VALENTINA 5

for Revolution Reference

Paradigma Software Inc.
www.paradigmasoft.com

Contents

Conventions

Classes

Valentina is Object-Relational database. It is self made in the C++ way and it offers OO API to access database. Revolution is not the Object-Oriented language, so we use emulation of classes. To emulate a class, we just use prefix with its name for each function which belong to this class. Note that the first parameter is object of this class. For example:

```
VDatabase_Create( dbRef, ... )
```

Revolution developers can think about classes as about group of functions. Each group has its own prefix for functions.

Object References

When you create some object of Valentina you get its reference. In Revolution reference this is just integer type. You should be careful in use of references.

To simplify development V4REV executes check of references that you send to functions as parameters if Valentina_DebugLevel has value other than "kLogNothing". In case of bad reference, warning about this will be logged and action of function will be refused. For release of your application you will set DebugLevel to kLogNothing, so checks will be disabled and this will not slow down your release builds.

Constants

Many functions require as parameters values some specific set of constants. In many programming languages exists for this pupose enumerated types. Revolution does not have enumerated types, so we will use just strings.

For example:

```
VDatabase_CreateTable( "tblPerson", "kRam" )
```

If you make mistake in the name of a constant then Valentina will return error about this.

Default parameters

Many functions of Valentina has optional parameters with default values. This means that you can skip this parameter during call of the function and the default value will be used. In this documentation such parameters are marked by [] or having default value after '='.

For example:

```
func( param1, [param2] )
func( param1, param2 = false )
```

Error Handling

Valentina engine itself is a C++ library, and uses exceptions in case of errors. Valentina for Livecode external catches these exceptions and store error code and error message into own internal variables, which you can acces with help of:

Valentina_ErrString(), Valentina_ErrNumber()
VDatabase_ErrString( ), VDatabase_ErrNumber()

Note, that Valentina and VDatabase versions do absolutly the same - return values of global variables.

Since error state is stored into global variable of external, you can check error right after call of some function. Call of another function will cleanup state of global variables.

Becides, V4REV external in case of error, put into result string with this error started by "ERROR: " prefix. Some Livecode developers prefer to check this prefix as we know.

This is code snip from V4REV/Examples/SQL_way/CreateDRop_Table

```
function ShowError
  put Valentina_ErrNumber() into errCode
  put Valentina_ErrString()    into errString

  put errCode  into field ErrCodeField
  put errString into field ErrStringField

end ShowError
```

Each external function can return error, especially if you work with a remote Valentina Server. Usually you self define when to check errors. But you should always check error after SQL command.

Valentina Enumeration Types (Enums)

Valentina for Revolution in the documentation uses term of Enumerated Types. Revolution do not support enums, so this is just a convention to be similar with other languages that Valentina supports.

Each Valentina's enumeration type starts with the prefix "EV".

EVColAttribute
- kFrenchCollation
- kAlternateHandling
- kCaseFirst
- kCaseLevel
- kNormalizationMode
- kStrength
- kHiraganaQuaternaryMode
- kNumericCollation
- kAttributeCount

EVColAttributeValue
- kDefault

- kPrimary
- kSecondary
- kTertiary
- kDefaultStrength
- kQuaternary
- kIdentical

- kOFF
- kON

- kShifted
- kNonIgnorable

- kLowerFirst
- kUpperFirst

EVCursorDirection
- kForwardOnly
- kRandom

EVCursorLocation
- kClientSide
- kServerSide

EVDateFormat
- kMDY
- kDMY
- kYMD

EVDbMode
- kDscDatBlbInd
- kDsc_DatBlbInd
- kDsc_DatBlb_Ind
- kDsc_Dat_Blb_Ind
- kDscDatBlb_Ind
- kDscDat_Blb_Ind
- kDscDatInd_Blb
- kDsc_DatInd_Blb

EVDebugLevel
- kLogNothing
- kLogErrors
- kLogFunctions
- kLogParams

EVDumpType
- kSQL
- kXML

EVDumpData
- kStructureOnly
- kStructureAndRecords
- kRecordsOnly

EVFieldType

- kTypeEmpty
- kTypeEnum
- kTypeBoolean
- kTypeByte
- kTypeShort
- kTypeUShort
- kTypeMedium
- kTypeUMedium
- kTypeLong
- kTypeULong
- kTypeLLong
- kTypeULLong

- kTypeFloat
- kTypeDouble
- kTypeLDouble
- kTypeDecimal

- kTypeDate
- kTypeTime
- kTypeDateTime

- kTypeString
- kTypeVarChar

- kTypeFixedBinary
- kTypeVarBinary

- kTypeBLOB
- kTypeText
- kTypePicture
- kTypeSound
- kTypeMovie

- kTypeRecID
- kTypeOID

- kTypeObjectPtr
- kTypeObjectsPtr

- kTypeTimeStamp

EVFlag

- fNone
- fNullable
- fIndexed
- fUnique
- fIndexByWords
- fCompressed
- fMethod

EVLinkType
- kMany
- kOne

EVLockType
- kNoLocks
- kReadOnly
- kReadWrite

EVOnDelete
- kNoAction
- kSetNull
- kCascade
- kRestrict
- kDefault

EVOs
- kOsDefault
- kOsMac
- kOsWindows
- kOsUnix

EVOnUpdate
- kNoAction
- kSetNull
- kCascade
- kRestrict
- kDefault

EVPictType
- kUnknown
- kMacPict
- kWinDIB
- kJPG
- kTIFF

EVQueryType
- kCursor
- kBoolean
- kULong
- kString

EVQueryFlag
- kNone
- kSchemaWasModified
- kDatabaseWasChanged
- kXMLResult

EVRecursionDirection
- kFromParentToChild
- kFromChildToParent

EVStorageType
- kDefault
- kDisk
- kRAM

EVSearch
- kPreferIndexed
- kPreferNonIndexed

EVTableKind
- kTblPermanent
- kTblTemporary

EVVerboseLevel
- kNone
- kLow
- kNormal
- kHigh
- kVeryHigh

EVValueAccess
- forAdd
- forUpdate

Valentina Class

Properties

Valentina_CacheSize() (r/o)
Valentina_DatabaseCount() (r/o)
Valentina_Database(inIntIndex) (r/o)
Valentina_DebugLevel([inLevel]) (r/w)
Valentina_FlushEachLog([inOnOff]) (r/w)
Valentina_LocalConnection() (r/o)
Valentina_Version() (r/o)

Initialization Functions

Valentina_Init(inCacheSize, inMacSN = "" , inWinSN = "" , inLinSN = "")
Valentina_InitClient(inCacheSize)
Valentina_InitReports(inMacSN = "" , inWinSN = "" , inLinSN = "")

Valentina_ShutDown()
Valentina_ShutDownClient()

Valentina_Convert_1_2(inOldDb, inNewDb, inBoolLoadRecords, inNewSegmentSize = 0)

Utility Functions

Valentina_SetExtensions(inStrDesc, inStrDat, inStrBlb, inStrInd)

Valentina_EscapeString(inStr)

Valentina_GetDatabaseVersion(inVdbFile)
Valentina_GetCurrentVersion( )

Valentina_GetSchemaVersion(inVdbFile)
Valentina_GetDatabaseMode(inVdbFile)
Valentina_GetIsStructureEncrypted(inVdbFile)

Error Check Functions

Valentina_ErrNumber()
Valentina_ErrString()

Properties

Valentina_CacheSize() (r/o)

Returns: integer

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

Example:

```
put Valentina_CacheSize() into var
```

Valentina_DatabaseCount() (r/o)

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:

```
get Valentina_DatabaseCount
```

Valentina_Database(inIntIndex) (r/o)

Returns: integer

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

See also:

Valentina_DatabaseCount()

Example:

```
get Valentina_Database( inIntIndex ) into dbRef
```

Valentina_DebugLevel([inLevel]) (r/w)

Returns: integer

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

Any debug level above 0 will create a file which outputs the results. The file will be named “V4REV_Log_ddddddddd.txt”. It will be created in the "vlogs" subdirectory of the current executable directory (your app or Livecode IDE itself)

The valid values are:

kLogNothing - no debug messages.
kLogErrors - log a message only when an error occurs.
kLogFunctions - log every function.
kLogParams - log every function and its parameters.

Example:

```
get Valentina_DebugLevel( "kLogParams" )
```

Example:

```
put Valentina_DebugLevel() into var
```

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

Valentina_FlushEachLog([inOnOff]) (r/w)

Returns: 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:

```
get Valentina_FlushEachLog(true)
        -- some debugged code
get Valentina_FlushEachLog(false)

put Valentina_FlushEachLog into BOOLVAR
```

VConnection localConnection (r/o)

Returns: VConnection

Returns the VConnection object for local databases. This allows you work with local databases in the way similar to remote databases. In particular you get access to VConnection SqlQuery(), SqlSelect(), SqlExecute() methods that do SQL query without VDatabase object.

See also:

VConnection.SqlQuery()
VConnection.SqlSelect()
VConnection.SqlExecute()

Example:

```
Valentina.localConnnection.SqlQuery( "SHOW DATABASES" )
```

Valentina_Version() (r/o)

Returns: string

Returns the version of the Valentina engine.

Example:

```
put Valentina_Version() into var
```

Initialization Functions

```
Valentina_Init(
        inCacheSize = 10 * 1024 *1024,
        inMacSN = "",
        inWinSN = "",
        inLinSN = "" )
```

Parameter:	**Description:**
inCacheSize	The size of the cache in bytes.
inMacSN	The serial for Mac OS.
inWinSN	The serial for Windows.
inLinSN	The serial for Linux.

To improve disk access, Valentina uses a cache mechanism. Using the Valentina.Init() Function, 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 no more than 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 Functions to ensure that you do not expose your serial numbers in your built applications.

Example:

```
get Valentina_Init( 5 * 1024 * 1024 )    --   demo
```

```
Valentina_InitClient(
        inCacheSize as Integer = 10 * 1024 * 1024 )
```

Parameter:	**Description:**
inCacheSize	The size of the cache in bytes.

Initializes the Valentina Client for work.

* VClient uses this cache only for server-side cursors.
* Each server-side cursor keeps the list of cached records. When cursors dies it destroy cache buffers also.
* Also exists list of usage history. If cache limit reached then older buffers are released.

Example:

```
get Valentina_InitClient()
```

InitReports(
string inMacSN = "",
string inWinSN = "",
string inLinSN = "")

Parameter	Description
inMacSN	The serial number for use under Mac OS or "" in the demo mode.
inWinSN	The serial number for use under Windows or "" in the demo mode.
inLinSN	The serial number for use under Windows or "" in the demo mode.

Returns: VOID

Description:

Initializes the work with Valentina reports for your application.

If you not specify serials for Valentina Reports then generated reports will have DEMO watermark.

Example:

```
Valentina_Init( 9 * 1024 * 1024, "",  "", "" )
Valentina_InitReports( "",  "", "" )
```

Valentina_ShutDown()

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

Example:

```
get Valentina_Init( 5 * 1024 * 1024 )
        .....--   some work here
get Valentina_ShutDown()
```

Valentina_ShutDownClient()

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

Example:

```
get Valentina_ShutDownClient()
```

```
Valentina_Convert_1_2(
        inOldDb,
        inNewDb,
        inBoolLoadRecords,
        inNewSegmentSize = 0 )
```

Parameter:	**Description:**
inOldDb	location of database in 1.x format.
inNewDb	Location for new database of 2.0 format.
inBoolLoadRecords	If TRUE then records are copied to new database.
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 Function.

Note: This function do not change the old Database.

Example:

```
get Valentina_Convert_1_2( oldDB, newDB, true )
```

Utility Functions

```
Valentina_SetExtensions(
        inStrDesc,
        inStrDat,
        inStrBlb,
        inStrInd)
```

Parameter:	**Description:**
inStrDesc	Extension for description file (.vdb)
inStrDat	Extension for data file (.dat)
inStrBlb	Extension for BLOB file (.blb)
inStrInd	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 Function, then the standard four extensions are used by default. If you do use this Function, 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):

```
get Valentina_SetExtensions( "vdb", "dat", "blb", "ind" )

get Valentina_SetExtensions( "vdb", "", "", "tre" )
```

Valentina_EscapeString(inStr)

Parameter:	**Description:**
inStr	The string to be escaped.

Returns: string

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.

Example(s):

```
put Valentina_EscapeString( "Valentina's (day)" ) into var
```

Valentina_GetDatabaseFormatVersion(inVdbFile)

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

Returns: integer

Returns the version of database file format. Works on a closed database.

Example:

```
put Valentina_GetDatabaseFormatVersion( path ) into var
```

Valentina_GetCurrentFormatVersion()

Returns: integer

Returns the current format version of database file. Works on a closed database

Example:

```
put Valentina_GetCurrentFormatVersion() into var
```

Valentina_GetSchemaVersion(inVdbFile)

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

Returns: integer

Returns the version of database schema. Works on a closed database.

Example:

```
put Valentina_GetSchemaVersion( path )
```

Valentina_GetDatabaseMode(inVdbFile)

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

Returns: integer

Returns the database mode. Works on a closed database.

Example:

```
put Valentina_GetDatabaseMode( path )
```

Valentina_GetIsStructureEncrypted(inVdbFile)

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

Returns: integer

Returns TRUE if database structure is encrypted. Works on a closed database.

Example:

```
put Valentina_GetIsStructureEncrypted( path )
```

VConnection Class

Properties

VConnection_IsConnected (r/o)
VConnection_HostName (r/o)
VConnection_Port (r/o)
VConnection_UserName (r/o)

Construction Methods

```
VConnection_Constructor(
        inHostName,
        inUserName,
        inUserPassw,
        inPort=15432,
        inTimeOut=5,
        inOptions="" )
```

Methods

VConnection_Open()
VConnection_Close()

VConnection_UseSSL()

SQL Methods

```
VConnection_SqlQuery(
        inQuery,
        inCursorLocation  = kClientSide,
        inLockType = kReadOnly,
        inCursorDirection = kRandom,
        inArrayNameOfBinds = "" )

VConnection_SqlSelect(
        inQuery,
        inCursorLocation  = kClientSide,
        inLockType = kReadOnly,
        inCursorDirection = kRandom,
        inArrayNameOfBinds = "" )

VConnection_SqlExecute(
        inQuery,
        inArrayNameOfBinds = "" )
```

Properties

VConnection_IsConnected(conRef) (r/o)

Returns TRUE if the connection is available, this method can send a ping-package to server to check this.

Example:

```
put VConnection_IsConnected( connRef ) into res
```

VConnection_HostName(conRef) (r/o)

Returns a string that contains the name of the Valentina Server host to which this VConnection is connected.

Example:

```
put VConnection_HostName( connRef ) into hName
```

VConnection_Port(conRef) (r/o)

Returns the port number of the server host to which this connection is connected to.

Example:

```
put VConnection_Port( connRef ) into port
```

VConnection_UserName(conRef) (r/o)

Returns user name of this connection.

Note: this is the same name that was used on creation of this Connection.

Example:

```
put VConnection_UserName( connRef ) into userName
```

Construction Functions

```
VConnection_Construction(
        string   inHostName,
        string   inUserName,
        string   inUserPassw,
        integer inPort=15432,
        integer inTimeOut=5,
        string   inOptions="" )
```

Returns: VOID

inHostName	The IP-assress or DNS name of a host
inUserName	The user name
inUserPassw	The password of user
inPort	The port number of Valentina Server. By default is used the standard port of Valentina Server
inTimeOut	Timeout in seconds to wait for the Server response
inOptions	The string of additional options

Description:

This function constructs a VConnection object. This constructor simply stores parameters and does not try connect. The real connection occurs using the Open() function.

inTimeOut parameter specify how much seconds we want wait while DNS will find required server. If during this N seconds server is not reached by TCP/IP protocol then we get “stream error” message.

[NEW in v4.0] If the required server is found and you have not unlimited license then into game may come "pool of connections" of VServer. Let license of a VServer allows K simultaneous connections. Let you try to establish one more K+1 connection. License do not allow this. But Valentina Server will not refuse your request immediately. It will pool this connection request to wait MaxConnectionTimeout seconds in hope that some existed connection will be release. If this happens then VServer opens connection for you. If during MaxConnectionTimeout any connection was not released, then you will get error message that connection was not established (in this case you need try it again).

Example:

```
put VConnection_Constructor( "localhost", "sa", "sa" ) into connRef
VConnection_Open( connRef )
```

Example:

```
put VConnection_Constructor( "123.456.789.123", "sa", "sa" ) into connRef
```

Connection Functions

VConnection_Open(connRef)

Returns: VOID

Establishes a connection to a Valentina Server.

Errors:

* Wrong user name,
* Wrong password,
* The user is not an administrator,
* Connection cannot be established.

Example:

```
put VConnection_Constructor( "localhost", "sa", "sa" ) into connRef
VConnection_Open( connRef )
```

VConnection_Close(connRef)

Returns: VOID

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() Functions are similar to Init/ShutDown Functions 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:

```
put VConnection_Constructor( "localhost", "sa", "sa" ) into connRef
VConnection_Open( connRef )
...
VConnection_Close( connRef )
```

VConnection_UseSSL(connRef)

Returns: VOID

You must call this function right BEFORE VConnection.Open() function 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:

```
put VConnection_Constructor( "localhost", "sa", "sa" ) into connRef
VConnection_UseSSL( connRef )
VConnection_Open( connRef )
...
VConnection_Close( connRef )
```

SQL Functions

The following three Functions of VConnection class are very similar to Functions of VDatabase class except that they do not have the first inDatabase parameter.

These Functions can send SQL commands that are not related to a single database, or are not related to database at all. For example "SHOW DATABASES", "DROP USER".

If the command should be sent to a single database, then such database should be set active with help of command "SET DATABASE db_name". Or you can just use Functions of VDatabase class.

Since these Functions by syntax and usage are 100% the same as VDatabase class Functions, we just refer you that Functions.

```
VConnection_sqlQuery(
        inQuery,
        inCursorLocation = "kClientSide",
        inLockType = "kReadOnly",
        inCursorDirection = "kRandom",
        inArrayNameOfBinds = "" )
```

See description of VDatabase_SqlQuery() function.

```
VConnection_sqlSelect(
        inQuery,
        inCursorLocation = "kClientSide",
        nLockType = "kReadOnly",
        nCursorDirection = "kRandom",
        inArrayNameOfBinds = "" )
```

See description of VDatabase_SqlSelect() function.

```
VConnection_sqlExecute(
        inQuery,
        inArrayNameOfBinds = "" )
```

See description of VDatabase_SqlExecute() function.

