColumHeader	TRUE if import file has colum header line.

This command exports the fields and records of a Cursor to the designated text file. Using the SELECT statement, you can define the fields to export and their order, as well as the records to be exported.

Example:

```
        curs.ExportText( fileToExport, chr(09), chr(13) )
```

ToArraySet() as VArraySet

This method establish a brige between cursors and sets. You can use this method to obtain an ArraySet that contains RecID values selected by cursor and in the correct order.

Important to note, that this method will work only with cursor built on the single table. You cannot use it for JOIN or GROUP BY results, for example.

TIP. If your target is to build cursor and convert it into set, then it is good idea to SELECT RecID only.

Example:

```
curs = db.SqlSelect( "SELECT RecID FROM T WHERE ..." )

arraySet = curs.ToArraySet()

curs = nil        // we do not need cursor any more.
```

Class VSet

Properties

Count	as Integer (r/o)
IsSortedByRecID	as Boolean
IsEmpty	as Boolean (r/o)

Constructor

Clone() as VSet

Element methods

Append(inValue as Integer)
Remove(inValue as Integer)
Include(inValue as Integer) as Boolean

MakeNewIterator() as VSetIterator
SortByRecID()

Properties Description

Count as Integer (r/o)

The number of items in the Set.

Example:

```
count = set1.Count
```

IsSortedByRecID as Boolean (r/o)

Returns TRUE if the Set is sorted by RecID values.

Example:

```
sorted = set1.isSortByRecID
```

IsEmpty as Boolean (r/o)

Returns TRUE if the Set is empty.

Example:

```
empty = set1.IsEmply
```

Constructor

Clone() as VSet

Clones this Set, i.e. create and return a new set which is of the same type, has the same size and contains the same items.

Example:

```
dim s2 as VSet
s2 = s1.Clone()
```

Element Methods

Append(inValue as Integer)

Parameter:	**Description:**
inValue	A value.

Appends a new value to the Set.

Example:

```
set.Append( rec )
```

Remove(inValue as Integer)

Parameter:	**Description:**
inValue	A value.

Removes the specified value from the Set.

Example:

```
set.Remove( rec )
```

Include(inValue as Integer) as Boolean

Parameter:	**Description:**
inValue	A value.

Returns TRUE if the Set contains the specified value.

Example:

```
found = set.Include( rec )
```

MakeNewIterator() as VSetIterator

Creates and returns a new Iterator for this Set.

Example:

```
iter = s1.MakeNewIterator()
```

SortByRecID()

Sorts the Set.

Example:

```
s1.SortByRecID()
```

Class VArraySet

Constructor

VArraySet(inCount as Integer)
VArraySet(inArraySet as VArraySet)
VArraySet(inBitSet as VBitSet)

Methods

ItemAt(inPosition as Integer) as Integer
ItemAt(inPosition as Integer, Assigns inValue as Integer)

Set operations

Union(inRightSet as VArraySet) as VArraySet
Intersection(inRightSet as VArraySet) as VArraySet
Difference(inRightSet as VArraySet) as VArraySet
SymmetricDifference(inRightSet as VArraySet) as VArraySet

Constructor

VArraySet(inCount as Integer)

Parameter:	**Description:**
inCount	The initial size of ArraySet.

Constructor. Creates an ArraySet with the specified reserved size.

Note: inCount is not the maximum limit. It is just an initial size. If the ArraySet will require more space then it reallocates more RAM automatically.

Example:

```
dim as1
as1 = new VArraySet( 50 )
```

VArraySet(inArraySet as VArraySet)

Parameter:	**Description:**
inArraySet	Another ArraySet

Copy constructor. Creates a new ArraySet from the given inArraySet. The new ArraySet is an exact copy of the inArraySet.

Example:

```
dim as2
as2 = new VArraySet( as1 )
```

VArraySet(inBitSet as VBitSet)

Parameter:	**Description:**
inBitSet	The BitSet.

Constructor. Creates a new ArraySet from the given inBitSet. The ArraySet contains the same items as inBitSet.

Example:

```
dim as3
as3 = new VArraySet( bitSet1 )
```

Methods

ItemAt(inPosition as Integer) as Integer

Parameter:	Description:
inPosition	Position of item in the array set.

Returns the item of the set at the specified position.

Example:

```
recID = as1.ItemAt( 5 )
```

ItemAt(inPosition as Integer, Assigns inValue as Integer)

Parameter:	Description:
inPosition	Position of item in the array set.
inValue	A value.

