VALENTINA 5

for COM Reference

Paradigma Software Inc.
www.paradigmasoft.com
© 1998 - 2014

Contents

Valentina for COM installation

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

Installer for Windows installs on your computer:

• VCOM_5 folder into COM:plugins folder. This folder contains VCOM plugin itself and folder Examples.

• VComponents folder into
C:/Program Files/Paradigma Software/vcomponents_win_vc

This folder contains several DLLs, see detailed description of VComponents folder in the Valentina Kernel.pdf.

• append to the system variable PATH the path to VComponents folder, so Windows can find and load DLLs.

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

Where to start

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

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

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

Deployment of your application

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

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

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

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

Now you can distribute this "MyAppFolder" folder.

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

2) Valentina folder is located in the system area.

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

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

Valentina Enumeration Types (Enums)

Valentina for COM have Enumaration Types - Enums.

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

EVValueAccess

forAdd	= 1
forUpdate	= 2

EVOs

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

EVDateFormat

kMDY	= 0
kDMY	= 1
kYMD	= 2

EVDebugLevel

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

EVDbMode

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

EVFlag

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

EVOnDelete

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

EVOnUpdate

kUpdNoAction	= 0
kUpdSetNull	= 1
kUpdCascade	= 2
kUpdRestrict	= 3
kUpdDefault	= 4

EVRecursionDirection

kFromParentToChild	= 0
kFromChildToParent	= 1

EVStorageType

kStorageDefault	= 0
kStorageDisk	= 1
kStorageRAM	= 2

EVTableKind

kTblPermanent	= 0
kTblTemporary	= 1

EVCursorLocation

kClientSide	= 1
kServerSide	= 2

EVLockType

kNoLocks	= 1
kReadOnly	= 2
kReadWrite	= 3

EVCursorDirection

kForwardOnly	= 1
kRandom	= 2

EVLinkType

kMany	= 0
kOne	= 1

EVFieldType

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

EVDumpType

kSQL	= 1
kXML	= 2

EVDataKind

kStructureOnly	= 1
kStructureAndRecords	= 2
kRecordsOnly	= 3

EVSearch

kPreferIndexed	= 0
kPreferNonIndexed	= 1

EVVerboseLevel

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

EVColAttribute

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

EVColAttributeValue

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

EVPictType

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

Interface IValentina

Properties

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

Database(inIndex as Integer) as IVDatabase

Initialisation Methods

```
Init(     inCacheSize as Integer = 10,
          inWinSerialNumber as String = "" )

InitClient(
          inCacheSize as Integer = 10 )

InitReports()

ShutDown()
ShutDownClient()
ShutDownReports()
```

Utility Methods

```
SetExtensions( inDesc as String, inDat as String, inBlb as String, inInd as String)
EscapeString( inStr as String ) As String

GetDatabaseFormatVersion( inVdbFile as String ) as Integer
GetCurrentFormatVersion() as Integer

GetSchemaVersion( inVdbFile as String ) as Integer
GetDatabaseMode( inVdbFile as String ) as Integer

GetIsStructureEncrypted( inVdbFile as String ) as Boolean
```

Properties

CacheSize as Integer (r/o)

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

Example (Visual BASIC):

```
size = Valentina.CacheSize
```

DebugLevel as EVDebugLevel

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

Any debug level above 0 will create a file which outputs the results. The file will be named “VCOM_Log.txt”. It will be created in the same directory as the project.

The valid values are:

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

Example (Visual BASIC):

```
Valentina.Init( 16 * 1024  )
Valentina.DebugLevel = EVDebugLevel.kLogParams
```

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

FlushEachLog as Boolean

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

TIP: You can wrap the problematic code only.

Example (Visual BASIC):

```
Valentina.FlushEachLog = True
        // some debugged code
Valentina.FlushEachLog = False
```

Version as String (r/o)

Returns the version of the Valentina engine.

Example (Visual BASIC):

```
ver = Valentina.Version
```

DatabaseCount as Integer (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 (Visual BASIC):

```
res = Valentina.DatabaseCount
```

Database(inIndex as Integer) as IVDatabase

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

See also:

Valentina.DatabaseCount()

Example (Visual BASIC):

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

Initialisation Methods

```
Init(
        inCacheSize as Integer = 10,
        inWinSerialNumber as String = "" )
```

Parameter:	Description:
inCacheSize	The size of the cache in mega-bytes.
inWinSerialNumber	The serial for Windows.

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

Tip: By default, it is a good idea to allocate not more than half of available computer memory to the cache. Usually 10-50Mb is enough.

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 serial number. If Valentina receives an empty string, it will work in the time limited, demonstration mode. After ten minutes in demonstration mode, any request to the database will be ignored and Valentina will respond with three beeps.