VQueryResult Class

Properties

VQueryResult_Type(ResultRef) (r/o) -- Returns the type of contents in this QueryResult.
VQueryResult_Flags(ResultRef) (r/o) -- Returns additional flags about SQL command.

VQueryResult_Cursor(ResultRef) (r/o) -- Gives you cursorRef if this QueryResult contains it.
VQueryResult_ULong(ResultRef) (r/o) -- Gives you ULONG value if this QueryResult contains it.

Construction

VQueryResult_Destructor(ResultRef) -- Destroys object of QueryResult

Description

Historically, Valentina API did have:
a) SqlSelect() to execute SQL commands that return VCursor back. For example, SELECT or "SHOW ..." commands.
b) SqlExecute() to execute all other commands of Valentina SQL. It returns ULONG integer - the number of affected records.

With time we have come to need, for example, in Valentina Studio's SQL Editor, to have single function, which is able execute any Valentina SQL commands.

Answer was in a new function SqlQuery() on each level:

database scope:	VDatabase.SqlQuery()
connection scope:	VConnection.SqlQuery()
global scope:	Valentina.SqlQuery()

This new function must be able return different results: cursor or ULONG.

We resolve this with the help of small class VQueryResult

When you have got a QueryResult you need, first of all, to define the type of its contents. Then you can extract contents.

When QueryResult is not needed, you should destroy this object using VQueryResult_Destructor().

Properties

VQueryResult_Type(ResultRef) (r/o)

Description:

Returns the type of the result of a Valentina SQL command.

Example:

```
put VDatabase_SqlQuery( dbRef, SomeSqlCommand ) into res
if VQueryResult_Type(res) = "kCursor" then
   put VQueryResult_AsCursor( res ) into curs
end if
```

VQueryResult_Flags(ResultRef) (r/o)

Description:

Can return additional information about operation.

VQueryResult_Cursor(ResultRef) (r/o)

Description:

Extracts Cursor from the result. Note, that if result type is not cursor, then this property will be nil.

Example:

```
put VDatabase_SqlQuery( dbRef, SomeSqlCommand ) into res
if VQueryResult_Type(res) = "kCursor" then
  put VQueryResult_Cursor( res ) into cursRef
end if
```

VQueryResult_ULong(ResultRef) (r/o)

Description:

Extracts ULONG value from result. Usually you get this kind of result from non-SELECT commands, such as INSERT, UPDATE, DELETE. This value usually means the number of affected records

Example:

```
put VDatabase_SqlQuery( dbRef, SomeSqlCommand ) into res
if VQueryResult_Type(res) = "kULong" then
  put VQueryResult_ULong( res ) into affectedRows
end if
```

VDatabase Class

Properties

```
VDataBase_CenturyBound( dbRef, [inIntValue] )                -- default 20.
VDataBase_Creator( dbRef, [inStrValue] )                     -- Mac creator signature
VDataBase_CollationAttribute( dbRef, inEnumColAttribute, [inEnumColAttributeValue] )

VDataBase_DateFormat( dbRef,[ inEnumDateFormat] )            -- specifies the format of date.
VDataBase_DateSep( dbRef, [inStrValue] )                     -- separator for date, e.g. '/'

VDataBase_ErrNumber( dbRef ) (r/o)                           -- The last error number, 0 if OK.
VDataBase_ErrString( dbRef ) (r/o)                           -- String description of error.

VDataBase_IOEncoding( dbRef, [inStrValue] )

VDataBase_IndexCount( dbRef ) (r/o)
VDataBase_IsReadOnly( dbRef ) (r/o)
VDataBase_IsOpen( dbRef ) (r/o)
VDtabase_IsRemote( dbRef ) (r/o)

VDatabase_LastInsertedRecID( dbRef ) (r/o)
VDataBase_LinkCount( dbRef ) (r/o)
VDataBase_Locale( dbRef, [inStrLocale] )
VDataBase_Mode( dbRef ) (r/o)
VDataBase_Name( dbRef )   (r/o)
VDataBase_Path( dbRef ) (r/o)
VDataBase_SchemaVersion( dbRef, [inIntValue] )               -- Version of db Schema
VDataBase_StorageEncoding( dbRef, [inStrValue] )
VDataBase_TableCount( dbRef ) (r/o)
VDataBase_TimeSep( dbRef, [inStrValue] )                     -- separator for time, e.g. ':'
VDataBase_SegmentSize( dbRef ) (r/o)

VDataBase_IsEncrypted( dbRef ) (r/o)                         -- TRUE if the database is encrypted.
VDataBase_IsStructureEncrypted( dbRef ) (r/o)                -- TRUE if the database structure is encrypted.

--   for CLIENT only:
VDataBase_ResponseTimeOut( dbRef, [inStrValue] )             -- default 60 seconds.
```

Construction Functions

```
VDataBase_Constructor( inEnumStorageType = "kDefault"  )
VDataBase_Constructor( inConnection  )
VDataBase_Constructor_FromRevDB( inDB_ID )


VDatabase_Destructor( dbRef )
```

Disk Functions

```
VDataBase_Create(
        dbRef,
        inStrPath,
        inEnumMode = "kDsc_Dat_Blb_Ind",
        inIntSegmentSize = 32768,
        inEnumOS = "kDefault" )

VDataBase_Flush( dbRef )
VDataBase_Open( dbRef, inStrPath )
VDataBase_Close( dbRef )
VDataBase_ThrowOut( dbRef )

VDatabase_Clone( inTarget, inLoadRecords = true, inDoLog = false )

VDatabase_SetMacTypes( dbRef, inStrDescType, inStrDatType, inStrBlbType, inStrIndType )
```

Schema Functions

```
VDataBase_CreateTable( dbRef,
        inStrName,
        inEnumTableKind = "kTblPermanent",
        inEnumStorageType = "kDefault" )

VDataBase_DropTable( dbRef, tblRef )

VDataBase_CreateBinaryLink( dbRef,
        inStrLinkName,
        inLeftTblRef,
        inRightTblRef,
        inEnumLinkTypeLeft = "kOne",
        inEnumLinkTypeRight = "kMany,
        inEnumOnDelete = "kSetNull",
        inEnumStorageType = "kDefault"
        inBoolTemporary = false )

VDataBase_CreateForeignKeyLink( dbRef,
        inStrLinkName,
        inKeyFieldRef,
        inPtrFieldRef,
        inEnumOnDelete = "kSetNull",
        inEnumOnUpdate = "kCascade",
        inBoolTemporary = false )

VDataBase_DropLink( dbRef, lnkRef )
```

Table Functions

VDataBase_Table(dbRef, inIntIndexOrStrName)

Link Functions

VDataBase_Link(dbRef, inIntIndexOrStrName)

SQL Functions

```
VDataBase_SqlExecute(
        dbRef,
        inStrCommand,
        inArrayNameOfBinds = nil )

VDataBase_SqlSelect(
        dbRef,
        inStrQuery,
        inEnumCursorLocation = "kClientSide",
        inEnumLockType = "kReadOnly",
        inEnumDirection = "kForwardOnly",
        inArrayNameOfBinds = "" )

VDataBase_SqlQuery(
        dbRef,
        inStrQuery,
        inEnumCursorLocation = "kClientSide",
        inEnumLockType = "kReadOnly",
        inEnumDirection = "kForwardOnly",
        inArrayNameOfBinds = "" )

VDatabase_SqlSelectRecords(
        dbRef,
        inStrQuery,
        inEnumCursorLocation = "kClientSide",
        inEnumLockType = "kReadOnly",
        inArrayNameOfBinds = "",
        inFromRec = 1,
        inMaxRecords = -1,
        inFldDelim = "\t",
        inRecDelim = '\n',
        inFldPrefix = "",
        inFldSuffix = "",
        inRecPrefix = "",
        inRecSuffix = "" )
```

IndexStyle Functions

```
VDataBase_CreateIndexStyle( dbRef, inStrName )
VDataBase_DropIndexStyle( dbRef, indexStyleRef )
VDataBase_IndexStyle( dbRef, inStrName )
```

Encryption Function

```
VDataBase_Encrypt( dbRef, inStrKey, inEnumDataKind = "kRecordsOnly" )
VDataBase_Decrypt( dbRef, inStrKey, inEnumDataKind = "kRecordsOnly" )
VDataBase_ChangeEncryptionKey( dbRef, inStrOldKey, inStrNewKey, inEnumDataKind = "kRecordsOnly" )

VDataBase_RequiresEncryptionKey( dbRef, inEnumDataKind = "kRecordsOnly" )
VDataBase_UseEncryptionKey( dbRef, inStrKey, inEnumDataKind = "kRecordsOnly" )
```

Dump Functions

```
VDataBase_Dump(
        dbRef,
        inStrDumpPath,
        inEnumDumpType,
        inEnumDumpData = "kStructureAndRecords",
        inBoolFormatDump = false,
        inEncoding = "UTF-16" )

VDataBase_LoadDump(
        dbRef,
        inStrDumpPath,
        inStrNewDBPath,
        inEnumDumpType,
        inEncoding = "UTF-16" )
```

Description

* These Class manage 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

VDataBase_CenturyBound(dbRef, [inIntValue])

Returns: 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:

```
get VDataBase_CenturyBound(dbRef,30)

put VDataBase_CenturyBound() into var
```

VDataBase_Creator(dbRef, [inStrValue])

Returns: 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:

```
get VDataBase_Creator(dbRef,"MYDB")

put VDataBase_Creator(dbRef) into var
```

VDataBase_CollationAttribute(dbRef, inEnumColAttribute, [inEnumColAttributeValue])

Returns: EVColAttributeValue

The value of the specified collation attribute for this database.

Example:

```
get VDataBase_CollationAttribute( dbRef, "kStrength", "kPrimary" )

put VDataBase_CollationAttribute(dbRef,"kStrength") into var
```

VDataBase_DateFormat(dbRef, [inEnumDateFormat])

Returns: 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:

```
get VDataBase_DateFormat(dbRef,kYMD)

put VDataBase_DateFormat(dbRef) into var
```

VDataBase_DateSep(dbRef, [inStrValue])

Returns: string

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

Example:

```
get VDataBase_DateSep(dbRef," / ")

put VDataBase_DateSep(dbRef) into var
```

VDataBase_ErrNumber(dbRef) (r/o)

Returns: integer

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:

```
put VDataBase_ErrNumber(dbRef) into var
```

VDataBase_ErrString(dbRef) (r/o)

Returns: string

Returns the string that describes the last error.

Example:

```
put VDataBase_ErrString(dbRef) into var
```

VDataBase_IOEncoding(dbRef, [inStrValue]) (r/w)

Returns: string

Allows you specify the Input/Output encoding for this database. On default it is "Latin-1". After you assign IOEncoding to object, all your strings to this object should be in this encoding, and Valentina also will return you strings in this encoding. See Valentina Kernel manual for details.

Example:

```
put VDataBase_IOEncoding(dbRef) into var

get VDatabase_IOEncoding( dbRef, "Greek" )
```

VDataBase_IndexCount(dbRef) (r/o)

Returns: integer

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

Example:

```
put VDataBase_IndexCount(dbRef) into var
```

VDataBase_IsEncrypted(dbRef) (r/o)

Returns: boolean

Returns TRUE if this database is encrypted.

Example:

```
put VDataBase_isEncrypted(dbRef) into var
```

VDataBase_IsStructureEncrypted(dbRef) (r/o)

Returns: boolean

Returns TRUE if this database structure is encrypted.

Example:

```
put VDataBase_IsStructureEncrypted(dbRef) into var
```

VDataBase_IsReadOnly(dbRef) (r/o)

Returns: boolean

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:

```
put VDataBase_ReadOnly(dbRef) into var
```

VDataBase_IsRemote(dbRef) (r/o)

Returns: boolean

Returns TRUE if this database is remote.

Example:

```
put VDataBase_IsRemote( dbRef ) into var
```

VDataBase_IsOpen(dbRef) (r/o)

Returns: boolean

Returns TRUE if this database is open now.

Example:

```
put VDataBase_IsOpen(dbRef) into var
```

VDatabase_LastInsertedRecID(dbRef) (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:

```
put VDatabase_LastInsertedRecID( dbRef ) into var
```

VDataBase_LinkCount(dbRef) (r/o)

Returns: integer

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:

```
put VDataBase_LinkCount(dbRef) into var
```

VDataBase_Locale(dbRef, [inStrLocale]) (r/w)

Returns: string

Define the Locale string for this database. Tables and fields of this database will inherit this parameter.

Example:

```
get VDataBase_locale( dbRef,"en_US")

put VDataBase_Locale(dbRef) into var
```

VDataBase_Mode(dbRef) (r/o)

Returns: EVDbMode

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

Example:

```
put VDataBase_Mode(dbRef) into var
```

VDataBase_Name(dbRef) (r/o)

Returns: string

The name of database.

Example:

```
put VDataBase_Name(dbRef) into var
```

VDataBase_Path(dbRef) (r/o)

Returns: string

The full path to this database.

Example:

```
put VDataBase_Path(dbRef) into var
```

VDataBase_SchemaVersion(dbRef, [inIntValue]) (r/w)

Returns: 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:

```
get VDataBase_SchemaVersion(dbRef,3)

put VDataBase_SchemaVersion(dbRef) into var
```

VDataBase_StorageEncoding(dbRef, [inStrValue]) (r/w)

Returns: 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:

```
get VDataBase_StorageEncoding(dbRef,"UTF-16")

put VDataBase_StorageEncoding(dbRef) into var
```

VDataBase_TableCount(dbRef) (r/o)

Returns: integer

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:

```
put VDataBase_TableCount(dbRef) into var
```

VDataBase_TimeSep(dbRef, [inStrValue]) (r/w)

Returns: string

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

Example:

```
get VDataBase_TimeSep(dbRef,":")

put VDataBase_TimeSep(dbRef) into var
```

VDataBase_ResponseTimeOut(dbRef, [inStrValue]) (r/w)

Returns: 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:

```
get VDataBase_ResponseTimeOut(dbRef,100)

put VDataBase_ResponseTimeOut(dbRef) into var
```

VDataBase_SegmentSize(dbRef) (r/o)

Returns: integer

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

Example:

```
put VDatabase_SegmentSize(dbRef)
```

Construction Methods

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

VDataBase_Constructor(inEnumStorageType = "kDefault")

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

Returns: dbRef

You should use this form of VDatabase constructor, 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:

```
-- Local Disk DB.
put VDatabase_Constructor() into dbRef
get VDataBase_Create( dbRef, path, "kDscDatBlbInd", 32 * 1024 )
```

Example:

```
-- Local RAM DB.
put VDatabase_Constructor( "kRAM" ) into dbRef
get VDataBase_Create( dbRef, "ram", "kDscDatBlbInd", 32 * 1024 )
```

VDataBase_Constructor(inConnectionRef)

Parameter	**Description**
inConnectionRef	VConnection object

Returns: dbRef

You need this form of VDatabase constructor to create a VDatabase object to access a remote database. The connection should be opened already.

Example:

```
-- Remote database on some VServer
put VConnection_Constructor( "somecompany.com", "sa", "sa" ) into connRef
get VConnection_Open( connRef )

put VDatabase_Constructor( connRef ) into dbRef
```

VDataBase_Constructor_FromRevDB(inDB_ID)

Parameter	Description
inDB_ID	IF of database object created by REVDB API.

Returns: dbRef

This form of VDatabase constructor plays role of bridge from REV DB API to Valentina API.

This allow you develop application using REVDB code, but still be able access Valentina specific functions.

Example:

```
put VDatabase_Constructor_FromRevDB( db_id ) into dbRef
```

VDatabase_Destructor(dbRef)

Returns: ZERO

This function must be called before Valentina_ShutDown() to free resources allocated by VDatabase_Constructor.

You may use form "put into dbRef" to zero dbRef.

Example:

```
// to clear dbRef you can use syntax with put:
put VDatabase_Destructor(dbRef) into dbRef

// although this is also enough:
get VDatabase_Destructor(dbRef)
```

Disk Functions

```
VDataBase_Create(
        dbRef,
        inStrPath,
        inEnumMode =  "kDsc_Dat_Blb_Ind",
        inIntSegmentSize = "32768",
        inEnumOS = kDefault )
```

Parameter:	**Description:**
dbRef	The reference of database object.
inStrPath	The path to the database on the disk.
inEnumMode	How many files for databases will be used, range 1-8; default 4.
inIntSegmentSize	The size of one cluster in the database file; default 32KB.
inEnumOS	The byte order for the database.

Creates a new, empty database on disk or in the RAM. For RAM dbs inStrPath parameter is ignored.

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:

```
--   Local disk DB.
get VDataBase_Create( dbRef, path, "kDscDatBlb_Ind", 32 * 1024 )
```

Example:

```
--   Local RAM DB.
get VDataBase_Create( dbRef, "ram", "kDscDatBlbInd", 32 * 1024 )
```

Example:

```
--  For a remote database, you need to specify only
--  the name of the database that is registered with Valentina Server.
--  Only name without file extension (although it not harm, just ignored).
get VDataBase_Create( dbRef, filename, kDscDatBlb_Ind, 32 * 1024 )
```

VDataBase_Flush(dbRef)

Parameter:	**Description:**
dbRef	The reference of database object.

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

Example:

```
get VDataBase_Flush(dbRef)
```

VDataBase_Open(dbRef, inStrPath)

Parameter:	**Description:**
dbRef	The reference of database object.
inStrPath	The path to the database on the disk.

Opens an existing database at the specified location.