Assigns inValue to the item of the set at the specified position.

Example:

```
as1.ItemAt( 5 ) = recID
```

Set Operations

Union(inRightSet as VArraySet) as VArraySet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

Executes a union of this set with the inRightSet set. The result becomes this set. Such an operation is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
s1.Union( s2 )
```

Intersection(inRightSet as VArraySet) as VArraySet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

Executes an Intersection of this set with the inRightSet. The result becomes this set. Such an operation is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
s1.Intersection( s2 )
```

Difference(inRightSet as VArraySet) as VArraySet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

Executes the difference of this set with the inRightSet. The result becomes this set. Such an operation is said to be be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
s1.Difference( s2 )
```

SymmetricDifference(inRightSet as VArraySet) as VArraySet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

Executes the SymmetricDifference of this set with the inRightSet. The result becomes this set. Such operation is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
s1.SymmetricDifference( s2 )
```

Class VBitSet

Constructor

VBitSet(inMaxCount as Integer)
VBitSet(inMaxCount as Integer, inArraySet as VArraySet)

Set operations

Union(inRightSet as VBitSet) as VBitSet
Intersection(inRightSet as VBitSet) as VBitSet
Difference(inRightSet as VBitSet) as VBitSet
SymmetricDifference(inRightSet as VBitSet) as VBitSet

Constructor

VBitSet(inMaxCount as Integer)

Parameter:	**Description:**
inMaxCount	The maximum value that can be stored in the bitset.

Constructor. Creates a BitSet of the specified size.

Example:

```
dim bs1
bs1 = new VBitSet( 50 )
```

VBitSet(inMaxCount as Integer, inArraySet as VArraySet)

Parameter:	**Description:**
inMaxCount	The maximal value that can be stored in the bitset.
inArraySet	The ArraySet.

Constructor. Creates a new BitSet from the given inArraySet. The BitSet contains the same items as inArraySet.

Example:

```
dim bs2
bs2 = new VBitSet( as1 )
```

Set Operations

Union(inRightSet as VBitSet) as VBitSet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

Executes a union of this set with the inRightSet set. The result becomes this set. Such an operation is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
s1.Union( s2 )
```

Intersection(inRightSet as VBitSet) as VBitSet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

Executes an Intersection of this set with the inRightSet. The result becomes this set. Such an operation is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
s1.Intersection( s2 )
```

Difference(inRightSet as VBitSet) as VBitSet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

Executes the difference of this set with the inRightSet. The result becomes this set. Such an operation is said to be be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
s1.Difference( s2 )
```

SymmetricDifference(inRightSet as VBitSet) as VBitSet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

Executes the SymmetricDifference of this set with the inRightSet. The result becomes this set. Such operation is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
s1.SymmetricDifference( s2 )
```

Class VSetIterator

Properties

Value	as Integer (r\o)

Methods

FirstItem() as integer
LastItem() as integer
NextItem() as integer
PrevItem() as integer

Properties Description

Value as Integer (r\o)

Returns the current value of the iterator.

Example:

```
v = iter.Value
```

VSetIterator Methods

FirstItem() as integer

Moves the iterator to the first item of the Set.
Returns the value of the item if it is found, else returns 0.

Example:

```
v = iter.FirstItem
```

LastItem() as integer

Moves the iterator to the last item of the Set.
Returns the value of the item if it is found, else returns 0.

Example:

```
v = iter.LastItem
```

NextItem() as integer

Moves the iterator to the next item of the Set.
Returns the value of the item if it is found, else returns 0.

Example:

```
v = iter.NextItem
```

PrevItem() as integer

Moves the iterator to the prev item of the Set.
Returns the value of the item if it is found, else returns 0.

Example:

```
v = iter.PrevItem
```

Class VLink

Properties

BranchCount	as Integer (r/o)
ID	as Integer(r/o)
IsTemporary	as Boolean (r/o)
Name	as String
OnDelete	as EVOnDelete
OnUpdate	as EVOnUpdate
Owner	as VTable

Table methods

```
IsBetween(
        inTableA as VTable,
        inTableB as VTable ) as Boolean

Table( inIndex as integer ) as VTable

Flush( inFlushTables as Boolean = true )
```

Search methods

```
FindLinked(
        inRecID as Integer,
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
        as VArraySet

FindLinkedAsBitSet(
        inSet as VSet,
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
        as VBitSet

FindExclusivelyLinked(
        inRecID as Integer,
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
        as VArraySet

FindAllLinked(
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
        as VBitSet
```

Linking methods

```
CountLinked(
        inRecID as Integer,
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
        as Integer

LinkRecords( inRecID() as Integer )

UnlinkRecords( inRecID() as Integer )

DeleteLinkedRecords(
        inRecID as Integer,
        inTableA as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )

DeleteAllLinkedRecords( inTableA as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )

IsLinked( inLeftRecID as Integer, inRightRecID as Integer ) as Boolean


AsVObjectPtr() as VObjectPtr
AsVBinaryLink() as VBinaryLink
```

Properties Description

BranchCount as Integer (r/o)

Returns the number of branches for this link.

Example:

```
brc = Link.BranchCount
```

ID as Integer (r/o)

Returns the ID of this link. A temporary link has a negative ID.

Example:

```
link_id = Link.ID
```

IsTemporary as Boolean (r/o)

Returns TRUE if this link is temporary.

Example:

```
tmp = Link.IsTemporary
```

Name as String

Returns the name of the link.

Example:

```
s = L ink.Name
```

OnDelete as EVOnDelete

The behavior on deletion of the record-owner.

Example:

```
v = Link.OnDelete
```

OnUpdate as EVOnUpdate

The behavior on update of the record-owner.

Example:

```
v = Link.OnUpdate
```

Owner as VTable

The table which is owner of the link. For symmetric links 1:1 and M:M Valentina cannot define which of tables will be owner of the link. You can use this property to define the owner.

Note, you need specify this property only if you are going to use the DeletionControl for this link.

Example:

```
Link.Owner = tblPerson
```

Table Methods

```
IsBetween(
        inTableA as VTable,
        inTableB as VTable ) as Boolean
```

Parameter:	Description:
inTableA	Left table of link.
inTableB	Right table of link.

Returns TRUE if this Link links both specified Tables.

Example:

```
res = Link.IsBetween( TablA, TablB )
```

```
Table( inIndex as integer ) as VTable
```

Parameter:	Description:
inIndex	The index of table.

Returns a table of the link by index.

Example:

```
tbl = Link.Table( i )
```

```
Flush( inFlushTables as Boolean = true )
```

Parameter:	Description:
inFlushTables	TRUE if Tables of Link also should flush.

Flushes new or modified information of Link. On default it also pass flush() command to Tables of Link. You can set parameter to be FALSE, in this case Tables are not touched.

Example:

```
tbl = Link.Flush()
```

Search Methods

```
FindLinked(
        inRecID as Integer,
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as VArraySet
```

Parameter:	**Description:**
inRecID	The RecID of a record of the left table.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns the records from inTableB linked to record with inRecID from inTableA. If zero records are found then returns NIL.

For a recursive link you should specify the parameter inRecursionDirection. If you specify kFromParentToChild then the function will use child records of the inRecID record. Otherwise it will use parent record(s) of the inRecID record.

Example:

```
        res = Link.FindLinked( rec, TblA, TblB )
```

```
FindLinkedAsBitSet(
        inSet as VSet,
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
        as VBitSet
```

Parameter:	**Description:**
inSet	Selection of records.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns the records from inTableB linked to any record specified by inSet from inTableA. If zero records are found then returns NIL.

For a recursive link you should specify the parameter inRecursionDirection. If you specify kFromParentToChild then the function will use child records of the inRecID record. Otherwise it will use parent record(s) of the inRecID record.

Example:

```
        res = Link.FindLinkedAsBitSet( rec, TblA, TblB )
```

```
FindExclusivelyLinked(
        inRecID as Integer,
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as VArraySet
```

Parameter:	**Description:**
inRecID	The RecID of a record of the left table.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns the records from inTableB linked to the record inRecID of inTableA and only to it. If zero records are found then returns NIL.

For a recursive link you should specify the parameter inRecursionDirection. If you specify kFromParentToChild then the function will use child records of the inRecID record. Otherwise it will use parent record(s) of the inRecID record.

Note: This function returns result different from FindLinked() function only for M : M link.

Example:

```
        res = Link.FindExclusivelyLinked( rec, TblA, TblB )
```

```
FindAllLinked(
        inTableA as VTable,
        inTableB as VTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as VArraySet
```

Parameter:	**Description:**
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns all records of inTableB linked to any record of inTableA.If zero records are found then returns NIL.