Example:

```
get VDataBase_Open(dbRef,path)
```

Example:

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

get VDataBase_Open(dbRef,filename)
```

VDataBase_Close(dbRef)

Parameter:	**Description:**
dbRef	The reference of database object.

Closes the database.

Example:

```
get VDataBase_Open(dbRef)
        ....
get VDataBase_Close(dbRef)
```

VDataBase_ThrowOut(dbRef)

Parameter:	**Description:**
dbRef	The reference of database object.

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

Example:

```
get VDataBase_ThrowOut(dbRef)
```

VDatabase_SetMacTypes(dbRef,
inStrDescType,
inStrDatType,
inStrBlbType,
inStrIndType)

Parameter:	**Description:**
dbRef	The reference of database object.
inStrDescType	Mac Type of the ".vdb" file of the database.
inStrDatType	Mac Type of the ".dat" file of the database.
inStrBlbType	Mac Type of the ".blb" file of the database.
inStrIndType	Mac Type of the ".ind" file of the database.

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

Example:

```
get VDatabase_SetMacTypes(dbRef,"Mdsc","Mdat","Mblb","Mind")
```

VDatabase_Clone(inTarget, inLoadRecords = true, inDoLog = true)

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 Function 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:

```
get VDatabase_Clone( db, newDbLocation )
```

VDatabase_Clone(inTargetDb, inLoadRecords = true, inDoLog = false)

The same as above except that first parameter is not disc location, but already exictent 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() Function will copy the structure and records into this database.

Example:

```
put VDatabase_Constructor( ) into newDbRef
get VDatabase_Create( newDbRef, newDbLocation )

get VDatabase_Clone( db, newDbRef )
```

Database Structure Functions

VDataBase_CreateTable(dbRef,
inStrName,
[inEnumTableKind = "kTblPermanent"],
[inEnumStorageType = "kDefault"])

Returns: TableRef

Parameter:	**Description:**
dbRef	The reference of database object.
inStrName	The Name of a new Table.
inEnumTableKind	The kind of Table
inEnumStorageType	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() Function.

Example:

```
put VDataBase_CreateTable( dbRef, "Person" )  into tblPersonRef
```

VDataBase_DropTable(dbRef, tblRef)

Parameter:	**Description:**
dbRef	The reference of database object.
tblRef	The reference of Table to delete.

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

Example:

```
get VDataBase_DropTable( dbRef, tblPersonRef )
```

```
VDataBase_CreateBinaryLink( dbRef,
        inStrLinkName,
        inLeftTblRef,
        inRightTblRef,
        [inEnumLinkTypeLeft = "kOne"],
        [inEnumLinkTypeRight = "kMany"],
        [inEnumOnDelete = "kSetNull"],
        [inEnumStorageType = "kDefault"],
        [inBoolTemporary = false] )
```

Parameter:	**Description:**
dbRef	The reference of database object.
inStrLinkName	The name of the link.
inLeftTblRef	Pointer to the Left Table.
inRightTblRef	Pointer to the Right Table.
inEnumLinkTypeLeft	Link type for the Left Table.
inEnumLinkTypeRight	Link type for the Right Table.
inEnumOnDelete	The behavior on deletion of record-owner.
inEnumStorageType	Storage type of the link.
inBoolTemporary	TRUE if the link is temporary.

Returns: LinkRef

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:

```
put VDataBase_CreateBinaryLink( dbRef,
        "PersonPhone", tblPerson, tblPhone, "kMany", "kType.kMany")
into linkPersonPhoneRef
```

```
VDataBase_CreateForeignKeyLink( dbRef,
        inStrLinkName,
        inKeyFieldRef,
        inPtrFieldRef,
        [inEnumOnDelete = "kSetNull"],
        [inEnumOnUpdate = "kCascade"],
        [inBoolTemporary = false] )
```

Parameter:	**Description:**
dbRef	The reference of database object.
inStrLinkName	The name of link.
inKeyFieldRef	The PRIMARY KEY field of ONE Table.
inPtrFieldRef	The PTR field in the MANY Table.
inEnumOnDelete	The behavior on deletion of record-owner.
inEnumOnUpdate	The behavior on update of record-owner.
inBoolTemporary	TRUE if link is temprary.

Returns: LinkRef

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:

```
put VDataBase_CreateForeignKeyLink(dbRef,
        "PersonPhone",tblPerson.fldID, tblPhone.PersonPtr )  into linkRef
```

```
VDataBase_DropLink( dbRef, lnkRef )
```

Parameter:	**Description:**
dbRef	The reference of database object.
lnkRef	The reference of Link to delete.

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

Example:

```
get VDataBase_DropLink( dbRef, linkRef )
```

Table Functions

VDataBase_Table(dbRef, inIntIndexOrStrName)

Parameter:	Description:
dbRef	The reference of database object.
inIntIndexOrStrName	The index of a Table in a database, start from 1.

Returns: TableRef

Returns a Table by an numeric index.

Example:

```
put VDataBase_Table( dbRef, i )  into tbl
```

Example:

```
put VDataBase_Table( dbRef, "Person" )  into tbl
```

Link Functions

VDataBase_Link(dbRef, inIntIndexOrStrName)

Parameter:	Description:
dbRef	The reference of database object.
inIntIndexOrStrName	The index of a Link in a database, start from 1.

Returns: LinkRef

Returns a Link based on numeric index.

Example:

```
put VDataBase_Link( dbRef, i )   into link
```

Example:

```
put VDataBase_Link( dbRef, "PersonPhone" ) into link
```

SQL Functions

```
VDatabase_SqlSelect(
        dbRef,
        inStrQuery,
        inEnumCursorLocation = "kClientSide",
        inEnumLockType = "kReadOnly",
        inEnumDirection = "kForwardOnly",
        inArrayNameOfBinds = "" )
```

Parameter:	**Description:**
dbRef	The reference of database object.
inStrQuery	The SQL string of a query.
inEnumCursorLocation	The location of cursor.
inEnumLOckType	The lock type for records of a cursor.
inEnumDirection	The direction of a cursor.
inArrayNameOfBinds	The array of bound parameters.

Returns: CursorRef

SqlSelect() Function 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,	kServerSide	
inLockType:	kNoLocks,	kReadOnly,	kReadWrite
inCursorDirection:	kForwardOnly,	kRandom	

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

For the SELECT command you can define an array of bound parameters. This is an array of strings for V4REV. See for details the Valentina SQL section of Valentina WIKI.

Example:

```
put VDatabase_SqlSelect(dbRef,"SELECT * FROM T ") into curs
```

Example:

```
put VDatabase_SqlSelect(
        dbRef, "SELECT * FROM T ",
        "kServerSide", "kReadWrite", "kRandom") into curs
```

Example:

```
put "john" into tArray[1]
put "25"    into tArray[2]

-- NOTE quotes around "tArray" !!
put VDatabase_SqlSelect(
        dbRef, "SELECT * FROM T WHERE f1 = :1,f2 > :2",
        "kServerSide",  "kReadWrite", "kRandom",  "tArray" ) into curs
```

```
VDatabase_SqlQuery(
        dbRef,
        inStrQuery,
        inEnumCursorLocation = "kClientSide",
        inEnumLockType = "kReadOnly",
        inEnumDirection = "kForwardOnly",
        inArrayNameOfBinds = "" )
```

Parameter:	**Description:**
dbRef	The reference of database object.
inStrQuery	The SQL string of a query.
inEnumCursorLocation	The location of cursor.
inEnumLOckType	The lock type for records of a cursor.
inEnumDirection	The direction of a cursor.
inArrayNameOfBinds	The array of bound parameters.

Returns: VQueryResultRef

Description:

VDatabase_SqlQuery() method is very similar to VDatabase_SqlSelect() by syntax, so see description of parameters in that method. Difference is that VDatabase_SqlQuery() is able to accept any SQL command, i.e. it is combination of both VDatabase_SqlExecute() and VDatabase_SqlSelect() methods.

As result SqlQuery() returns VQueryResult - a small class, which is able to keep any result of any Valentina SQL command.

This command can be useful if you must be able accept any SQL command and you don't know what exactly this command is. For example this can be if user type SQL query self or if you get SQL command from some file.

Example:

```
put VDatabase_SqlQuery( strAnySqlCommand ) into vres

if VQueryResult_Type( vres ) = "kCursor" then
  put VQuery_Cursor(  vres ) into cursRef
end if

-- some work with cursor and later destroy it:

VCursor_Destructor( cursRef )
```

VDatabase_SqlExecute(dbRef, inStrCommand, inBinds)

Parameter:	**Description:**
dbRef	The reference of database object.
inStrCommand	The SQL string of a query.
inBinds	The array of bound parameters.

Returns: integer

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 Valentina SQL.

Returns the number of affected rows.

For commands that have an EXPR (expression) clause in the syntax, you can define an array of bound parameters. This is an array of strings for V4REV. See Valentina SQL manual for details.

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

Example:

```
put "UPDATE person SET name = 'john' WHERE name = 'jehn'" into query
put VDatabase_SQLExecute( dbRef, query ) into affectedRows
```

Example:

```
put "john" into tArray[1]
put "jehn" into tArray[2]

put "UPDATE person SET name = :1 WHERE name = :2" into query

-- NOTE quotes around "tArray" !!
put VDatabase_SQLExecute( dbRef, query", "tArray" )  into affectedRows
```

```
VDtabase_SQLSelectRecords(
                dbRef,
                inStrQuery,
                inEnumCursorLocation = 'kClientSide",
                inEnumLockType = 'kReadOnly",
                inArrayNameOfBinds = nil,
                inFromRec = 1,
                inMaxRecords = -1,
                inFldDelim = "\t",
                inRecDelim = '\n',
                inFldPrefix = "",
                inFldSuffix = "",
                inRecPrefix = "",
                inRecSuffix = "" )
```

Parameter:	**Description:**
dbRef	The reference of database object.
inStrQuery	The SQL string of a query.
inEnumCursorLocation	The location of cursor.
inEnumLOckType	The lock type for records of a cursor.
inArrayNameofBinds	The array of bound parameters.
inFromRec	The index of the start record.Default is current.
inMaxRec	The maximal number of the records to return. Default – all.
inFldDelim	The char to be used as the delimiter of the fields.
inRecDlmit	The char to be used as the delimiter of the record.
inFldPrefix	The prefix for each field.
inFldSuffix	The suffix for each field.
inRecPrefix	The prefix for record.
inRecSuffix	The suffix for record.

Returns: string

This method is combination of few actions, so it allows you to write less code to get the result of query as string with records delimited as you specify.

This single method is analog of the following actions:

* put VDatabase_SqlSelect(
  query, cursLocation, LockType, "ForwardOnly", arrBindings) into cursorRef
* loop to extract values of cursor fields.
* Cursor_Destroy(cursorRef)

This method can be useful to get easy records as, for example, HTML tables, or XML. For XML, although, in Valentina 4.0 was introduced SELECT ... FOR XML command.

Example:

```
put VDatabase_SQLSelectRecords(
        dbRef, "SELECT * FROM T ",
        "kServerSide", "kReadWrite", "",
        1, 3, "", "\n",  "<td>", "</td>" , "<tr>", "</tr>" )  into curs
```

result can be for example, as:

```
<tr><td>Person1</td><td>111111</td></tr>
<tr><td>Person2</td><td>2222</td></tr>
<tr><td>Person3</td><td>333333</td></tr>
```

IndexStyle Functions

VDataBase_CreateIndexStyle(dbRef, inStrName)

Parameter:	Description:
dbRef	The reference of database object.
inStrName	The name of an index style.

Returns: IndexStyleRef

Creates a new Index Style in the database.

Example:

```
put VDataBase_CreateIndexStyle( dbRef,"myStyle" ) into var
```

VDataBase_DropIndexStyle(dbRef, indexStyleRef)

Parameter:	Description:
dbRef	The reference of database object.
indexStyleRef	The index style to be deleted.

Deletes the specified index style from the database.

Example:

```
get VDataBase_DropIndexStyle( dbRef, IndexStyle1 )
```

VDataBase_IndexStyle(dbRef, inStrName)

Parameter:	Description:
dbRef	The reference of database object.
inStrName	The Name of a IndexStyle.

Returns: IndexStyleRef

Returns an IndexStyle by name.

Note: The parameter Name is case insensitive.

Example:

```
put VDataBase_IndexStyle( dbRef, "IndexStyle1" ) into var
```

Encryption Functions

The VDataBase Class has encryption Functions 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 Functions of the database, though it is posible to merge both of them.

VDatabase_Encrypt(dbRef, inKey, inForData = "kRecordsOnly")

Parameter:	Description:
dbRef	The reference of database object.
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:

```
get VDatabase_Open()
get VDatabase_.Encrypt ( db, "key12345" )
```

Example:

```
get VDatabase_Open()
get VDatabase_.Encrypt  ( db, "key12345", "kStructureOnly" )
```

VdataBase_Decrypt(dbRef, inKey, inForData = "kRecordsOnly")

Parameter:	**Description:**
dbRef	The reference of database object.
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:

```
get VDatabase_Open()
get VDatabase_Decrypt ( db, "key12345" )
```

Example:

```
get VDatabase_Open()
get VDatabase_Encrypt ( db, "key12345", "kStructureOnly" )
```

VDataBase_ChangeEncryptionKey(dbRef,
inOldKey
inNewKey
inForData = "kRecordsOnly")

Parameter:	**Description:**
dbRef	The reference of database object.
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:

```
get VDatabase_ChangeEncryptionKey ( db, "key12345" )
```

Example:

```
get VDatabase_ChangeEncryptionKey (
        db, "key12345", "key54321",  "kStructureOnly" )
```

VDataBase_RequiresEncryptionKey(dbRef) as boolean

Parameter:	**Description:**
dbRef	The reference of database object.

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:

```
put VDatabase_RequiresEncryptionKey() into res
```

VDataBase_UseEncryptionKey(dbRef,
inKey
inForData ="kRecordsOnly")

Parameter:	**Description:**
dbRef	The reference of database object.
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:

```
get VDatabase_.UseEncryptionKey( db, "key12345" )
get VDatabase_Open()
```

Example:

```
get VDatabase_UseEncryptionKey( db, "key12345", "kStructureOnly" )
get VDatabase_Open()
```

Dump Functions

```
VDataBase_Dump( dbRef,
        inStrDumpPath,
        inEnumDumpType,
        inEnumDumpData = "kStructureAndRecords",
        inBoolFormatDump = "false",
        inEncoding = "UTF-16" )
```

Parameter:	Description:
dbRef	The reference of database object.
inStrDumpPath	The location of dump file.
inEnumDumpType	The Type of dump.
inEnumDumpData	Specify which information to dump.
inBoolFormatDump	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:

```
get VDataBase_Dump( dbRef, strDumpPath )
```

```
VDataBase_LoadDump( dbRef,
        inStrDumpPath,
        inStrNewDBPath,
        inEnumDumpType,
        inEncoding = "UTF-16" )
```

Parameter:	**Description:**
dbRef	The reference of database object.
inStrDumpPath	The location of a dump file.
inStrNewDBPath	The location for a new database.
inEnumDumpType	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 in sense, that after it is done, on disk appears new db files and database in the opened state.

Example:

```
get VDataBase_LoadDump( dbRef, strDumpPath, NewDb, "kXML" )
```

VTable Class

Properties

```
VTable_CollationAttribute( tblRef, inEnumColAttribute, [inEnumColAttributeVallue] )
VTable_Database( tblRef )  (r/o)          --  Database of this Table.
VTable_FieldCount( tblRef )  (r/o)        --  number of fields in this Table
VTable_ID( tblRef )  (r/o)
VTable_IsEncrypted( tblRef ) (r/o)
VTable_IOEncoding( tblRef, [inStrValue] )
VTable_LinkCount( tblRef ) (r/o)
VTable_Locale( tblRef, [inStrValue] )
VTable_Name( tblRef, [inStrValue] )
VTable_PhysicalRecordCount( tblRef )  (r/o)
VTable_RecID( tblRef, [inIntValue] )
VTable_RecordCount( tblRef )  (r/o)       --  the number of logical records in this Table.
VTable_StorageEncoding( tblRef, [inStrValue] )
```

Field Functions

```
VTable_Field( tblRef, inIntIndexOrStrName )
VTable_FieldValue( tblRef, inIntIndexOrStrName, [inStrValue] )
```

Link Functions

```
VTable_Link( tblRef, inIntIndexOrStrName )
```

Record Functions

```
                                    --  Clears a memory buffer of a Table,
                                    --  set nullable fields to NULL
VTable_SetBlank( tblRef, [inEnumAccess = "forUpdate"] )

VTable_AddRecord( tblRef )          --  Adds a new record with the current value of fields

VTable_DeleteRecord( tblRef )       --  Deletes the current record
VTable_DeleteAllRecords( tblRef, [inSetRef = 0] ) --  Makes table empty, very fast.