Example:

```
        tbl = Link.FindAllLinked( TblA, TblB )
```

Linking Methods

```
CountLinked(
        inRecID as Integer,
        inTableA as VTable,
        inTableB as VTable
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as Integer
```

Parameter:	**Description:**
inRecID	The RecID of a record of the left table.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns the number of records of table inTableB linked to the record inRecID of table inTableA.

For a recursive link you should specify the parameter inRecursionDirection. If you specify kFromParentToChild then the function will use child records of the inRecID record. Otherwise it will use parent record(s) of the inRecID record.

Example:

```
        tbl = Link.CountLinked( rec, TblA, TblB )
```

LinkRecords(inRecID() as Integer)

Parameter:	**Description:**
inRecID	The RecID of a record of the left table.

Establishes a link between records of linked Tables, specified as an array of RecID values (Valentina 2.0 supports 2-branch links only, so 2 records must be specified).

The array must contains the correct number of values, in the order of branches of this link. The order of branches corresponds to the order of Tables on link creation.

Example:

```
dim recs(1) as integer          // allocate array with 2 items.

// Link record 1 of the left table to record 3 of the right table of the Link.
recs(0) = 1
recs(1) = 3
Link.LinkRecords( recs )
```

Example:

```
// The same task in syntax:
Link.LinkRecords( Array(1, 3) )
```

UnlinkRecords(inRecID() as Integer)

Parameter:	**Description:**
inRecID	The RecID of a record of the left table.

Breaks the link between records of the linked Table specified as an array of RecID values.

The array must contain the correct number of values, in the order of branches of this link. The order of branches corresponds to the order of Tables on link creation.

Example:

```
Link.UnlinkRecords( Array(1, 3) )
```

DeleteLinkedRecords(

```
DeleteLinkedRecords(
        inRecID as Integer,
        inTableA as VTable
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
```

Parameter:	Description:
inRecID	The RecID of a record of the left table.
inTableA	Left table of link.
inRecursionDirection	The direction of movement for a recursive link.

Removes all records that are linked by this Link to the record inRecID of table inTableA.

The action of this function depends on the DeletionControl parameter of the link, which can be { refuse, delete some records, update some records }.

ERRORS: errRestrict.

Example:

```
        Link.DeleteLinkedRecords( rec, TblA )
```

DeleteAllLinkedRecords(inTableA as VTable)

Parameter:	Description:
inTableA	Left table of link.

Removes all records linked by this Link to the any record of table inTableA.

The action of this function depends on the DeletionControl parameter of the link, which can be { refuse, delete some records, update some records }.

ERRORS: errRestrict.

Example:

```
        Link.DeleteAllLinkedRecords( TblA )
```

IsLinked(inLeftRecID as Integer, inRightRecID as Integer) as Boolean

Параметр:	Описание:
inLeftRecID	The RecID of a record of the left table.
inRightRecID	The RecID of a record of the right table.

Returns TRUE, if the two specified records are linked.

Example:

```
        res = Link.IsLinked( 3, 2 )
```

Class VLink2

Properties

LeftType	as EVLinkType (r\o)
RightType	as EVLinkType (r\o)

Properties Description

LeftType as EVLinkType (r\o)

Returns the relation type for the left branch. Can be kOne or kMany.

Example:

```
lt = Link.LeftType
```

RightType as EVLinkType (r\o)

Returns the relation type for the right branch. Can be KOne or kMany.

Example:

```
rt = Link.RightType
```

Class VBinaryLink

```
VBinaryLink(
        inName as String,
        inLeftTable as VTable,
        inRightTable as VTable
        inLeftPower as  EVLinkType = kOne,
        inRightPower as EVLinkType = kMany
        inOnDelete as EVStorageType = kDefault
        inStorageType as Boolean = false ) as VLink
```

parameter:	**Description:**
inName	The name of the link.
inLeftTable	Pointer to Left Table.
inRightTable	Pointer to Right Table.
inLeftPower	The link type for the Left Table.
inRightPower	The link type for the Right Table.
inOnDelete	The behavior on deletion of the record-owner
inStorageType	The storage type of the link.

Creates a new Binary Link between 2 tables of this database.

To specify a link you need to define the following:

• A name of the link, unique in the scope of the database.

• Pointers to 2 tables. One table is named Left, the other is named Right.

• The type of link, i.e. if it is 1 : 1 or 1 : M or M : M.

• The behavior of the link on deletion of a record in the Table-Owner.
- In the case of a 1 : M link. The ONE table is the owner table
- In the rest of the cases (1:1 and M:M) the developer can assign any table to be the owner.

• The storage type for the link. Can be Disk-based or RAM-based.
The Binary Link creates files on disk to keep information about linked records. This is why we need to specify the StorageType.

You can specify the same table in the parameters inLeftTable and inRightTable. In this case you get a recursive link.

Example:

```
        linkPersonPhone = VBinaryLink(
                "PersonPhone", tblPerson, tblPhone,
                EVLinkType.kMany, EVLinkType.kMany )
```

Class VConnection

Only for V4RB Client.

Properties

IsConnected	as Boolean	// (r/o) Returns TRUE if connection is available.
HostName	as String	// (r/o) The name/IP of the host where a Valentina Server is located.
UserName	as String	// (r/o) The name of the current user.
Port	as Integer	// (r/o) Returns the port number of the server host.

Method

```
VConnection(
        inHost as String,
        inUserName as String,
        inUserPassword as String,
        inPort as Integer = 15432,
        inTimeOut as Integer = 5,
        inOptions as String = "")
```

Connection Methods

```
Open()
Close()
UseSSL()
```

Properties Description

IsConnected as Boolean (r/o)

Returns TRUE if the connection is available, this method can send a ping-package to server to check this.

Example:

```
res = connection.IsConnected
```

HostName as String (r/o)

Returns a string that contains the name of the Valentina Server host to which this VConnection is connected.

Example:

```
version = connection.HostName
```

Port as Integer (r/o)

Returns the port number of the server host to which this connection is connected to.

Example:

```
port = connection.Port
```

UserName as String (r/o)

Returns user name of this connection.

Note: this is the same name that was used on creation of this Connection.

Example:

```
userName = connection.UserName
```

Creation of VConnection

```
VConnection(
        inHost as String,
        inUserName as String,
        inUserPassword as String,
        inPort as Integer = 15432,
        inTimeOut as Integer = 5,
        inOptions as String = "" )
```

Parameter:	**Description:**
inHost	The IP-address or DNS name of the host.
inUserName	The user name.
inUserPassword	The user password.
inPort	The port number that listens to the Server on inHost. By default it is the standard port of Valentina Server.
inTimeOut	TimeOut in seconds to wait for a Server response.
inOptions	A string of additional options.

This method constructs a VConnection object. This constructor simply stores parameters and does not try connect. The real connection occurs using Open() method.

Example:

```
  dim connection as VConnection = new VConnection( "localhost", "sa", "sa" )

  dim connection as VConnection = new VConnection( "123.456.789.123", "sa", "sa" )
```

Connection Methods

Open()

Establishes a connection to a Valentina Server.

Errors: Wrong user name,
Wrong password,
the user is not an administrator,
connection cannot be established.

Example:

```
dim connection as VConnection
connection = new VConnection( "localhost", "sa", "sa" )
connection.Open()
```

Close()

Closes the connection with the server. After this any objects created in the scope of this connection (VDatabase, VTable, VCursor, ...) becomes invalid and you should not try to use it, otherwise most probably you will get ERR_STREAM_XXXX error.

NOTE: VConnection.Open() and .Close() methods are similar to Init/ShutDown methods in means that you cannot reuse any objects created between these calls in the scope of this connection. Instead on the next Open() you need to create all objects again starting from VDatabase object.

Example:

```
dim connection as VConnection
connection = new VConnection( "localhost", "sa", "sa" )
connection.Open()
...
connection.Close()
```

UseSSL()

You must call this method right BEFORE VConnection.Open() method if you want establish a secure connection to Valentina Server. Note that VServer should listen for SSL port to be able accept such connection.

Example:

```
dim connection as VConnection
connection = new VConnection( "localhost", "sa", "sa" )
connection.UseSSL()
connection.Open()
...
connection.Close()
```

Class VServer

Only for V4RB Client.

Properties

ConnectionCount	as Integer	// (r/o) The number of active connections to a server.
DatabaseCount	as Integer	// (r/o) The number of databases that the server recognizes.
UserCount	as Integer	// (r/o) Returns the count of registered users.
Version	as String	// (r/o) Version of the server.