VTable_UpdateRecord( tblRef )       --  Updates an existing record with new values
VTable_UpdateAllRecords( tblRef, [inSetRef = 0] )
```

Cach Functions

```
VTable_Flush( tblRef )              --  Saves information of this Table on disk only.
```

Navigation Functions

```
VTable_FirstRecord( tblRef )
VTable_LastRecord( tblRef )
VTable_PrevRecord( tblRef )
VTable_NextRecord( tblRef )

VTable_RecordExists( tblRef, inRecID )
```

```
-- Set of handy CreateXXXField()

CreateBooleanField          ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateByteField             ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
VTable_CreateShortField     ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateUShortField           ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateMediumField           ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateUMediumField          ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateLongField             ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateULongField            ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateLLongField            ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateULLongField           ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateFloatField            ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateDoubleField           ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateDateField             ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateTimeField             ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )
CreateDateTimeField         ( tblRef, inStrName, [inEnumFlags =  "fNone"], [inStrFunction = ""] )

VTable_CreateStringField    ( tblRef, inStrName, inIntMaxLength, [inEnumFlags = "fNone"], [inStrFunction = ”"])
VTable_CreateVarCharField( tblRef, inStrName, inIntMaxLength, [inEnumFlags = "fNone"], [inStrFunction = ”"])

VTable_CreateFixedBinaryField( tblRef, inStrName,  inIntMaxLength )
VTable_CreateVarBinaryField    ( tblRef, inStrName, inIntMaxLength )


VTable_CreateBLOBField( tblRef, inStrName, inIntSegmentSize )
VTable_CreateTextField   ( tblRef, inStrName, inIntSegmentSize, [inEnumFlags = "fNone"], [inStrFunction = ”"] )

VTable_CreatePictureField( tblRef, inStrName, inIntSegmentSize )

VTable_CreateObjectPtrField(
        tblRef,
        inStrName,
        inTargetTblRef,
        [inEnumOnDeletion = "kCascade"],
        [inEnumFlags =  "fNone"] )
```

Schema Functions

DropField(inFld)

ChangeType(inFld, inNewType, inParam1)

Encryption Functions

VTable_Encrypt(tblRef, inStrKey)
VTable_Decrypt(tblRef, inStrKey)
VTable_ChangeEncryptionKey(tblRef, inStrOldKey, inStrNewKey)

VTable_RequiresEncryptionKey(tblRef)
VTable_UseEncryptionKey(tblRef, inStrKey)

Dump Functions

```
VTable_Dump( tblRef,
        dbRef,
        inStrDumpPath,
        inEnumDumpType,
        [inEnumDataKind = "kStructureAndRecords"],
        [inBoolFormatDump = false] )

VTable_LoadDump( tblRef,
        dbRef,
        inStrDumpPath,
        inEnumDumpType )
```

Selection Functions

VTable_SelectAllRecords(tblRef)
VTable_SelectNoneRecords(tblRef)

VTable_Sort(tblRef, setRef, fidRef, [inAscending = true])

```
VTable_SortN( tblRef,
        setRef,
        sortItemDataRef1],
        [sortItemDataRef2 = nil],
        [sortItemDataRef3 = nil],
        [sortItemDataRef4 = nil] )
```

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

VTable_CollationAttribute(tblRef, inEnumColAttribute, [inEnumColAttributeVallue])

Returns: EVColAttributeValue

The value of the specified collation attribute for this table.

Example:

```
get VTable_CollationAttribute(tblRef,"kStrength")

put VTable_CollationAttribute(tblRef,'kStrength") into var
```

VTable_Database(tblRef) (r/o)

Returns: DataBaseRef

Returns the parent database of this table.

Example:

```
put VTable_Database( tblRef ) into var
```

VTable_FieldCount(tblRef) (r/o)

Returns: integer

Returns the number of custom fields in the table.

Example:

```
put VTable_FieldCount( tblRef ) into var
```

VTable_ID(tblRef) (r/o)

Returns: integer

Returns the unique identifer of the table.

Example:

```
put VTable_ID( tblRef ) into var
```

VTable_IOEncoding(dbRef, [inStrValue])

Returns: string

Allows you specify the Input/Output encoding for this database. On default it is "Latin-1". After you assign IOEncoding to object, all your strings to this object should be in this encoding, and Valentina also will return you strings in this encoding. See ValentinaKernel manual for details.

Example:

```
put VTable_IOEncoding( tblRef ) into var

get VTable_IOEncoding( tblRef, "Greek" )
```

VTable_IsEncrypted(tblRef) (r/o)

Returns: boolean

Returns TRUE if the database is encrypted.

Example:

```
put VTable_IsEncrypted( tblRef ) into var
```

VTable_LinkCount(tblRef) (r/o)

Returns: integer

Returns the number of links in the table.

Example:

```
put VTable_LinkCount( tblRef ) into var
```

VTable_Locale(tblRef, [inStrValue])

Returns: string

Specifies for this table the locale name.

Example:

```
get VTable_Locale(tblRef,"en_US")
put VTable_Locale( tblRef ) into var
```

VTable_Name(tblRef, [inStrValue])

Returns: string

The name of the table.

Example:

```
get VTable_Name( tblRef,"NewName" )
put VTable_Name( tblRef ) into var
```

VTable_PhysicalRecordCount(tblRef) (r/o)

Returns: integer

Returns the number of physical records in the table.

Example:

```
put VTable_PhysicalRecordCount( tblRef ) into var
```

VTable_RecID(tblRef, [inIntValue])

Returns: 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:

```
get VTable_RecID(tblRef,5)

put VTable_RecID( tblRef ) into var
```

VTable_RecordCount(tblRef) (r/o)

Returns: integer

Returns the number of logical records in the table.

Example:

```
put VTable_RecordCount( tblRef ) into var
```

VTable_StorageEncoding(tblRef, [inStrValue])

Returns: string

Specifies for this table the string encoding stored on disk.

Example:

```
get VTable_StorageEncoding(tblRef,"UTF-16")

put VTable_StorageEncoding( tblRef ) into var
```

Field Functions

VTable_Field(tblRef, inIntIndexOrStrName)

Parameter:	Description:
tblRef	The reference of Table object.
inIntIndexOrStrName	The index of a Field in a Cursor (starts from 1) or name of a field.

Returns: FieldRef

This Function 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:

```
put VTable_Field(tblRef,1) into var
```

Example:

```
put VTable_Field(tblRef,"LastName") into fldRef
```

VTable_FieldValue(db Ref, inIntIndexOrStrName, [inStrValue])

Parameter:	Description:
tblRef	The reference of Table object.
inIntIndexOrStrName	The index of a Field in a Cursor (starts from 1) or name of a Field.
inStrValue	Value of the field.

The form of this function with two parameters returns a value of cursor field.
The form of it with three parameters can assign a new value to cursor field.
This function allows to you write one line of code instead of two.

The order of fields in the cursor is the same as the order of fields in the SELECT statement of the query.

Example:

```
put VTable_FieldValue( tblRef ) into value
```

Example:

```
get VTable_FieldValue( tblRef, 1, "Name" )
```

Link Functions

VTable_Link(tblRef, inIntIndexOrStrName)

Parameter:	Description:
tblRef	The reference of Table object.
inIntIndexOrStrName	The index of a link.

Returns: LinkRef

Returns a link of this table by numeric index.

Example:

```
put VTable_Link( tblRef,1) into linkRef
```

Example:

```
put VTable_Link( tblRef,"link1") into linkRef
```

Record Functions

VTable_SetBlank(tblRef, [inEnumAccess = "forUpdate"])

Parameter	Description
tblRef	The reference of Table object.
inEnumAccess	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() Function, 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:

```
get VTable_SetBlank( tblRef )
```

VTable_AddRecord(tblRef)

Parameter:	Description:
tblRef	The reference of Table object.

Returns: 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:

```
put VTable_AddRecord( tblRef )  into var
```

VTable_DeleteRecord(tblRef)

Parameter:	Description:
tblRef	The reference of Table object.

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:

```
get VTable_DeleteRecord( tblRef )
```

VTable_DeleteAllRecords(tblRef, [inSetRef = 0])

Parameter	**Description**
tblRef	The reference of Table object.
inSetRef	The selection of records.

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

Example:

```
get VTable_DeleteAllRecords( tblRef )
```

VTable_UpdateRecord(tblRef)

Parameter:	**Description:**
tblRef	The reference of Table object.

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

Example:

```
get VTable_UpdateRecord( tblRef )
```

VTable_UpdateAllRecords(tblRef, [inSetRef = 0])

Parameter	**Description**
tblRef	The reference of Table object.
inSet	The selection of records.

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

Example:

```
get VTable_UpdateAllRecords( tblRef )
```

Cache Functions

VTable_Flush(tblRef)

Parameter:	Description:
tblRef	The reference of Table object.

This Function 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:

```
get VTable_Flush(tblRef)
```

Navigation Functions

VTable_FirstRecord(tblRef)

Parameter:	Description:
tblRef	The reference of Table object.

Returns: 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:

```
put VTable_FirstRecord( tblRef )  into var
```

VTable_LastRecord(tblRef)

Parameter:	Description:
tblRef	The reference of Table object.

Returns: 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:

```
put VTable_LastRecord( tblRef )  into var
```

VTable_PrevRecord(tblRef)

Parameter:	Description:
tblRef	The reference of Table object.

Returns: 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:

```
put VTable_PrevRecord( tblRef )  into var
```

VTable_NextRecord(tblRef)

Parameter:	**Description:**
tblRef	The reference of Table object.

Returns: 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:

```
put VTable_NextRecord( tblRef )   into var
```

VTable_RecordExists(tblRef, inRecID)

Parameter:	**Description:**
tblRef	The reference of Table object.
inRecID	RecID of a record.

Returns: boolean

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

Example:

```
put VTable_RecordExists(tblRef,1)   into var
```

Working with Database Structure

The Valentina API for Revolution lets you not only create or work with static database structures but also exposes you to Functions 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 Revolution provides the set of Functions to create fields. There exists several groups of Functions which have similar parameters. So we will describe the groups of these Functions.

Functions to create numeric fields

```
VTable_CreateShortField( tblRef,
        inStrName,
        [inEnumFlags = EVFlag.fNone,
        inStrFunction = ""] )
```

Parameter:	**Description:**
tblRef	The reference of Table object.
inStrName	The name of the field.
inEnumFlags	The flags of the field.
inStrFunction	The text of the Function for a calculation field.

Returns: ShortRef

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

- 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 Function text.

Example:

```
put VTable_CreateShortField(tblRef,
                "age",EVFlags.fNullable + EVFlags.fIndexed)   into var
```

Functions to create string/varchar fields

```
VTable_CreateStringField( tblRef,
        inStrName,
        inIntMaxLength,
        inEnumFlags = EVFlag.fNone,
        inStrFunction = "")
```

Returns: StringRef

```
VTable_CreateVarCharField( tblRef,
        inStrName,
        inIntMaxLength,
        [inEnumFlags = 0,
        inStrFunction = ""])
```

Parameter:	**Description:**
tblRef	The reference of Table object.
inStrName	The name of the field.
inIntMaxLength	The maximum length (in characters).
inFlags	The flags of the field.
inStrFunction	The text of the Function for a calculation field.

Returns: VarCharRef

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 Function text.

Example

```
put VTable_CreateStringField(tblRef,
        "name",40,EVFlags.fNullable + EVFlags.fIndexed)   into var
```

Functions to create fixed/var binary fields

```
VTable_CreateFixedBinaryField( tblRef,
        inStrName,
        inIntMaxLength )
```

Returns: FixedBinaryRef

```
VTable_CreateVarBinaryField( tblRef,
        inStrName,
        inIntMaxLength )
```

Parameter:	**Description:**
tblRef	The reference of Table object.
inStrName	The name of the field.
inIntMaxLength	The maximum length (in bytes)

Returns: VarBinaryRef

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

Example

```
put VTable_FixedBinaryField(tblRef,
        "nameStile",40,EVFlags.fNullable + EVFlags.fIndexed)   into var
```

Function to create BLOB fields.

```
VTable_CreateBLOBField( tblRef,
        inStrName,
        inIntSegmentSize )
```

Parameter:	**Description:**
tblRef	The reference of Table object.
inStrName	The name of the field.
inIntSegmentSize	The segment size of the BLOB field.

Returns: BLOBRef

Create a BLOB (Binary Large Object) field.

• You need to specify the segment size in bytes.

Example

```
put VTable_CreateBLOBField(tblRef,
        "notesStyle",256)   into var
```

Function to create TEXT fields.

```
VTable_CreateTextField( tblRef,
        inStrName,
        inIntSegmentSizer,
        [inEnumFlags = EVFlag.fNone,
        inStrFunction = ""] ) as VText
```

Parameter:	**Description:**
tblRef	The reference of Table object.
inStrName	The name of the field.
inIntSegmentSize	The segment size of the BLOB field.
inFlags	The flags of the field.
inStrFunction	The text of the Function for a calculation field.

Returns: TextRef

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 Function text.

Example

```
put VTable_CreateTextField(tblRef,
        "notes",256,EVFlags.fNullable + EVFlags.fIndexed)   into var
```

Function to create Picture fields.

```
VTable_CreatePictureField( tblRef,
        inStrName,
        inIntSegmentSize)
```

Parameter:	**Description:**
tblRef	The reference of Table object.
inStrName	The name of the field.
inIntSegmentSize	The segment size of the field.

Returns: PictureRef

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

Example

```
        put VTable_CreatePictureField(tblRef,
                "foto",256,"fNullable" + "fIndexed")   into var
```

Function to create ObjectPtr fields.

```
VTable_CreateObjectPtrField( tblRef,
        inStrName,
        inTargetTblRef,
        [inEnumOnDeletion = kCascade,
        inEnumFlags = fNone] )
```

Parameter:	**Description:**
tblRef	The reference of Table object.
inStrName	The name of the field.
inTargetTblRef	The target table.
inEnumOnDeletion	The behavior on deletion of the record-owner.
inEnumFlags	The flags of the field.

Returns: ObjectPtrRef

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

```
        put VTable_CreateObjectPtrField(tblRef,
                "ParentPtr","fNullable" + "fIndexed')   into var
```

VTable_DropField(tblRef, fldRef)

Parameter:	**Description:**
tblRef	The reference of Table object.
fldRef	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:

```
get VTable_DropField(tblRef,fld)
```

VTable_ChangeType(tblRef,
fldRef,
inEnumFieldType,
inIntParam)

Parameter:	**Description:**
tblRef	The reference of Table object.
fldRef	The field whose type should be changed.
inEnumFieldType	New type for a field.
inIntParam	The Additional parameter (see below).

Returns: FldRef

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:

```
put VTable_ChangeType( tblRef, fld, "kTypeString", 40 )  into newFldRef
```

VTable Encryption Functions

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.

VDatabase_Encrypt(tblRef, inKey)

Parameter:	Description:
tblRef	The reference of Table object.
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:

```
get VDatabase_Encrypt( tbl, "key12345" )
```

VDatabase_Decrypt(tblRef, inKey)

Parameter:	Description:
tblRef	The reference of Table object.
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:

```
get VDatabase_Decrypt( tbl, "key12345" )
```

VDatabase_ChangeEncryptionKey(tblRef,
inOldKey,
inNewKey)

Параметр:	**Описание:**
tblRef	The reference of Table object.
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:

```
get VDatabase_ChangeEncryptionKey( tbl, "key12345", "key54321"  )
```

VDatabase_RequiresEncryptionKey(tblRef)

Parameter:	**Description:**
tblRef	The reference of Table object.

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 Function 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:

```
put VDatabase_RequiresEncryptionKey() t res
```

VDatabase_UseEncryptionKey(tblRef, inKey)

Parameter:	**Description:**
tblRef	The reference of Table object.
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() Function must be called before opening of the database, the VTable.UseEncryptionKey() Functions must be called after opening the database and before the first attempt to work with data of the Table.

Example:

```
get VDatabase_EncryptionKey( tbl, "key12345" )
get VDatabase_Open()

get VDatabase_UseEncryptionKey( tbl, "key12345" )
```

Dump Functions

VTable_Dump(
tblRef,
inStrDumpPath,
inEnumDumpType,
[inEnumDataKind = "kStructureAndRecords"],
[inBoolFormatDump = false])

Parameter:	Description:
tblRef	The reference of Table object.
inStrDumpPath	The location of the dump file.
inEnumDumpType	The Type of dump.
inEnumDumpData	Specify which information to dump.
inBoolFormatDump	If TRUE then formats the dump file to be human readable.

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

Example:

```
get VTable_Dump( tblRef, strDumpPath, "kXML" )
```

VTable_LoadDump(
tblRef,
inStrDumpPath,
inEnumDumpType)

Parameter:	Description:
tblRef	The reference of Table object.
inStrDumpPath	The location of the dump file.
inEnumDumpType	The type of dump.

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

Example:

```
get VTable_LoadDump( tblRef, strDumpPath, "kXML" )
```

Selection Functions

VTable_SelectAllRecords(tblRef)

Parameter:	Description:
tblRef	The reference of Table object.

Returns: BitSetRef

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

Example:

```
put VTable_SelectAllRecords( tblRef ) into bitset1
```

VTable_SelectNoneRecords(tblRef)

Parameter:	Description:
tblRef	The reference of Table object.

Returns: BitSetRef

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:

```
put VTable_SelectNoneRecords( tblRef ) into bitset2
```

VTable_Sort(tblRef, setRef, fldRef, [inAscending = true])

Parameter:	**Description:**
tblRef	The reference of Table object.
setRef	The set of records to be sorted. .
fldRef	The field on which to do sorting.
inAscending	The direction of sorting.

Returns: ArraySetRef

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:

```
put VTable_Sort( tblRef, allRecs, fldName ) into var
```

VField Class

Properties

```
VField_CollationAttribute( fldRef, inEnumColAttribute, [inEnumColAttributeValue] )
VField_DefaultValue( fldRef, [inStrValue] )
VField_ID( fldRef )  (r/o)
VField_IndexStyle( fldRef, [inIndexStyleRef] )
VField_IOEncoding( dbRef, [inStrValue] )
VField_Locale( fldRef, [inStrValue] )
VField_MethodText( fldRef, [inStrValue] )
VField_Name( fldRef, [inStrValue] )            --  up to 32 bytes
VField_StorageEncoding( fldRef, [inStrValue] )
VField_Table( fldRef )  (r/o)
VField_Type( fldRef )  (r/o)
VField_TypeString( fldRef )  (r/o)


VField_IsEncrypted( fldRef )  (r/o)
VField_IsIndexed( fldRef, [inBoolValue] )
VField_IsMethod( fldRef )  (r/o)               --  TRUE if the field is a Function.
VField_IsNullable( fldRef, [inBoolValue] )     --  TRUE if the field accepts NULL values
VField_IsUnique( fldRef, [inBoolValue] )       --  TRUE the field only has unique values
```

Value Functions and Properties

```
VField_SetBlank( fldRef )                      --  clear the value of the field.