Method

```
VServer(
        inConnection As VConnection )
```

Connection Methods

```
Restart()
Shutdown()
CancelConnection ( inConnectionID as Integer )
Refresh()
```

INI-File Methods

```
GetVariable( inName as String ) as String
SetVariable( inName as String, inValue as String )
```

Master databases methods

```
RegisterDatabase( inDbName as String, inServerPath as String = "" )
UnregisterDatabase( inDbName as String ) as Boolean
```

User Methods

```
AddUser(
        inUserName as String,
        inPassword as String,
        isAdmin as Integer = FALSE )

RemoveUser(inUserName as String)

ChangeUserPassword(
        inUserName as String,
        inNewPassword as String)

GetUserName( inUserIndex as Integer) as String
GetUserIsAdmin( inUserIndex as Integer) as Boolean
```

DatabaseInfo methods

```
DatabaseInfo( inIndex as Integer) as VDatabaseInfo
```

Class Description

You will only need to use this class in developing the server portion of a Server application. This class allows you to develop your own front end for VServer. It allows to managing parameters of the Server for a user which has administration rights, locally or remotely.

Properties Description

ConnectionCount as Integer (r/o)

Returns the number of all active connections to the server.

Example:

```
connCount = server.ConnectionCount
```

DatabaseCount as Integer (r/o)

Returns the number of databases that a server knows about. In other words, this is the number of databases registered in the Master Database of the VServer.

Example:

```
dbCount = server.DatabaseCount
```

UserCount as Integer (r/o)

Returns the number of registered users.

Example:

```
count = server.UserCount
```

Version as String (r/o)

Returns a string that contains the VServer version number.

Example:

```
version = server.Version
```

Creation of VServer

```
VServer(
        inConnection As VConnection )
```

Parameter:	**Description:**
inConnection	VConnection object.

This method constructs a VServer object. This constructor simply stores parameters and does not try connect. The real connection occurs using Open().

Note: Only Administrator User(s) can use this object.

Example:

```
        dim server as VServer = new VServer( inConnection )
```

Connection Methods

CancelConnection(inConnectionID as Integer)

Parameter	**Description**
inConnectionID	The connection ID.

Cancels an existing connection by its ID.

Example:

```
server.CancelConnection( connID )
```

Restart()

Forces a restart of the VServer.

Example:

```
server.Restart()
```

Refresh()

This method allows you to refresh the list of DatabaseInfo objects. This method sends a request to the Valentina Server.

Example:

```
server.Refresh()
```

Shutdown()

Shuts down the VServer.

Note: After this operation there is no way to restart VServer from the application. If you want to restart the VServer, use Restart().

Example:

```
server.Shutdown()
```

INI-File Methods

```
GetVariable( inName as String ) as String
```

Parameter:	Description:
inName	The name of server variable.

This method allows you to read a value of the specified Server Variable. The name of the variable is case insensitive. With names of variables you can use constants of the INI-file of VServer. For more information, refer to the Valentina Server documentation.

Example:

```
        cache = server.GetVariable( "CacheSize" )
```

```
SetVariable(
        inName as String,
        inNewValue as String )
```

Parameter:	Description:
inName	The name of the server variable.
inNewValue	New value for this variable.

This method allows you to change a value of the specified Server Variable. The name of variable is case insensitive. With names of variables you can use constants of the INI-file of VServer. For more information, refer to the Valentina Server documentation.

NOTE: Some variables require a restart of VServer to affect changes.

Example:

```
        server.SetVariable( "CacheSize",  8 )
```

Master Database Methods

```
RegisterDatabase(
        inDbName as String,
        inServerFullPath as String = "" )
```

Parameter:	Description:
inDbName	The name of the database.
inServerFullPath	The full path of the database located on the server computer.

You can use this method to register in Vserver some existed database. This command adds a new record to the Master Database.

Usually you need just to drop a database into the folder pointed by .ini varable "System-Catalog", and call this method specifying only the name of database. Also it is possible to specify the full path of database on the server computer.

Note: For a MacOS X version of Valentina Server, use a UNIX path.

Errors:

The Database Name already exists.

Example:

```
server.RegisterDatabase( "DbName"  )
```

This assumes that a database with name "DbName" or "DbName.vdb" exists in the "data-bases" folder of VServer.

Example:

```
server.RegisterDatabase( "Accounting", "C:\SomeCompany\account2002.vdb" )
```

```
UnregisterDatabase( inDbName as String ) as Boolean
```

Parameter:	Description:
inDbName	The name of a database.