VField_IsNull( fldRef, [inBoolValue] )         --   TRUE if the current value of the field is  NULL.
VField_Value( fldRef, [inStrValue] )
```

Search Functions

```
// Single value searches:

VField_ValueExists_1( fldRef, inStrValue, [inSetRef = nil], inEnumSearch = "kPreferIndexed"] )
VField_ValueExists_2( fldRef, inStrValue, [inSetRef = nill, inEnumSearch = "kPreferIndexed"] )

VField_FindValue(
        fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

VField_FindValueAsArraySet(
        fldRef,
        inStrValue,
        [inSetRef = nil,
        inIntMaxCount = &hffffffff,
        inEnumSearch = "kPreferIndexed"] )
```

```
// Range seaches:

VField_FindRange( fldRef,
        inBoolLeftInclude,
        inStrLeftValue,
        inStrRightValue,
        inBoolRightInclude,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

VField_FindRangeAsArraySet( fldRef,
        inIntLeftInclude,
        inStrLeftValue,
        inStrRightValue,
        inIntRightInclude,
        [inSetRef = nil,
        inIntMaxCount = &hffffffff,
        inEnumSearch = "kPreferIndexed"] )

VField_FindSingle( fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

VField_FindDistinct( fldRef,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

// NULL searches:

VField_FindNulls( fldRef,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

VField_FindNotNulls( fldRef,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )
```

```
// String searches:

VField_FindStartsWith( fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

VField_FindContains( fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

VField_FindEndsWith( fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

VField_FindLike( fldRef,
        inStrValue,
        [inStrEscapeChar = "\",
        inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )

VField_FindRegEx( fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch =  kPreferIndexed] )
```

Encryption Functions

```
VField_Encrypt( fldRef, inStrKey )
VField_Decrypt( fldRef, inStrKey )
VField_ChangeEncryptionKey( fldRef, inStrOldKey, inStrNewKey )

VField_RequiresEncryptionKey( fldRef )
VField_UseEncryptionKey( fldRef, inStrKey )
```

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.

Properties

VField_CollationAttribute(fldRef, inEnumColAttribute, [inEnumColAttributeValue])

Returns: EVColAttributeValue

The value of the specified collation attribute for this table.

Example:

```
get VField_CollationAttribute( fldRef, "kStrength" )

put VField_CollationAttribute( fldRef ) into var
```

VField_DefaultValue(fldRef, [inStrValue])

Returns: Variant

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:

```
get VField_DefaultValue( fldRef )

put VField_DefaultValue( fldRef ) into var
```

VField_ID(fldRef) (r/o)

Returns: integer

Return the unique identifier of the field.

Example:

```
put VField_ID( fldRef ) into var
```

VField_IndexStyle(fldRef, [inIndexStyleRef])

Returns: IndexStyleRef

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:

```
get VField_IndexStyle( fldRef )

put VField_IndexStyle( fldRef ) into var
```

VField_IsEncrypted(fldRef) (r/o)

Returns: boolean

Returns TRUE if the database is encrypted.

Example:

```
put VField_IsEncrypted( fldRef ) into var
```

VField_IsIndexed(fldRef, [inBoolValue])

Returns: boolean

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

Example:

```
get VField_IsIndexed(fldRef,FALSE)
        ... --   add many records for example
get VField_IsIndexed( fldRef,TRUE)

put VField_IsIndexed( fldRef ) into var
```

VField_IsMethod(fldRef) (r/o)

Returns: boolean

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

Example:

```
put VField_IsMethod( fldRef) into var
```

VField_IOEncoding(dbRef, [inStrValue])

Returns: string

Allows you specify the Input/Output encoding for this database. On default it is "Latin-1". After you assign IOEncoding to object, all your strings to this object should be in this encoding, and Valentina also will return you strings in this encoding. See ValentinaKernel manual for details.

Example:

```
put VField_IOEncoding( fldRef ) into var

get VField_IOEncoding( fldRef, "Greek" )
```

VField_IsNullable(fldRef, [inBoolValue])

Returns: boolean

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

Example:

```
get VField_IsNullable( fldRef, TRUE )

put VField_Nullable( fldRef ) into var
```

VField_IsUnique(fldRef, [inBoolValue])

Returns: boolean

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

Example:

```
get VField_IsUnique( fldRef, TRUE )

put VField_IsUnique( fldRef ) into var
```

VField_Locale(fldRef, [inStrValue])

Returns: string

Specifies for this field the locale name. Also can be used to obtain local name for this field.

Example:

```
get VField_Locale( fldRef, "en_US" )

put VField_Locale( fldRef ) int
```

VField_MethodText(fldRef, [inStrValue])

Returns: string

Returns the text of Table Method (calculated field). Also you can use this property to change the text of Table Method.

Example:

```
get VField_MethodText (fldRef,“CONCAT(fldRef,FirstName,‘ ‘,LastName)”)

put VField_MethodText( fldRef ) into var
```

VField_Name(fldRef, [inStrValue])

Returns: string

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

Example:

```
get VField_Name(fldRef,"last")

put VField_Name( fldRef ) into var
```

VField_StorageEncoding(fldRef, [inStrValue])

Returns: string

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

Example:

```
get VField_StorageEncoding( fldRef, "UTF-16" )

put VField_StorageEncoding( fldRef ) into var
```

VField_Table(fldRef) (r/o)

Returns: TableRef

Returns the Table of this field.

Example:

```
put VField_Table( fldRef ) into var
```

VField_Type(fldRef) (r/o)

Returns: FieldTypeRef

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:

```
put Field_Type( fldRef ) into var
```

See also: VField_ChangeType

VField_TypeString(fldRef) (r/o)

Returns: string

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

Example:

```
put Field_TypeString( fldRef ) into var
```

Value Functions

VField_SetBlank(fldRef)

Parameter:	**Description:**
fldRef	The reference of Field object.

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:

```
get VField_SetBlank(fldRef)
```

VField_IsNull(fldRef, [inBoolValue])

Returns: boolean

This is a property of the current record. 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 isNullable! The isNullable is a property of the column of a table, IsNull is a property of the current record.

Example:

```
get VField_IsNull(fldRef,TRUE)

put VField_IsNull( fldRef ) into var
```

VField_Value(fldRef, [inStrValue])

Returns: String with Field Value

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.

Example:

```
get VField_Value( fldRef, 5 )

put VField_Value( fldRef ) into var
```

Search Functions

VField_ValueExists_1(fldRef,
inStrValue,
[inSetRef = nil,
inEnumSearch = "kPreferIndexed"])

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search.
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: boolean

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:

```
put VField_ValueExists(fldRef,"Jo",S) into ValExists
```

VField_ValueExists_2(fldRef,
inStrValue,
[inSetRef = nill,
inEnumSearch = "kPreferIndexed"])

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search.
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: integer

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

Example:

```
put VField_ValueExists(fldRef,"Hn",count,S) into count
```

```
VField_FindValue( fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )
```

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

Returns: BitSetRef

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:

```
put VField_FindValue(fldRef,"Jo") into var
put VField_FindValue(fldRef,"Jo",s1) into var
```

```
VField_FindValueAsArraySet( fldRef,
        inStrValue,
        [inSetRef = nil,
        inIntMaxCount = &hffffffff,
        inEnumSearch = "kPreferIndexed"] )
```

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search
inSetRef	Selection of records.
inIntMaxCount	The maximum number of records to return.
inEnumSearch	Specifies if the search should use index.

Returns: ArraySetRef

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:

```
put VField_FindValueAsArraySet(fldRef,"Jo") into var
put VField_FindValueAsArraySet(fldRef,"Jo",s1) into var
```

```
VField_FindRange( fldRef,
        inBoolLeftInclude,
        inStrLeftValue,
        inStrRightValue,
        inBoolRightInclude,
        [inSetRef = nil],
        [inEnumSearch = "kPreferIndexed"] )
```

Parameter:	**Description:**
fldRef	The reference of Field object.
inBoolleftInclude	TRUE if the left value of the range must be included.
inStrLeftValue	The left value of the range.
inStrRightValue	TRUE if the right value of the range must be included.
inrBoolRightInclude	The right value of the range.
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindRange(fldRef,true ,5,8,true) into var      -- [5,8]
put VField_FindRange(fldRef,false,5,8,true) into var      -- (5,8]
put VField_FindRange(fldRef,true ,5,8,false) into var     -- [5,8)
put VField_FindRange(fldRef,false,5,8,false) into var     -- (5,8)
```

```
VField_FindRangeAsArraySet( fldRef,
        inIntLeftInclude,
        inStrLeftValue,
        inStrRightValue,
        inIntRightInclude,
        [inSetRef = nil,
        inIntMaxCount = &hffffffff,
        inEnumSearch = "kPreferIndexed"] )
```

Parameter:	**Description:**
fldRef	The reference of Field object.
inIntleftInclude	TRUE if the left value of the range must be included.
inStrLeftValue	The left value of the range.
inStrRightValue	TRUE if the right value of the range must be included.
inrIntRightInclude	The right value of the range.
inSetRef	Selection of records.
inIntMaxCount	The maximum number of records to return.
inEnumSearch	Specifies if the search should use index.

Returns: ArraySetRef

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:

```
put VField_FindRangeAsArraySet(fldRef,true ,5,8,true) into var      -- [5,8]
put VField_FindRangeAsArraySet(fldRef,false,5,8,true) into var      -- (5,8]
put VField_FindRangeAsArraySet(fldRef,true ,5,8,false) into var     -- [5,8)
put VField_FindRangeAsArraySet(fldRef,false,5,8,false) into var     -- (5,8)
```

VField_FindSingle(fldRef,
inStrValue,
[inSetRef = nil,
inEnumSearch = "kPreferIndexed"])

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: integer

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:

```
put VField_FindSingle(fldRef,"Jo") into var
```

VField_FindDistinct(fldRef,
[inSetRef = nil,
inEnumSearch = "kPreferIndexed"])

Parameter:	**Description:**
fldRef	The reference of Field object.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindDistinct( fldRef ) into var
```

VField_FindNulls(fldRef,
[inSetRef = nil,
inEnumSearch = "kPreferIndexed"])

Parameter:	**Description:**
fldRef	The reference of Field object.
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindNulls( fldRef ) into var
```

VField_FindNotNulls(fldRef,
[inSetRef = nil,
inEnumSearch = "kPreferIndexed"])

Parameter:	**Description:**
fldRef	The reference of Field object.
inSelection	Selection of records.
inSearchPref	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindNotNulls( fldRef ) into var
```

String Search Functions

The following Functions 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.

```
VField_FindStartsWith( fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )
```

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindStartsWith(fldRef,"Jo" ) into var
```

VField_FindContains(fldRef,
inStrValue,
[inSetRef = nil,
inEnumSearch = "kPreferIndexed"])

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindContains(fldRef,"Jo") into var
```

VField_FindEndsWith(fldRef,
inStrValue,
[inSetRef = nil,
inEnumSearch = "kPreferIndexed"])

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindEndsWith(fldRef,"hn") into var
```

```
VField_FindLike( fldRef,
        inStrValue,
        [inStrEscapeChar = "\",
        inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )
```

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search.
inEscapeChar	The character to be used as escape character.
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindLike(fldRef,"%eter") into var
```

```
VField_FindRegEx( fldRef,
        inStrValue,
        [inSetRef = nil,
        inEnumSearch = "kPreferIndexed"] )
```

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search
inSetRef	Selection of records.
inEnumSearch	Specifies if the search should use index.

Returns: BitSetRef

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:

```
put VField_FindRegEx(fldRef,"Pe?") into var
```

VField Encryption Functions

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.

VDataBase_Encrypt(fldRef, inKey)

Parameter:	**Description:**
fldRef	The reference of Field object.
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:

```
get VDatabase_Encrypt( fldRef, "key12345" )
```

VDataBase_Decrypt(fldRef, inKey)

Parameter:	**Description:**
fldRef	The reference of Field object.
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:

```
get VDatabase_Decrypt( fldRef, "key12345" )
```

VDataBase_ChangeEncryptionKey(fldRef,
inOldKey,
inNewKey)

Параметр:	**Описание:**
fldRef	The reference of Field object.
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:

```
get VDatabase_ChangeEncryptionKey( fldRef, "key12345", "key54321" )
```

VDataBase_RequiresEncryptionKey(fldRef) as Boolean

Parameter:	**Description:**
fldRef	The reference of Field object.

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 Function 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:

```
put VDatabase_RequiresEncryptionKey( fldRef ) to res
```

VDataBase_UseEncryptionKey(fldRef, inKey)

Parameter:	**Description:**
fldRef	The reference of Field object.
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() Function must be called before opening of the database, the VField.UseEncryptionKey() Functions must be called after opening the database and before the first attempt to work with data of the Field.

Example:

```
get VDatabase_UseEncryptionKey( fldRef, "key12345" )
get VDatabase_Open()

get VDatabase_UseEncryptionKey( fldRef, "key12345" )
```

VDate Class

Properties:

VDate_Year(fldRef, [inYear])
VDate_Month(fldRef, [inMonth])
VDate_Day(fldRef, [inDay])

Function

VDate_Set(fldRef, inIntYear, inIntMonth, inIntDay)

VDate Functions

VDate_Set(fldRef,
inIntYear,
inIntMonth,
inIntDay)

Parameter:	**Description:**
fldRef	The reference of Field object.
inIntYear	The Year of a new value.
inIntMonth	The Month of a new value.
inIntDay	The Day of a new value.

Set the value of date field.

Example:

```
get VDate_Set( fldRef, 1972, 3, 20 )
```

VTime Class

Properties:

VTime_Hour(fldRef, [inHour])
VTime_Minute(fldRef, [inMinute])
VTime_Second(fldRef, [inSecond])
VTime_Milisecond(fldRef, [inMilisecond])

Function

VTime_Set(fldRef, inIntHour, inIntMinute, inIntSecond, [inIntMSecond = 0])

VTime Functions

VTime_Set(fldRef,
inIntHour,
inIntMinute,
inIntSecond,
[inIntMSecond])

Parameter:	**Description:**
fldRef	The reference of Field object.
inIntHour	Hours of a new value.
inIntMinute	Minutes of a new value.
inIntSecond	Seconds of a new value.
inIntMSecond	MicroSeconds 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 Function Set() that allows for setting all three properties in one call.

Example:

```
get VTime_Set(fldRef,7,20,0)
```

VDateTime Class

Properties:

VDateTime_Year(fldRef, [inYear])
VDateTime_Month(fldRef, [inMonth])
VDateTime_Day(fldRef, [inDay])

VDateTime_Hour(fldRef, [inHour])
VDateTime_Minute(fldRef, [inMinute])
VDateTime_Second(fldRef, [inSecond])
VDateTime_Milisecond(fldRef, [inMilisecond])

Functions

VDateTime_SetDate(fldRef, inIntYear, inIntMonth, inIntDay)
VDateTime_SetTime(fldRef, inIntHour, inIntMinute, inIntSecond, [inIntMSecond])

VDateTime Functions

VDateTime_SetDate(fldRef,
inIntYear,
inIntMonth,
inIntDay)

Parameter:	**Description:**
fldRef	The reference of Field object.
inIntYear	The Year of a new value.
inIntMonth	The Month of a new value.
inIntDay	The Day of a new value.

Sets the day, month and year.

Example:

```
get VDataTime_SetDate(fldRef,1972,03,20)
```

VDateTime_SetTime(fldRef,
inIntHour,
inIntMinute,
inIntSecond,
[inIntMSecond])

Parameter:	**Description:**
fldRef	The reference of Field object.
inIntHour	Hours of a new value.
inIntMinute	Minutes of a new value.
inIntSecond	Seconds of a new value.
inIntMSecond	MicroSeconds of a new value.

Sets the time of day.

Example:

```
get VDataTime_DataTime.SetTime(fldRef,7,20,00)
```

VString Class
VVarChar Class

Properties

VString_MaxLength(fldRef, [inIntValue]) -- the maximal length of a string which can be stored
VString_IndexByWords(fldRef, inBoolValue) -- if TRUE then each word of the string is indexed separately

Properties

VString_MaxLength(fldRef, [inIntValue])

Returns: 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:

```
get VField_MaxLength(fldRef)

put VField_MaxLength(fldRef)   into var
```

VString_IndexByWords(fldRef, inBoolValue)

Returns: boolean

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

Example:

```
get VField_IndexByWords (fldRef,TRUE)

put VField_IndexByWords(fldRef)   into var
```

Properties Description

VBinary_MaxLength(fldRef, [inIntValue])

Returns: integer

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

Example:

```
get VField_MaxLength(fldRef,120)

put VField_MaxLength(fldRef) into var
```

VBLOB Function

Properties

VBLOB_DataSize(fldRef) (r/o)
VBLOB_IsCompressed(fldRef, [inBoolValue]) -- TRUE if this BLOB field is compressed.
VBLOB_SegmentSize(fldRef) (r/o) -- (in bytes), N * 1024

Functions

VBLOB_DeleteData(fldRef)

VBLOB_ReadRawData(fldRef)
VBLOB_WriteRawData(fldRef, inStrValue)

VBLOB_FromFile(fldRef, inStrLocation)
VBLOB_ToFile(fldRef, inStrLocation)

Properties

VBLOB_DataSize(fldRef) (r/o)

Returns: integer

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

Example:

```
put VBLOB_DataSize(fldRef) into var
```

VBLOB_IsCompressed(fldRef, [inBoolValue])

Returns: boolean

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

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

Example:

```
get VBLOB_IsCompressed(fldRef,TRUE)

put VBLOB_IsCompressed(fldRef) into var
```

VBLOB_SegmentSize(fldRef) (r/o)

Returns: integer

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

Example:

```
put VBLOB_SegmentSize(fldRef) into var
```

Functions

VBLOB_DeleteData(fldRef)

Parameter: **Description:**
fldRef The reference of Field object.

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

Example:

```
get VBLOB_DeleteData(fldRef)
```

VBLOB_ReadRawData(fldRef)

Parameter: **Description:**
fldRef The reference of Field object.

Returns: string

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

Example:

```
put VBLOB_ReadRawData(fldRef,fldRef) into var
```

VBLOB_WriteRawData(fldRef, inStrValue)

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrValue	The value to search.

These Functions allow you to store in the BLOB field any raw data using Revolution String.

The second form of the Functions allow you to randomly access the context of the BLOB field.

Example:

```
get VBLOB_WriteRawData(fldRef,“aaaaaa”)
```

VBLOB_FromFile(fldRef, inStrLocation)

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrLocation	A location of the file.

Read the whole file into the BLOB field.

Example:

```
get VBLOB_FromFile(fldRef,location)
```

VBLOB_ToFile(fldRef, inStrLocation)

Parameter:	**Description:**
fldRef	The reference of Field object.
inStrLocation	A location of the file.

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

Example:

```
get VBLOB_ToFile(fldRef,location)
```

VString Class

Properties

VString_IndexByWords(fldRef, inBoolValue) -- TRUE if indexed by each word of the string

VPicture Class

Properties

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

Functions

```
VPicture_ReadPicture( fldRef )

VPicture_WritePictureAs( fldRef,
        inPict,
        [inEnumPictType = kJPG,
        inQuality = 50] )
```

Functions

VPicture_WritePictureAs(fldRef,
inPict,
[inEnumPictType = kJPG,
inQuality = 50])

Parameter:	Description:
fldRef	The reference of Field object.
inPict	The Picture to be stored.
inEnumPictType	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.

Example:

```
get VPicture_WritePictureAs(fldRef,inPict,"kJPG",50)
```

VPicture_ReadPicture(fldRef)

Parameter:	Description:
fldRef	The reference of Field object.

Returns: PictureRef

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

Example:

```
put VPicture_ReadPicture(fldRef)   into var
```

VObjectPtr Class

Properties

VObjectPtr_OnDelete(fldRef, [inIntValue])
VObjectPtr_Target(fldRef, [tblRef])

Function

VObjectPtr_ConvertFromRDB(fldRef,
inPrimaryKeyRef,
inForeignKeyRef)

Properties Description

VObjectPtr_OnDelete(fldRef, [inIntValue])

Returns: integer

The behavior on deletion of the record-owner.

Example:

```
get VObjectPtr_OnDelete(fldRef)

put VObjectPtr_OnDelete(fldRef)   into var
```

VObjectPtr_Target(fldRef, [tblRef])

Returns: TableRef

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:

```
get VObjectPtr_Target(fldRef)

put VObjectPtr_Target(fldRef)   into var
```

VObjectPtr_ConvertFromRDB(fldRef,
inPrimaryKeyRef,
inForeignKeyRef)

Parameter:	**Description:**
fldRef	The reference of Field object.
inPrimaryKeyRef	The field of the target table that plays role of the PRIMARY KEY field.
inForeignKeyRef	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:

```
get VObjectPtr_ConvertFromRDB(fldRef,fldPersonID,fldPersonPtr)
```

VCursor Class

Properties

```
VCursor_BOF( tblRef ) (r/o)
VCursor_EOF( tblRef ) (r/o)
VCursor_Database( cursorRef )  (r/o)          --  (r/o) Database of this Cursor.
VCursor_FieldCount( cursorRef )               --  (r/o) number of selected fields for this Cursor.
VCursor_Position( cursorRef, [inIntValue] )
VCursor_RecordCount( cursorRef, [inIntValue] )  --  Number of selected records, it can be reduced.
VCursor_ReadOnly( cursorRef)  ( r/o )         --  (r/o) TRUE if records can't be changed
                                              --  i.e. you can't add/update/delete records.
```

Construction

```
VCursor_Destructor( cursorRef )
```

Field Functions

```
VCursor_Field( cursorRef, inIntIndexOrStrName )
VCursor_FieldValue( cursorRef, inIntIndexOrStrName, [inStrValue]  )
```

Navigation Functions

```
VCursor_FirstRecord( cursorRef )
VCursor_LastRecord( cursorRef )
VCursor_PrevRecord( cursorRef )
VCursor_NextRecord( cursorRef )
```

Record Functions

```
                                              -- blank the memory buffer of the record
VCursor_SetBlank( cursorRef, [inEnumAccess = "forUpdate"] )

VCursor_AddRecord( cursorRef )                -- adds a new record to a cursor
VCursor_UpdateRecord( cursorRef )             -- updates the current records of the cursor
VCursor_UpdateAllRecords( cursorRef )         -- updates ALL records of the cursor with a new value.
VCursor_DeleteRecord( cursorRef )             -- deletes the current record of the cursor
VCursor_DeleteAllRecords( cursorRef )         -- deletes all records of the cursor

VCursor_DropRecord( cursorRef )               -- removes the current record from cursor
                                              -- but don't delete it from the original Table.

                                              -- returns a single record of the cursor as string
VCursor_GetRecord( cursorRef, [RecIndex], [fldDelimiter] )

                                              -- returns few/all  records of the cursor as string
VCursor_GetRecords( cursorRef, [FromRec], [MaxRecords], [fldDelimiter], [recDelimiter] )
```

Import/export Functions

```
VCursor_ImportText( cursorRef,
        inStrPath,
        [inStrFldDelimiter = chr(09),
        inStrLineDelimiter = LE,
        inStrEncoding = ”UTF-16",
        inIntHasColumHeader = FALSE,
        inIntMaxRecordsToImport = 0 )

VCursor_ExportText( cursorRef,
        inStrPath,
        [inStrFldDelimiter = chr(09),
        instrLineDelimiter = LE,
        inStrEncoding =”UTF-16",
        inIntHasColumHeader = FALSE] )
```

Conversion Functions

```
VCursor_ToArraySet()
```

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 Table, each of which points to a different record.

See description of methods VDatabase_SqlSelect() and VDatabase_SqlQuery() for example of usage. These methods create a new cursor. You should destroy cursor as soon as posible when you do not need it anymore using VCursor_Destructor().

Properties

VCursor_BOF(cursorRef) (r/o)

Returns: boolean

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

Note: This property provides a way used to ODBC API.

Example:

```
put VCursor_BOF( cursorRef ) into res
```

VCursor_Database(cursorRef) (r/o)

Returns: DataBaseRef

Returns the database of this Cursor.

Example:

```
put VCursor_Database( cursorRef ) into var
```

VCursor_EOF(cursorRef) (r/o)

Returns: boolean

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

Note: This property provides a way used to ODBC API.

Example:

```
put VTable_EOF( cursorRef ) into res
```

VCursor_FieldCount(cursorRef)

Returns: integer

Returns the number of fields of this cursor.

Example:

```
put VCursor_FieldCount( cursorRef ) into VAR
```

VCursor_Position(cursorRef, [inIntValue])

Returns: 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 RecordCount.

When you assign a new value to the CurrentPosition, 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:

```
get VCursor_Postion(cursorRef,5)
put VCursor_Postion(cursorRef)   into var
```

Example:

```
put VDatabase_SqlSelect( query ) into cursorRef
put VCursor_RecordCount( cursorRef ) into recCount
repeat for i = 1 to recCount
        VCursor_Position( cursorRef, i )

        -- read fields of cursor or other job...
end repeat
```

VCursor_RecordCount(cursorRef, [inIntValue])

Returns: integer

Returns the number of records of cursor.

Example:

```
--   store into a local variable to avoid of calling it loop
get  VCursor_RecordCount(cursorRef,2)

put  VCursor_RecordCount(cursorRef)   into var
```

VCursor_ReadOnly(cursorRef) (r/o)

Returns: boolean

Returns TRUE if the Cursor is read only, otherwise returns FALSE.

Example:

```
put VCursor_ReadOnly(cursorRef)   into var
```

Field Functions

VCursor_Field(cursorRef, inIntIndexOrStrName)

Parameter:	Description:
cursorRef	The reference of Cursor object.
inIntIndexOrStrName	The index of a Field in a Cursor (starts from 1) or name of a Field.

Returns: FieldRef

You can use these functions 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:

```
put VCursor_Field(cursorRef,1)   into fieldRef
```

Example:

```
put VCursor_Field(cursorRef,"Jo")   into fieldRef
```

VCursor_FieldValue(cursorRef, inIntIndexOrStrName, [inStrValue])

Returns: string

Parameter:	Description:
cursorRef	The reference of Cursor object.
inIntIndexOrStrName	The index of a Field in a Cursor (starts from 1) or name of a Field.
inStrValue	Value of the field.

The form of this function with two parameters returns a value of cursor field.
The form of it with three parameters can assign a new value to cursor field.
This function allows to you write one line of code instead of two.

The order of fields in the cursor is the same as the order of fields in the SELECT statement of the query.

Example:

```
put VCursor_FieldValue( cursorRef, 1 ) into value
```

Example:

```
get VCursor_FieldValue( cursorRef, 1, "Name" )
```

Navigation Functions

VCursor_FirstRecord(cursorRef)

Parameter: **Description:**
cursorRef The reference of Cursor object.

Returns: boolean

Go to the first logical record of a Cursor. Returns TRUE if the first record is found.

Example:

```
put  VCursor_FirstRecord(cursorRef)   into var
```

VCursor_LastRecord(cursorRef)

Parameter: **Description:**
cursorRef The reference of Cursor object.

Returns: boolean

Go to the last record of a Cursor. Returns TRUE if the last record is found

Example:

```
put  VCursor_LastRecord(cursorRef)   into var
```

VCursor_PrevRecord(cursorRef)

Parameter: **Description:**
cursorRef The reference of Cursor object.

Returns: 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:

```
put  VCursor_PrevRecord(cursorRef)   into var
```

VCursor_NextRecord(cursorRef)

Parameter:	**Description:**
cursorRef	The reference of Cursor object.

Returns: 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:

```
put VCursor_NextRecord(cursorRef) into BOOLVAR
```

Example:

```
function getListFromValCursor( cursRef )

    -- We jump to first record of cursor, if it exists then we have success.
    if VCursor_FirstRecord( cursRef ) then
        repeat forever
                -- do some job with the current record of cursor

                if VCursor_NextRecord( cursRef )  is false then
                        exit repeat
        end repeat
    end if

end getListFromValCursor2
```

Record Functions

VCursor_SetBlank(cursorRef, [inEnumAccess = "forUpdate"])

Parameter:	Description:
cursorRef	The reference of Cursor object.
inEnumAccess	Optimisation parameter.

Each Cursor has a RAM buffer for field values of the current record. This buffer can be cleared by the SetBlank() Function, 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.

Example:

```
get VCursor_SetBlank(cursorRef)
```

VCursor_AddRecord(cursorRef)

Parameter:	Description:
cursorRef	The reference of Cursor object.

Returns: integer

Adds a new record to the Cursor with the current field values in the RAM buffer.

Returns FALSE if the record cannot be added, e.g. cursor is ReadOnly.

Example:

```
put VCursor_AddRecord(cursorRef) into VAR
```

VCursor_UpdateRecord(cursorRef)

Parameter:	**Description:**
cursorRef	The reference of Cursor object.

Updates the current record of a Cursor with the values in the RAM buffer.

Example:

```
get VCursor_UpdateRecord(cursorRef)
```

VCursor_UpdateAllRecords(cursorRef)

Parameter:	**Description:**
cursorRef	The reference of Cursor object.

Returns: boolean

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 then an iteration of the cursor records in a loop to assign new values.

Returns FALSE if the records cannot be updated, e.g. cursor is ReadOnly.

Example:

```
get VCursor_UpdateAllRecords(cursorRef)
```

VCursor_DeleteRecord(cursorRef)

Parameter:	**Description:**
cursorRef	The reference of Cursor object.

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:

```
get  VCursor_DeleteRecord(cursorRef)
```

VCursor_DeleteAllRecords(cursorRef)

Parameter:	**Description:**
cursorRef	The reference of Cursor object.

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:

```
get  VCursor_DeleteAllRecords(cursorRef)
```

VCursor_DropRecord(cursorRef)

Parameter:	**Description:**
cursorRef	The reference of Cursor object.

Removes the current record from a Cursor, but does not delete it from the original Table.

Example:

```
get  VCursor_DropRecord(cursorRef)
```

VCursor_GetRecord(cursorRef, [RecIndex], [fldDelimiter])

Parameter:	**Description:**
cursorRef	The reference of Cursor object.
RecIndex	The index of a record in cursor. From 1 to N.
fldDelimiter	A character to be used as a field delimiter. Default '\t'

LiveCode users often are used to work with strings. On demand of users we have added this function to Livecode ADK. It is not a standard function of Valentina DB API.

This function allows you to get all fields of Nth record of the cursor as a string, where fields are seprated by a char-delimiter. On default this is '\t' - TAB symbol.

If RecIndex is not specified, then the current record of the cursor is used.

Example:

```
put VCursor_SqlSelect( "SELECT * FROM T" ) into CursorRef
put VCursor_GetRecord( cursorRef, 1 ) into recStr
get VCursor_Destructor( cursorRef )
```

VCursor_GetRecords(cursorRef, [FromRec], [MaxRecords], [fldDelimiter], [recDelimiter])

Parameter:	**Description:**
cursorRef	The reference of Cursor object.
FromRec	First record to use. Default 1.
MaxRecords	How much records to use. Default ULONG_MAX.
fldDelimiter	A character to be used as a field delimiter. Default '\t'
recDelimiter	A character to be used as a record delimiter. Default '\r'

LiveCode users often are used to work with strings. On demand of users we have added this function to Livecode ADK. It is not a standard function of Valentina DB API.

This function is similar to VCursor_GetRecord(), but it can return few or even all records of cursor.

Therefore, we get recDelimiter parameter to specify a character to be record delimiter. Also you can specify any range of records, for example, get 5 records from the first.

Example:

```
put VCursor_SqlSelect( "SELECT * FROM T" ) into CursorRef
put  VCursor_DeleteAllRecords(cursorRef)  into allRecordsStr
get VCursor_Destructor( cursorRef )
```

Import/Export Functions

```
VCursor_ImportText( cursorRef,
        inStrPath,
        [inStrFldDelimiter = numtochar(09),
        inStrLineDelimiter = LE,
        inStrEncoding = ”UTF-16",
        inIntHasColumHeader = FALSE,
        inIntMaxRecordsToImport = 0 )
```

Parameter:	Description:
cursorRef	The reference of Cursor object.
inStrPath	File to be imported.
inStrFldDelimiter	Character to be used as a field delimeter, default is a tab-chr(0x09).
inStrLineDelimiter	Character to be used as a record delimeter, default is the OS Line Ending.
inStrEncoding	Encoding of the imported file.
inIntHasColumnHeader	TRUE if the import file has a column header line.
inIntMaxRecordsToImport	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:

```
get  VCursor_ImportText(
        cursorRef,fileToImport,numtochar(09),numtochar(13))
```

```
VCursor_ExportText( cursorRef,
        inStrPath,
        [inStrFldDelimiter = numtochar(09),
        instrLineDelimiter = LE,
        inStrEncoding =”UTF-16",
        inIntHasColumHeader = FALSE] )
```

Parameter:	**Description:**
cursorRef	The reference of Cursor object.
inStrPath	The file to be imported.
inStrFldDelimiter	The character to be used as a field delimeter, default is TAB chr(0x09).
inStrLineDelimiter	The character to be used as a record delimeter, default is the OS Line Ending.
inStrEncoding	Encoding of the imported file.
inIntHasColumHeader	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:

```
get  VCursor_ExportText(
        cursorRef,fileToExport,numtochar(09),numtochar(13))
```

VCursor_ToArraySet()

Returns: ArraySetRef

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:

```
put VDatabase_SqlSelect( dbRef, "SELECT RecID FROM T WHERE ..." ) into cursRef

put VCursor_ToArraySet( cursRef ) into arraySetRef

VCursor_Destructor( cursRef )  // we do not need cursor any more.
```

VSet Class

Properties

VSet_Count(setRef) (r/o)
VSet_IsSortedByRecID(setRef, inBoolValue)
VSet_IsEmpty(setRef) (r/o)

Construction

VSet_Clone(setRef)

VSet_Destructor(setRef)

Element Functions

VSet_Append(setRef, inIntValue)
VSet_Remove(setRef, inIntValue)
VSet_Include(setRef, inIntValue)

VSet_MakeNewIterator(setRef)
VSet_SortByRecID(setRef)

Set operations

VSet_Union(setRef1, setRef2)
VSet_Intersection(setRef1, setRef2)
VSet_Difference(setRef1, setRef2)
VSet_SymmetricDifference(setRef1, setRef2)

Constructor

Note, you cannot create instances of just VSet class. Instead you should create VArraySet or VBitSet instance.

VSet_Clone(setRef)

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:

```
put VSet_Clone( setRef ) into NewSetRef
```

VSet_Destructor(setRef)

Destroys given VSet object.

Example:

```
put VSet_Destructor( setRef ) into setRef
```

Properties

VSet_Count(setRef) (r/o)

Returns: integer

The number of items in the Set.

Example:

```
put VSet_Count(setRef)   into var
```

VSet_IsSortedByRecID(setRef, inBoolValue)

Returns: boolean

Returns TRUE if the Set is sorted by RecID values.

Example:

```
get VSet_isSortByRecID(setRef,set1)

put VSet_isSortByRecID(setRef)   into var
```

VSet_IsEmpty(setRef) (r/o)

Returns: boolean

Returns TRUE if the Set is empty.

Example:

```
put VSet_IsEmply(setRef)
```

Element Functions

VSet_Append(setRef, inIntValue)

Parameter:	Description:
setRef	The reference of Set object.
inIntValue	A value.

Appends a new value to the Set.

Example:

```
get Set_Append(setRef,rec)
```

VSet_Remove(setRef, inIntValue)

Parameter:	Description:
setRef	The reference of Set object.
inIntValue	A value.

Removes the specified value from the Set.

Example:

```
get Set_Remove(setRef,rec)
```

VSet_Include(setRef, inIntValue)

Parameter:	Description:
setRef	The reference of Set object.
inIntValue	A value.

Returns: boolean

Returns TRUE if the Set contains the specified value.

Example:

```
put Set_Include(setRef,rec)   into var
```

VSet_MakeNewIterator(setRef)

Parameter:	**Description:**
setRef	The reference of Set object.

Returns: SetIteratorRef

Creates and returns a new Iterator for this Set.

Example:

```
put VSet_MakeNewIterator(setRef)
```

VSet_SortByRecID(setRef)

Parameter:	**Description:**
setRef	The reference of Set object.

Sorts the Set.

Example:

```
get VSet_SortByRecID(setRef)
```

Set Functions

VSet_Union(setRef, inRightSet)

Parameter:	Description:
setRef	The reference of Set object.
inRightSet	The set to be used in the Function.

Returns: setRef

Executes a union of this set with the inRightSet set. The result becomes this set. Such an Function is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
put VSet_Union( setRef, set2Ref ) into resSet
```

VSet_Intersection(setRef, inRightSet)

Parameter:	Description:
setRef	The reference of Set object.
inRightSet	The set to be used in the Function.

Returns: setRef

Executes an Intersection of this set with the inRightSet. The result becomes this set. Such an Function is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
put VSet_Intersection( setRef, set2Ref ) into resSet
```

VSet_Difference(setRef, inRightSet)

Parameter:	**Description:**
setRef	The reference of Set object.
inRightSet	The set to be used in the Function.

Returns: setRef

Executes the difference of this set with the inRightSet. The result becomes this set. Such an Function is said to be be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
put VSet_Difference( setRef, set2Ref ) into resSet
```

VSet_SymmetricDifference(setRef, inRightSet)

Parameter:	**Description:**
setRef	The reference of Set object.
inRightSet	The set to be used in the Function.

Returns: setRef

Executes the SymmetricDifference of this set with the inRightSet. The result becomes this set. Such Function is said to be "in place".

Note: Both sets must be of the same type (BitSet or ArraySet).

Example:

```
put VSet_SymmetricDifference( setRef, set2Ref ) into resSet
```

VArraySet Class

Constructor

VArraySet_WithCount(inCount as Integer)
VArraySet_FromArraySet(inArraySet as VArraySet)
VArraySet_FromBitSet(inBitSet as VBitSet)

Functions

VArraySet_ItemAt(inPosition as Integer) as Integer
VArraySet_ItemAt(inPosition as Integer, Assigns inValue as Integer)

Constructor

VArraySet_WithCount(inCount)

Parameter: **Description:**
inCount The initial size of ArraySet.