If you want to remove some database from the scope of the VServer, you need to remove the record about it from the Master Database. You can do this using this method.

Errors:

Database Name not found.

Example:

```
server.UnregisterDatabase( "Accounting" )
```

User Methods

```
AddUser(
        inUserName as String,
        inPassword as String,
        isAdmin as integer = FALSE )
```

Parameter:	**Description:**
inUserName	The user name.
inPassword	The password for this user.
IsAdmin	TRUE if this user has administrator permissions.

An Administrator can add new users to the Master Database.

Errors:

The user name already exists.

Example:

```
server.AddUser( "Peter", "a1234fteg4" )
```

```
RemoveUser( inUserName as String )
```

Parameter:	**Description:**
inUserName	The user name.

An administrator can remove users from the Master Database.

Errors:

The user name is not found.

Example:

```
server.RemoveUser( "Peter" )
```

ChangeUserPassword(
inUserName as String,
inNewPassword as String)

Parameter:	**Description:**
inUserName	The user name.
inNewPassword	New password for this user.

An administrator can change the password of a user.

Errors:

The user name is not found.

Example:

```
server.ChangeUserPassword( "Peter", "rvsa3341" )
```

GetUserName(inUserIndex as Integer) as String

Parameter	**Description**
inUserIndex	The user index.

Returns the name of the user by index.

Example:

```
server.GetUserName()
```

GetUserIsAdmin(inUserIndex as Integer) as Boolean

Parameter:	**Description:**
inUserIndex	The user index.

Returns TRUE if the specified user is an administrator.

Example:

```
res = server.GetUserIsAdmin( i )
```

DatabaseInfo Methods

DatabaseInfo(inIndex as Integer) as VDatabaseInfo

Parameter:	**Description:**
inIndex	1-based index

This method allows you to iterate through the collection of DatabaseInfo objects.

The Vserver instance obtains a list of the DatabaseInfo upon OpenSession(). You can periodically refresh this list using the Refresh() method.

Example:

```
dim dbi as VDatabaseInfo

for i = 1 to server.DatabaseCount
        dbi = server.DatabaseInfo( i )
        ....
next
```

Class VDatabaseInfo

Only for V4RB Client.

Properties

```
ClientCount   as  Integer     // (r/o) The number of connected clients.
CursorCount   as Integer      // (r/o) The number of cursors currently on this database.
Name          as String       // (r/o) The name of the database.
Path          as String       // (r/o) The full path of the database on the server.
```

Methods

```
ClientInfo( inIndex as Integer ) as VClientInfo

Refresh()
```

Methods

ClientInfo(inIndex as Integer) as VClientInfo

Parameter:	**Description:**
inIndex	The index of ClientInfo object.

This method allows you to iterate through the collection of ClientInfo objects.

The object of a DatabaseInfo gets the list of ClientInfo on its creation. You can periodically refresh this list using the Refresh() method.

Example:

```
dim ci as VclientInfo

for i = 1 to dbi.ClientCount
        ci = dbi.DatabaseInfo
        ....
next
```

Refresh()

This method allows you to refresh the list of ClientInfo objects. This method sends a request to the Valentina Server.

Example:

```
dbi.Refresh()
```

Class VClientInfo

Only for a V4RB Client.

Properties

Address	as String	// (r/o) The IP address of the client computer.
ConnectionID	as Integer	// (r/o) The ID of this connection.
CursorCount	as Integer	// (r/o) The number of cursors of this connection.
Login	as String	// (r/o) The login of this connection.
Port	as Integer	// (r/o) The port number of the client computer..

Class VReport

Properties

PageCount	as Integer (r/o)
PaperSize	as EVPaperSize(r/o)
PaperOrientation	as EVPaperOrientation (r/o)
PreviewZoom	as Integer
PreviewWidth	as Integer
PreviewHeight	as Integer

ReportProperty(inName as String) as String
ReportProperty(inName as String, inValue as String)

Construction

VReport(inCursor as VCursor, inReportLocation as Location)

Printing Methods

Print(inPageIndex as Unsigned Integer = 0)
PrintToPDF(inPdfLocation as Location, inPageIndex as Unsigned Integer = 0)

PageCount as Integer (r/o)

Returns the count of pages that will be produced fro this report using the specified Cursor and the current Page format settings.

Example:

```
pages = report.PageCount
```

PaperSize as EVPaperSize (r/o)

Returns the count of pages that will be produced fro this report using the specified Cursor and the current Page format settings.