Returns: setRef

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:

```
put VArraySet_WithCount(50) into arraySetRef
```

VArraySet_FromArraySet(inArraySet)

Parameter: **Description:**
inArraySet Another ArraySet

Returns: setRef

Copy constructor. Creates a new ArraySet from the given inArraySet. The new ArraySet is an exact copy of the inArraySet.

Example:

```
put VArraySet_FromArraySet( as1Ref ) into as2Ref
```

VArraySet_FromBitSet(inBitSet)

Parameter: **Description:**
inBitSet The BitSet.

Returns: setRef

Constructor. Creates a new ArraySet from the given inBitSet. The ArraySet contains the same items as inBitSet.

Example:

```
put VArraySet_FromBitSet( bitSet1Ref ) into arraySetRef
```

Functions

VArraySet_ItemAt(setRef, inPosition)

Parameter:	Description:
setRef	The reference of Set object.
inPosition	Position of item in the array set.

Returns: integer

Returns the item of the set at the specified position.

Example:

```
put VArraySet_ItemAt( setRef, 5 ) into nextRecID
```

VArraySet_ItemAt(setRef, inPosition, Assigns inValue)

Parameter:	Description:
setRef	The reference of Set object.
inPosition	Position of item in the array set.
inValue	A value.

Assigns inValue to the item of the set at the specified position.

Example:

```
get VArraySet_ItemAt( setRef, 5, recID )
```

VBitSet Class

Constructor

VBitSet_WithCount(inMaxCount)
VBitSet_FromArraySet(inMaxCount, inArraySet)

Constructor

VBitSet_WithCount(inMaxCount)

Parameter:	Description:
inMaxCount	The maximum value that can be stored in the bitset.

Returns: BitSetRef

Constructor. Creates a BitSet of the specified size.

If you want create BitSet that corresponds to a Table, you need use physicalRecordCount of that table as inMaxCount.

Example:

```
put VBitSet_New( 50 ) into bitSetRef
```

VBitSet_FromArraySet(inMaxCount, inArraySet)

Parameter:	Description:
inMaxCount	The maximal value that can be stored in the bitset.
inArraySet	The ArraySet.

Returns: BitSetRef

Constructor. Creates a new BitSet from the given inArraySet. The BitSet contains the same items as inArraySet.

If you want create BitSet that corresponds to a Table, you need use physicalRecordCount of that table as inMaxCount.

Example:

```
put VBitSet_FromArraySet( as1 ) into bitSetRef
```

VSetIterator Class

Properties

VSetIterator_Value(setIteratorRef) (r\o)

Destructor

SetIterator_Destructor(setIteratorRef)

Item Functions

VSetIterator_FirstItem(setIteratorRef)
VSetIterator_LastItem(setIteratorRef)
VSetIterator_NextItem(setIteratorRef)
VSetIterator_PrevItem(setIteratorRef)

Properties

VSetIterator_Value(setIteratorRef) (r\o)

Returns: integer

Returns the current value of the iterator.

Example:

```
put VSetIterator_Value(setRef)   into var
```

Destructor

VSetIterator_Destructor(setIteratorRef)

Parameter:	Description:
setIteratorRef	The reference of SetIterator object.

Returns: VSetIteratorRef

Destroyes object of SetIterator.

Example:

```
put VSetIterator_Destructor(setIteratorRef)   into setIteratorRef
```

VSetIterator Functions

VSetIterator_FirstItem(setIteratorRef)

Parameter:	**Description:**
setIteratorRef	The reference of SetIterator object.

Returns: 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:

```
put VSetIterator_FirstItem(setRef)   into var
```

VSetIterator_LastItem(setIteratorRef)

Parameter:	**Description:**
setIteratorRef	The reference of SetIterator object.

Returns: 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:

```
put VSetIterator_LastItem(setRef)   into var
```

VSetIterator_NextItem(setIteratorRef)

Parameter:	**Description:**
setIteratorRef	The reference of SetIterator object.

Returns: 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:

```
put VSetIterator_NextItem(setRef)   into var
```

VSetIterator_PrevItem(setIteratorRef)

Parameter:	**Description:**
setIteratorRef	The reference of SetIterator object.

Returns: 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:

```
put VSetIterator_PrevItem(setRef)   into var
```

VLink Class

Properties

VLink_BranchCount(lnkRef) (r/o)
VLink_ID(lnkRef) (r/o)
VLink_IsTemporary(lnkRef) (r/o)
VLink_Name(lnkRef, [inStrValue])
VLink_OnDelete(lnkRef, [inEnumOnDelete])
VLink_OnUpdate(lnkRef, [inEnumOnUpdate])
VLink_Owner(lnkRef, [tblRef])

Table Functions

VLink_IsBetween(lnkRef, inTblRefA, inTblrefB)

VLink_Table(lnkRef, inIntIndex)

VLink_Flush(lnkRef, inBoolFlushTables)

Search Functions

```
VLink_FindLinked( lnkRef,
        inRecID,
        inTblARef,
        inTblBRef,
        [inEnumRecursionDirection =  "kFromParentToChild"] )

VLink_FindLinkedToSet( lnkRef,
        inSetRef,
        inTblARef,
        inTblBRef,
        [inEnumRecursionDirection =  "kFromParentToChild"] )

VLink_FindExclusivelyLinked( lnkRef,
        inRecID,
        inTblARef,
        inTblBRef,
        [inEnumRecursionDirection =  "kFromParentToChild"] )

VLink_FindAllLinked( lnkRef,
        inTblARef,
        inTblBRef,
        [inEnumRecursionDirection =  "kFromParentToChild"] )
```

Linking Functions

```
VLink_CountLinked( lnkRef,
        inRecID,
        inTblARef,
        inTblBRef,
        [inEnumRecursionDirection =  "kFromParentToChild"] )

VLink_LinkRecords( lnkRef, inLeftRecID, inRightRecID )

VLink_UnlinkRecords( lnkRef, inLeftRecID, inRightRecID )

VLink_DeleteLinkedRecords( lnkRef,
        inRecID,
        inTblARef,
        [inEnumRecursionDirection =  "kFromParentToChild"] )

VLink_DeleteAllLinkedRecords( lnkRef,
        inTblARef,
        [inEnumRecursionDirection =  "kFromParentToChild"] )

VLink_IsLinked( lnkRef, inLeftRecID, inRightRecID )
```

Properties

VLink_BranchCount(lnkRef) (r/o)

Returns: integer

Returns the number of branches for this link.

Example:

```
put VLink_BranchCount( lnkRef ) into var
```

VLink_ID(lnkRef) (r/o)

Returns: integer

Returns the ID of this link. A temporary link has a negative ID.

Example:

```
put VLink_ID( lnkRef ) into var
```

VLink_IsTemporary(lnkRef) (r/o)

Returns: boolean

Returns TRUE if this link is temporary.

Example:

```
put VLink_IsTemporary ( lnkRef ) into var
```

VLink_Name(lnkRef, [inStrValue])

Returns: string

Returns the name of the link.

Example:

```
get VLink_Name(lnkRef,"s")
put VLink_Name( lnkRef ) into var
```

VLink_OnDelete(lnkRef, [inEnumOnDelete])

Returns: OnDeletionRef

The behavior on deletion of the record-owner.

Example:

```
get VLink_OnDelete(lnkRef,"kCaskade")

put VLink_OnDelete( lnkRef ) into var
```

VLink_OnUpdate(lnkRef, [inEnumOnUpdate])

Returns: OnUpdateRef

The behavior on update of the record-owner.

Example:

```
get VLink_OnUpdate(lnkRef,"kCaskade")

put VLink_OnUpdate( lnkRef ) into var
```

VLink_Owner(lnkRef, [tblRef])

Returns: tblRef

The table which is owner the link. For symmetric links 1:1 and 1:M Valentina cannot define which of tables will be owner of the link. You can use this property to define the owner.

Example:

```
get VLink_Owner( lnkRef )

put VLink_Owner( lnkRef ) into tblRef
```

Table Functions

VLink_IsBetween(lnkRef,
inTblRefA,
inTblrefB)

Parameter:	Description:
lnkRef	The reference of Link object.
inTblRefA	Left table of link.
inTblRefB	Right table of link.

Returns: boolean

Returns TRUE if this Link links both specified Tables.

Example:

```
put VLink_IsBetween(lnkRef,TablA,TablB) into var
```

VLink_Table(lnkRef, inIntIndex)

Parameter:	Description:
lnkRef	The reference of Link object.
inIntIndex	The index of table.

Returns: TblRef

Returns a table of the link by index.

Example:

```
put VLink_Link.Table(lnkRef,i) into var
```

VLink_Flush(lnkRef, inBoolFlushTables)

Parameter:	Description:
lnkRef	The reference of Link object.
inBoolFlushTables	TRUE if Tables of Link also should flush.

Returns: boolean

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:

```
get VLink_Flush(lnkRef,TRUE)
```

Search Functions

```
VLink_FindLinked( lnkRef,
        inRecID,
        inTblARef,
        inTblBRef,
        [inEnumRecursionDirection =  "kFromParentToChild"] )
```

Parameter:	Description:
lnkRef	The reference of Link object.
inRecID	The RecID of a record of the left table.
inTblARef	Left table of link.
inTblBRef	Right table of link.
inEnumRecursionDirection	The direction of movement for a recursive link.

Returns: ArraySetRef

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:

```
put VLink_FindLinked(lnkRef,rec,TblA,TblB) into var
```

VLink_FindLinkedToSet(lnkRef,
inSetRef,
inTblARef,
inTblBRef,
[inEnumRecursionDirection =kFromParentToChild])

Parameter:	**Description:**
lnkRef	The reference of Link object.
inSetRef	Selection of records.
inTblARef	Left table of link.
inTblBRef	Right table of link.
inEnumRecursionDirection	The direction of movement for a recursive link.

Returns: BitSetRef

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:

```
put VLink_FindLinked(lnkRef,rec,TblA,TblB) into var
```

VLink_FindExclusivelyLinked(lnkRef,
inRecID,
inTblARef,
inTblBRef,
[inEnumRecursionDirection = "kFromParentToChild"])

Parameter:	**Description:**
lnkRef	The reference of Link object.
inRecID	The RecID of a record of the left table.
inTblARef	Left table of link.
inTblBRef	Right table of link.
inEnumRecursionDirection	The direction of movement for a recursive link.

Returns: ArraySetRef

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:

```
put VLink_FindExclusivelyLinked(lnkRef,rec,TblA,TblB) into var
```

VLink_FindAllLinked(lnkRef,
inTblARef,
inTblBRef,
[inEnumRecursionDirection = "kFromParentToChild"])

Parameter:	**Description:**
lnkRef	The reference of Link object.
inTblARef	Left table of link.
inTblBRef	Right table of link.
inEnumRecursionDirection	The direction of movement for a recursive link.

Returns: BitSetRef

Returns all records of inTableB linked to any record of inTableA.If zero records are found then returns NIL.

Example:

```
put VLink_FindAllLinked(lnkRef,TblA,TblB) into var
```

Linking Functions

VLink_CountLinked(lnkRef,
inRecID,
inTblARef,
inTblBRef,
[inEnumRecursionDirection = "kFromParentToChild"])

Parameter:	Description:
lnkRef	The reference of Link object.
inRecID	The RecID of a record of the left table.
inTblARef	Left table of link.
inTblBRef	Right table of link.
inEnumRecursionDirection	The direction of movement for a recursive link.

Returns: Integer

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:

```
put VLink_CountLinked( lnkRef, rec, TblA, TblB ) into var
```

VLink_LinkRecords(lnkRef, inLeftRecID, inRightRecID)

Parameter:	Description:
lnkRef	The reference of Link object.
inLeftRecID	The RecID of a record of the left table.
inRightRecID	The RecID of a record of the right 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:

```
get VLink_LinkRecords( lnkRef, Tbl_L, Tbl_R )
```

VLink_UnlinkRecords(lnkRef, inLeftRecID, inRightRecID)

Parameter:	**Description:**
lnkRef	The reference of Link object.
inLeftRecID	The RecID of a record of the left table.
inRightRecID	The RecID of a record of the right 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:

```
get VLink_UnlinkRecords( lnkRef, Tbl_L, Tbl_R )
```

VLink_DeleteLinkedRecords(lnkRef,
inRecID,
inTblARef,
[inEnumRecursionDirection = "kFromParentToChild"])

Parameter:	**Description:**
lnkRef	The reference of Link object.
inRecID	The RecID of a record of the left table.
inTblARef	Left table of link.
inEnumRecursionDirection	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:

```
get VLink_DeleteLinkedRecords( lnkRef, rec, TblA )
```

VLink_DeleteAllLinkedRecords(lnkRef,
inTblARef,
[inEnumRecursionDirection = "kFromParentToChild"])

Parameter:	**Description:**
lnkRef	The reference of Link object.
inTblARef	Left table of link.
inEnumRecursionDirection	The direction of movement for a recursive 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:

```
get VLink_DeleteAllLinkedRecords( lnkRef, TblA )
```

VLink_IsLinked(lnkRef, inLeftRecID, inRightRecID)

Parameter:	**Description:**
lnkRef	The reference of Link object.
inLeftRecID	The RecID of a record of the left table.
inRightRecID	The RecID of a record of the right table.

Returns: boolean

Returns TRUE, if the two specified records are linked.

Example:

```
put VLink_IsLinked( lnkRef, RecID_L, RecID_R ) into res
```

VLink2 Class

Properties

VLink_LeftType(lnkRef) (r/o) -- returns the type of link for LEFT table.
VLink_RightType(lnkRef) (r/o) -- returns the type of link for RIGHT table.

Properties Description

VLink_LeftType(lnkRef) (r/o)

Returns: LinkTypeRef

Returns the relation type for the left branch. Can be kOne or kMany.

Example:

```
put VLink_LeftType( lnkRef ) into var
```

VLink_RightType(lnkRef) (r/o)

Returns: LinkTypeRef

Returns the relation type for the right branch. Can be KOne or kMany.

Example:

```
put VLink_RightType( lnkRef ) into var
```

VServer Class

To be used only with Valentina Server.

Properties

```
VServer_Available( svrRef ) (r/o)          --  (r/o) Returns TRUE if the server is available.
VServer_ConnectionCount( svrRef ) (r/o)    --  (r/o) The number of active connections to a server.
VServer_DatabaseCount( svrRef ) (r/o)      --  (r/o) The number of databases that the server recognizes.
VServer_HostName( svrRef ) (r/o)           --  (r/o) The name of the host where the server is located.
VServer_Port( svrRef ) (r/o)               --  (r/o) Returns the port number of the server host.
VServer_UserName( svrRef ) (r/o)           --  (r/o) The name of the current user.
VServer_UserCount( svrRef ) (r/o)          --  (r/o) Returns the count of registered users.
VServer_Version( svrRef ) (r/o)            --  (r/o) Version of the server.
```

INI-File Property

```
VServer_Variable( svrRef, inStrName )
VServer_Variable( svrRef, inStrName, inNewValue )
```

Function

```
VServer_New(
        inHost,
        inUserName,
        inUserPassword,
        [inPort = 15432],
        [inTimeOut = 5],
        [inOptions =""] )

VServer_Destructor()
```

Connection Functions

```
VServer_OpenSession( svrRef )
VServer_CloseSession( svrRef )
VServer_CancelConnection( svrRef, inConnectionID )

VServer_Restart( svrRef )
VServer_Refresh( svrRef )
VServer_Shutdown( svrRef )
```

Master databases Functions

```
VServer_RegisterDatabase   ( svrRef, inDbName, inServerFullPath )
VServer_UnregisterDatabase( svrRef, inDbName )
```

User Functions

VServer_AddUser (svrRef, inUserName, inStrPassword, [isAdmin = FALSE])
VServer_RemoveUser(svrRef, inUserName)

VServer_ChangeUserPassword(svrRef, inUserName, inNewPassword)

VServer_UserName(svrRef, inUserIndex)
VServer_UserIsAdmin(svrRef, inUserIndex)

DatabaseInfo Functions

VServer_DatabaseInfo(svrRef, inIndex)

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

VServer_Available(svrRef) (r/o)

Returns: integer

Returns TRUE if the server is available.

Example:

```
put VServer_Available(svrRef) into var
```

VServer_ConnectionCount(svrRef) (r/o)

Returns: integer

Returns the number of all active connections to the server.

Example:

```
put VServer_ConnectionCount(svrRef)   into var
```

VServer_DatabaseCount(svrRef) (r/o)

Returns: integer

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:

```
put VServer_DatabaseCount(svrRef)   into var
```

VServer_HostName(svrRef) (r/o)

Returns: string

Returns a string that contains the name of the server host.

Example:

```
put VServer_HostName(svrRef)   into var
```

VServer_Port(svrRef) (r/o)

Returns: integer

Returns the port number of the server host.

Example:

```
put VServer_Port(svrRef)   into var
```

VServer_UserName(svrRef) (r/o)

Returns: string

Returns user name of this connection (i.e. administrator self).

Note: this is the same name that was used to connect to the Server.

Example:

```
put VServer_UserName(svrRef)   into var
```

VServer_UserCount(svrRef) (r/o)

Returns: integer

Returns the number of registered users.

Example:

```
put VServer_UserCount(svrRef)   into var
```

VServer_Version(svrRef) (r/o)

Returns: string

Returns a string that contains the VServer version number.

Example:

```
put VServer_Version(svrRef)   into var
```

Creation of VServer

```
VServer_New(
        inHost,
        inUserName,
        inUserPassword,
        [inPort = 15432],
        [inTimeOut = 5],
        [inOptions = ""] )
```

Returns: srvRef

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 Function constructs a VServer object. This constructor simply stores parameters and does not try connect. The real connection occurs using OpenSession().

Note: Only Administrator User(s) can use this object.

If a VServer uses the default port, then the port number is optional when clients make a connection to the VServer. Unique port values must be used to allow more than one Vserver to be accessed on a Host.

Example:

```
put VServer_New( "localhost", "sa", "sa" ) into srvRef
```

```
VServer_Destructor( svrRef )
```

Returns: zero

Parameter:	**Description:**
srvRef	The reference of Server object.

Destroys VServer object.

Example:

```
put VServer_VServer( svrRef ) int srvRef
```

Connection Functions

VServer_OpenSession(svrRef)

Parameter:	Description:
srvRef	The reference of Server object.

Establishes a connection to the server.

Errors: Wrong user name,
Wrong password,
the user is not an administrator,
connection cannot be established.

Example:

```
get VServer_VServer( "localhost", "sa", "sa" )
get VServer_OpenSession( svrRef )
```

VServer_CloseSession(svrRef)

Parameter:	Description:
srvRef	The reference of Server object.

Closes the connection with the server .

Example:

```
get VServer_VServer( "localhost", "sa", "sa" )
get VServer_OpenSession
...
get VServer_CloseSession( svrRef )
```

VServer_CancelConnection(svrRef, inConnectionID)

Parameter:	Description:
srvRef	The reference of Server object.
inConnectionID	The connection ID.

Cancels an existing connection by its ID.

Example:

```
get VServer_CancelConnection( svrRef, connID )
```

VServer_Restart(svrRef)

Parameter:	**Description:**
srvRef	The reference of Server object.

Forces a restart of the VServer.

Example:

```
get VServer_Restart( svrRef )
```

VServer_Refresh(svrRef)

Parameter:	**Description:**
srvRef	The reference of Server object.

This Function allows you to refresh the list of DatabaseInfo objects. This Function sends a request to the Valentina Server.

Example:

```
get VServer_Refresh( svrRef )
```

VServer_Shutdown(svrRef)

Parameter:	**Description:**
srvRef	The reference of Server object.

Shuts down the VServer.

Note: After this Function there is no way to restart VServer from the application. If you want to restart the VServer, use Restart().

Example:

```
get VServer_Shutdown( svrRef )
```

INI-File Functions

VServer_GetVariable(svrRef, inStrName)

Parameter:	Description:
srvRef	The reference of Server object.
inStrName	The name of server variable.

Returns: string

This Function 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:

```
put VServer_GetVariable( svrRef, "CacheSize" )  into var
```

VServer_SetVariable(svrRef,
inStrName,
inNewValue)

Parameter:	Description:
srvRef	The reference of Server object.
inStrName	The name of the server variable.
inNewValue	New value for this variable.

This Function 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:

```
get VServer_SetVariable( svrRef, "CacheSize", 8 )
```

Master Database Functions

VServer_RegisterDatabase(
svrRef,
inDbName,
[inServerFullPath = ""])

Parameter:	Description:
srvRef	The reference of Server object.
inDbName	The name of the database.
inServerFullPath	The full path of the database located on the server computer.

You can use this Function 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 Function 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:

```
get VServer_RegisterDatabase( svrRef, "DbName" )
```

This assumes that a database with name "DbName" or "DbName.vdb" exists in the "databases" folder of VServer. Vserver itself will create a path such as "systemCatalog/DbName".

Example:

```
get VServer_RegisterDatabase(
        svrRef, "Accounting", "C:\SomeCompany\account2002.vdb" )
```

VServer_UnregisterDatabase(svrRef, inDbName)

Parameter:	**Description:**
srvRef	The reference of Server object.
inDbName	The name of a database.

Returns: Boolean

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 Function.

Errors:

Database Name not found.

Example:

```
get VServer_UnregisterDatabase(svrRef,"Accounting")
```

User Functions

VServer_AddUser(svrRef,
inUserName,
inPassword,
[isAdmin = FALSE])

Parameter:	Description:
srvRef	The reference of Server object.
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:

```
get VServer_AddUser( svrRef, "Peter", "a1234fteg4" )
```

VServer_RemoveUser(svrRef, inUserName)

Parameter:	Description:
srvRef	The reference of Server object.
inUserName	The user name.

An administrator can remove users from the Master Database.

Errors:
The user name is not found.

Example:

```
get VServer_RemoveUser( svrRef, "Peter" )
```

VServer_ChangeUserPassword(svrRef,
inUserName,
inNewPassword)

Parameter:	**Description:**
srvRef	The reference of Server object.
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:

```
get VServer_ChangeUserPassword(svrRef,"Peter","rvsa3341")
```

VServer_UserName(svrRef, inUserIndex)

Parameter:	**Description:**
srvRef	The reference of Server object.
inUserIndex	The user index.

Returns: string

Returns the name of the user by index.

Example:

```
put VServer_UserName(svrRef)   into var
```

VServer_UserIsAdmin(svrRef, inUserIndex)

Parameter:	**Description:**
srvRef	The reference of Server object.
inUserIndex	The user index.

Returns: boolean

Returns TRUE if the specified user is an administrator.

Example:

```
put VServer_UserIsAdmin(svrRef,i)   into var
```

DatabaseInfo Functions

VServer_DatabaseInfo(svrRef, inIndex)

Parameter:	Description:
srvRef	The reference of Server object.
inIndex	1-based index

Returns: DatabaseInfoRef

This Function 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() Function.

Example:

```
put VServer_DatabaseCount( svrRef, count ) into var

put VServer_DatabaseInfo( svrRef, i ) into var
```

VDatabaseInfo Class

To be used only with Valentina Server.

Properties

VDatabaseInfo_ClientCount(DbInfoRef) -- (r/o) The number of connected clients.
VDatabaseInfo_CursorCount(DbInfoRef) -- (r/o) The number of cursors currently on this database.
VDatabaseInfo_Name(DbInfoRef) -- (r/o) The name of the database.
VDatabaseInfo_Path(DbInfoRef) -- (r/o) The full path of the database on the server.

Functions

VDatabaseInfo_ClientInfo(DbInfoRef, inIntIndex)

VDatabaseInfo_Refresh(DbInfoRef)

Properties

VDatabaseInfo_ClientCount(DbInfoRef)

Parameter:	Description:
DbInfoRef	The reference of DatabaseInfo object.

Returns: integer

The number of connected clients.

Example:

```
put VDatabaseInfo_ClientCount( DbInfoRef ) into var
```

VDatabaseInfo_CursorCount(DbInfoRef)

Parameter:	Description:
DbInfoRef	The reference of DatabaseInfo object.

Returns: integer

The number of cursors currently on this database.

Example:

```
put VDatabaseInfo_CursorCount( DbInfoRef ) into var
```

VDatabaseInfo_Name(DbInfoRef)

Parameter:	Description:
DbInfoRef	The reference of DatabaseInfo object.

Returns: integer

The name of the database.

Example:

```
put VDatabaseInfo_Name( DbInfoRef ) into var
```

VDatabaseInfo_Path(DbInfoRef)

Parameter:	**Description:**
DbInfoRef	The reference of DatabaseInfo object.

Returns: integer

The full path of the database on the server.

Example:

```
put VDatabaseInfo_Path( DbInfoRef ) into var
```

Functions

VDatabaseInfo_ClientInfo(DbInfoRef, inIntIndex)

Parameter:	Description:
DbInfoRef	The reference of DatabaseInfo object.
inIntIndex	The index of ClientInfo object.

Returns: ClientInfoRef

This Function 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() Function.

Example:

```
put VServerDatabaseInfo( svrRef, i ) into var
```

VDatabaseInfo_Refresh(DbInfoRef)

Parameter:	Description:
DbInfoRef	The reference of DatabaseInfo object.

This Function allows you to refresh the list of ClientInfo objects. This Function sends a request to the Valentina Server.

Example:

```
get VServer_Refresh( svrRef )
```

Class VClientInfo

Only for a V4REV Client.

Properties

```
VClientInfo_Address( ClientInfoRef )        // (r/o) The IP address of the client computer.
VClientInfo_ConnectionID( ClientInfoRef )   // (r/o) The ID of this connection.
VClientInfo_CursorCount( ClientInfoRef )    // (r/o) The number of cursors of this connection.
VClientInfo_Login( ClientInfoRef )          // (r/o) The login of this connection.
VClientInfo_Port( ClientInfoRef )           // (r/o) The port number of the client computer..
```

Properties

VClientInfo_Address(ClientInfoRef) (r/o)

Parameter: **Description:**
ClientInfoRef The reference of ClientInfo object.

Returns: String

The IP address of the client computer.

Example:

```
put VClientInfo_Address( ClientInfoRef ) into var
```

VClientInfo_ConnectionID(ClientInfoRef) (r/o)

Parameter: **Description:**
ClientInfoRef The reference of ClientInfo object.

Returns: Integer

The ID of this connection.

Example:

```
put VClientInfo_ConnectionID( ClientInfoRef ) into var
```

VClientInfo_CursorCount(ClientInfoRef) (r/o)

Parameter: **Description:**
ClientInfoRef The reference of ClientInfo object.

Returns: Integer

The number of cursors of this connection.

Example:

```
put VClientInfo_CursorCount( ClientInfoRef ) into var
```

VClientInfo_Login(ClientInfoRef) (r/o)

Parameter:	Description:
ClientInfoRef	The reference of ClientInfo object.

Returns: String

The login of this connection.

Example:

```
put VClientInfo_Login(ClientInfoRef)   into var
```

VClientInfo_Port(ClientInfoRef) (r/o)

Parameter:	Description:
ClientInfoRef	The reference of ClientInfo object.

Returns: Integer

The port number of the client computer..

Example:

```
put VClientInfo_Port(ClientInfoRef)   into var
```

VProject Class

Properties

VPorject_ReportCount(vprojectRef) (r/o)
VPorject_ReportName(vprojectRef, index) (r/o)

Construction Methods

```
VPorject_Constructor(
        inProjectLocation )

VProject_Constructor(
        inConnection,
        inProjectName )

VProject_Destructor(
        vprojectRef )
```

Report Factories for ANY Datasource (from v4.9)

```
VPorject_MakeNewReport_With_Datasource(
        vprojectRef,
        inIndexOrName,
        inDatasource,
        inQuery,
        inArrayNameOfBinds = "" )
```

Report Factories for Valentina DB

```
VPorject_MakeNewReport(
        vprojectRef,
        inIndexOrName,
        inDatabase,
        inQuery,
        inEnumCursorLocation = "kClientSide",
        inEnumLockType       = "kReadOnly",
        inEnumCursorDirection = "kForwardOnly",
        inArrayNameOfBinds = "" )
```

Properties

VProject_ReportCount(projectRef) (r/o)

Returns the count of reports inside of this container.

Example:

```
set VProject_ReportCount( projectRef ) to reports_count
```

VProject_ReportName(projectRef) (r/o)

Returns the name of Nth reports. This name can be used, for example, to show the list of all reports in the project.

Example:

```
for i = 1 to reports_count
   set VProject_ReportName( projectRef, i ) to report_name
end
```

Construction Methods

VProject_Constructor(inProjectLocation)

Returns: vprojRef

inProjectLocation The location of a Valentina project file “*.vsp”.

Description:

Constructs a new instance of VProject class. You need provide a disk location of ”.vsp” file that contains description of one or more Reports

Example:

```
my_project = VProject_Constructor( "MyProject.vsp" )

-- Now you can use methods of VProject 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.
```

VProject_Constructor(inConnection, inProjectName)

Returns: vprojRef

inConnection A connection to Valentina Server
inProjectName The name of a VProject hosted by that VServer.

Description:

Constructs a new instance of VProject class to handle project hosted on a Valentina Server. You need provide a connection object and the project name known to the Valentina Server.

Example:

```
my_project = VProject_Constructor( connectionToMyServer, "MyProject.vsp" )
```

VProject_Destructor(vprojRef)

Returns: ZERO

This function must be called before Valentina_ShutDownReports() to free resources allocated by a VProject_Constructor.

You may use form "put into dbRef" to zero vprojRef.

Example:

```
put VProject_Destructor(vprojRef) into vprojRef
```

Report Factories for ANY Datasource

```
VProject_MakeNewReport_With_Datasource(
   projectRef,
   inIndexOrName,
   inDatasource,
   inQuery = "",
   inArrayNameOfBinds = "" )
```

Parameter	Description
inIndexOrName	The index or name of a report.
inDatabase	The database that will be used as a data source.
inQuery	The SQL string of a query.
inArrayNameOfBinds	The array of bound parameters.

Returns: vreportRef

This method plays role of a VReport class factory. It returns a VReport class instance for the Nth report of this project. It will return NULL if the specified report is not found.

To create a report instance, the VREPORT DLL must know:
- Datasource that will be used to get data.
- Query that should be executed to get data.

* Parameter inQuery must contain any SQL that returns a VCursor. Usually this is a SELECT statement, although can be a SHOW statement or a CALL procedure that returns cursor.

* Parameter inQuery can be NULL on default. In this case the Report will use the original query, which is stored in the VProject, i,e, the same query that was used in the Report Editor, when this report was designed. You still can provide another query with the help of this parameter. For example you can change WHERE statement to select another records. In fact you can use very different database and table, important only that cursor have fields with same names as report expects.

IMPORTANT: When designing a report in Valentina Studio Pro, you have assigned a SQL SELECT query to this report. You have used the fields returned by that cursor to build the layout of this report. But that was during DESIGN mode.

Now, in RUNTIME mode, you can provide a completely different database and use a completely different query. The only requirement is that the query produces a cursor with the same field names, as the field names used by the report layout. If not the report will produce nothing for 'unmatched' fields.

Example:

```
put "sqlite://c:/somedb.sqlite" into sqlite_datasource

theReport = VProject_MakeNewReport_With_Datasource(
                  my_project, "report_1", sqlite_datasource, Query )

theReport = VProject_MakeNewReport_With_Datasource(
                  my_project, 1, sqlite_datasource, Query )
```

Report Factories for Valentina DB

```
VProject_MakeNewReport(
   projectRef,
   inIndexOrName,
   inDatabase,
   inQuery = "",
   inCursorLocation = "kClientSide",
   inLockType       = "kReadOnly",
   inCursorDirection = "kForwardOnly",
   inArrayNameOfBinds = "" )
```

Parameter	Description
inIndexOrName	The index or name of a report.
inDatabase	The database that will be used as a data source.
inQuery	The SQL string of a query.
inCursorLocation	The location of cursor. See CursorLocation types.
inLockType	The lock type for records of a cursor. See LockType types.
inCursorDirection	The direction of a cursor. See CursorDirection types.
inArrayNameOfBinds	The array of bound parameters.

Returns: vreportRef

This method plays role of a VReport class factory. It returns a VReport class instance for the Nth report of this project. It will return NULL if the specified report is not found.

To create a report instance, the VREPORT DLL must know:
- A database that will be used to get data.
- Query that should be executed to get data
- Some optional parameters for this query.

IMPORTANT: If the project is local then inDatabase can be both local and remote located on any Valentina Server. But if the project is hosted on a Valentina Server, then inDatabase MUST be hosted on the same server.

* Parameter inQuery must contain any SQL that returns a VCursor. Usually this is a SELECT statement, although can be a SHOW statement or a CALL procedure that returns cursor.

* Parameter inQuery can be NULL on default. In this case the Report will use the original query, which is stored in the VProject, i,e, the same query that was used in the Report Editor, when this report was designed. You still can provide another query with the help of this parameter. For example you can change WHERE statement to select another records. In fact you can use very different database and table, important only that cursor have fields with same names as report expects.

* Parameters inCursorLocation, inLockType, inCursorDirection, inArrayNameOfBinds are the same as for VDatabase.SqlSelect() method. Please read details about them there.

IMPORTANT: When designing a report in Valentina Studio Pro, you have assigned a SQL SELECT query to this report. You have used the fields returned by that cursor to build the layout of this report. But that was during DESIGN mode.

Now, in RUNTIME mode, you can provide a completely different database and use a completely different query. The only requirement is that the query used produces a cursor with the same field names as the field names used by the report layout. If not the report will produce nothing for 'unmatched' fields.

Example:

```
theReport = VProject_MakeNewReport( my_project, "report_1", mDB, Query )
theReport = VProject_MakeNewReport( my_project, 1, mDB, Query )
```

VReport Class

Properties

VReport_PageCount(reportRef) (r/o)

Preview Properties

VReport_PreviewZoom(reportRef, [inValue])
VReport_PreviewWidth(reportRef, [inValue])
VReport_PreviewHeight(reportRef, [inValue])

Preview Methods

```
VReport_PreviewPage(
        reportRef
        inPageIndex )
```

Printing Methods

```
VReport_PrintToDisk(
        reportRef
        inLocation,
        inPrintType,
        inStartPageIndex = 0,
        inEndPageIndex = 0 )
```

Properties

VReport_PageCount(reportRef) (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:

```
set VReport_PageCount( reportRef ) to pages
```

VReport_PreviewZoom(reportRef, [inValue])

Specifies the preview zoom in percents 1-100. Affects only following calls of VReport. PreviewPage() method.

Example:

```
get VReport_PageZoom( reportRef, 50 )
set VReport_PreviewPage( reportRef ) to preview
```

VReport_PreviewWidth(reportRef, [inValue])

Specifies the width of preview in pixels. Affects only the following calls of VReport.Preview-Page() method.

Example:

```
get VReport_PreviewWidth( reportRef, 600 )
set VReport_PreviewPage( reportRef ) to preview
```

VReport_PreviewHeight(reportRef, [inValue])

Specifies the height of preview in pixels. Affects only the following calls of VReport.Pre-viewPage() method.

Example:

```
get VReport_PreviewHeight( reportRef, 600 )
set VReport_PreviewPage( reportRef ) to preview
```

Preview Methods

VReport_previewPage(reportRef, inPageIndex)

Parameter:	Description:
reportRef	The reference of report object.
inPageIndex	index of report page to preview. Starts from 1.

Returns: Picture

Description:

This method generates preview of Nth page of the report.

Usually you need this to build some kind of user interface to show preview of first report pages before printing.

Example:

```
page_preview_pict = VReport_PreviewPage( reportRef, 1 )
```

Printing Methods

```
VReport_ PrintToDisk(
        reportRef,
        inLocation,
        inPrintType,
        inStartPageIndex = 0,
        inEndPageIndex = 0   )
```

Parameter:	**Description:**
reportRef	The reference of report object.
inLocation	The location for generated file.
inPrintType	Specifies the format of generated report.
inStartPageIndex	The index of the first page to be printed (1..N). Zero to print all records of the report
inEndPageIndex	The index of the last page to be printed (1..N).

Returns: VOID

Description:

Prints all pages or the specified range of pages of the report to the disk file at the given location.

You can specify format of produced file with help of the inPrintType parameter. . Usually you will use such values as kToPDF, kToHTML, kToPicture_JPG.

Example:

```
VReport_PrintToDisk( reportRef,  "kToHTML", "report_1.html" )
VReport_PrintToDisk( reportRef, "kToPDF", "report_1.pdf" )
```