Example:

```
psize = report.PaperSize
```

PaperOrientation as EVPaperOrientation (r/o)

Returns the count of pages that will be produced fro this report using the specified Cursor and the current Page format settings.

Example:

```
orient = report.PaperOrientation
```

PreviewZoom as Integer

Specifies the preview zoom in percents. Default is 100%. Can be in the range from 1% to 1000%.

Example:

```
report.PreviewZoom = 50
```

PreviewWidth as Integer

Specifies the preview width in pixels.

Example:

```
report.PreviewWidth = 150
```

PreviewHeight as Integer

Specifies the preview height in pixels.

Example:

```
report.PreviewHeight = 450
```

ReportProperty(inName as String) as String

Returns the value of a Report Property specified by its name.

Example:

```
propValue = report.ReportProperty( “ShowPhotos” )
```

ReportProperty(inName as String, inValue as String)

Changes the value of some ReportProperty specified by its name.

Example:

```
report.ReportProperty( “ShowPhotos” ) = “0”
```

Constructor(inCursor As VCursor, inReportLocation as FolderItem)

Parameter:	**Description:**
inCursor	VCursor prepared as you need to contain records to be reported.
inReportLocation	The location of a report file “*.vrp”

Constructs a new instance of VReport class.

You should provide some VCursor, which you must prepare before using e.g. db.SqlSelect() method. This cursor can be result of query as to a local database, so to a remote database managed by a Valentina Server.

Also you need provide disk location of “.vrp” file that contains description of some report. You should design a report with help of Valentina Studio Pro using its Report Editor, then you can export separate report into “.vrp” file.

Example:

```
sub PrintPeopleOfAge( integer inAge )
   dim rp1 as VReport
   dim cursPeople as VCursor

   cursPeople = mdb.SqlSelect(
     “SELECT * FROM Peson WHERE fldAge = “ + Str(inAge)  )

   rp1 = new VReport( cursPeople, GetLocation(“Reports/People.vrp”) )

   ...
end
```

NOTE: You also can use directly Valentina Project File “.vsp”, which contains many reports. You can do this with the help of VReportContainer class

Print(inPageIndex as Unsigned Integer = 0)

Parameter:	**Description:**
inPageIndex	The index of a page to be printed (1..N). Zero to print all records of the report.

Returns the version of the database file format. It can work even with a closed database.

Example:

```
theReport.Print()
```

PrintToPDF(inPdfLocation as Location, inPageIndex as Unsigned Integer = 0)

Parameter:	**Description:**
inLocation	The location for generated PDF file.
inPageIndex	The index of a page to be printed (1..N). Zero to print all records of the report.

Returns the version of the database file format. It can work even with a closed database.

Example:

```
theReport.PrintToPDF( GetLocation( “report_1.pdf” ) )
```

Class VReportContainer

Properties

ReportCount	as Integer (r/o)
ReportName(inIndex as Integer) as String (r/o)	

Construction

VReportContainer(inProjectLocation as FolderItem)

Printing Methods

MakeNewReport(inCursor as VCursor, inIndex as integer) as VReport
MakeNewReport(inCursor as VCursor, inName as string) as VReport

Constructor(inProjectLocation as FolderItem)

Parameter:	**Description:**
inProjectLocation	The location of a Valentina project file “*.vsp”.

Constructs a new instance of VReportContainer class.

You need provide a disk location of “.vsp” file that contains description of one or more Reports.

Example:

```
sub PrintPeopleOfAge( integer inAge )
   dim pAllReports as VReportContainer

   pAllReports = new VReportContainer( GetFolderItem( “Reports/MyProject.vsp” ) )

   // Now you can use methods of VReportContainer class to:
   //   * investigate how many reports are inside of this container.
   //   * get their names to display in e.g. menu
   //   * extract single reports creating VReport class instance.
end
```

ReportCount as Integer (r/o)

Returns the count of reports inside of this container.

Example:

```
dim my_reports as VReportContainer

for i = 1 to my_reports.ReportCount
    ...
end if
```

ReportName(inIndex as Integer) as String (r/o)

Returns the name of Nth reports from his container. This name can be used for example to show list of all reports in the container.

Example:

```
dim my_reports as VReportContainer
dim repName as String

for i = 1 to my_reports.ReportCount
    repName = my_reports.ReportName( i )
end if
```