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

Example (Visual BASIC):

```
        Valentina.Init( 16 )  ' demo mode, because no serial. 16 MB cache.
```

```
InitClient(
        inCacheSize as Integer = 10 )
```

Parameter:	Description:
inCacheSize	The size of the cache in mega-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 (Visual BASIC):

```
        Valentina.InitClient
```

ShutDown()

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

Example (Visual BASIC):

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

ShutDownClient()

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

Пример:

```
Valentina.ShutDownClient()
```

Utility Methods

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

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

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

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

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

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

Example(s):

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

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

EscapeString(inStr as String) as String

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

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):

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

query = "SELECT * FROM T WHERE f1 LIKE '" + s1 + "'
```

GetCurrentFormatVersion() as Integer

Returns the current format version of database file.

Example (Visual BASIC):

```
vers = Valentina.GetCurrentFormatVersion
```

GetDatabaseFormatVersion(inVdbFile as String) as Integer

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

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

Example (Visual BASIC):

```
Dim vers As Integer

vers = Valentina.GetDatabaseFormatVersion( dbPath )
```

GetSchemaVersion(inVdbFile as String) as Integer

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

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

Example (Visual BASIC):

```
dim SchemaVersion as Integer

SchemaVersion = Valentina.GetSchemaVersion( dbPath )
```

GetDatabaseMode(inVdbFile as String) as Integer

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

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

Example (Visual BASIC):

```
dim dbMode as Integer

dbMode = Valentina.GetDatabaseMode( dbPath )
```

GetIsStructureEncrypted(inVdbFile as String) as Boolean

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

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

Example (Visual BASIC):

```
dim isEncrypted as Integer

isEncrypted = Valentina.GetStructureEncrypted( dbPath )
```

Interface IVDatabase

Properties

CenturyBound	as Integer	// default 20.

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

DateFormat	as EVDateFormat	// specifies the format of date.
DateSep	as String	// separator for date, e.g. ‘/’
IndexCount	as Integer (r/o)	
IsEncrypted	as Boolean (r/o)	// TRUE if the database is encrypted.
IsOpen	as Boolean (r/o)	
IsReadOnly	as Boolean (r/o)	
IsRemote	as Boolean (r/o)	
LastInsertedRecID	as Integer (r/o)	
LinkCount	as Integer (r/o)	
LocaleName	as String	
Mode	as EVDbMode (r/o)	
Name	as String (r/o)	
Path	as String (r/o)	
SchemaVersion	as Integer	// Version of db Schema
SegmentSize	as Integer (r/o)	
StorageEncoding	as String	
TableCount	as Integer (r/o)	
TimeSep	as String	// separator for time, e.g. ‘:’

// for CLIENT only:

ResponseTimeout	as Integer	// default 60 seconds.

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

Constructor

InitLocal(inStorageType as EVStorageType = kDefault)

```
InitClient(
        inHost as String,
        inUserName as String,
        inUserPassword as String,
        inPort as Integer = 15432,
        inTimeOut as Integer = 5,
        inOptions as String = "")
```

InitClientWithServer(inServer as VServer)

Disk Methods

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

Open( inLocation as String )
Close()
ThrowOut()
Flush()

Clone1( inTargetDb as String, inLoadRecords as Boolean = true, inDoLog as Boolean = false )
Clone2( inTargetDb as IVDatabase, inLoadRecords as Boolean = true, inDoLog as Boolean = false )
```

Structura methods

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

DropTable(inTable as IVTable)

CreateForeignKeyLink(
        inName as String,
        inKeyField as IVField,
        inPtrField as IVField,
        inOnDelete as EVOnDelete = kSetNull,
        inOnUpdate as EVOnUpdate = kUpdCascade,
        inTemporary as Boolean = FALSE ) as IVLink

CreateBinaryLink(
        inName as String,
        inLeftTable as IVTable,
        inRightTable as IVTable,
        inLeftPower as EVLinkType = kOne,
        inRightPower as EVLinkType = kMany,
        inOnDelete as EVOnDelete = kSetNull,
        inStorageType as EVStorageType = kStorageDefault,
        inTemporary as Boolean = false) as IVBinaryLink

DropLink( inLink as IVLink )
```

Table methods

Table(inNameOrIndex as Variant) as IVTable

Link methods

Link(inNameOrIndex as VARAINT) as IVLink

IndexStyle methods

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

SQL methods

```
SqlExecute(
        inQuery as String,
        inBinds() as Variant = null ) as Integer

SqlSelect(
        inQuery as String,
        inCursorLocation as EVCursorLocation = kClientSide,
        inLockType as EVLockType = kReadOnly,
        inCursorDirection as EVCursorDirection = kForwardOnly
        inBinds() as Variant = null ) as IVCursor
```

Encryption method

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

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

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

RequiresEncryptionKey()

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

Dump methods

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

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

Utility methods

```
Diagnose(
        inVerboseLevel as EVVerboseLevel = kNone,
        inFile as String = "" ) as Boolean
```

Description

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

Properties

CenturyBound as Integer

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

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

The default is 20.

Example (Visual BASIC):

```
cntb = db.CenturyBound
```

CollationAttribute(
inColAttribute as EVColAttribute) as EVColAttributeValue

CollationAttribute(
inColAttribute as EVColAttribute,
inColAttributeValue as EVColAttributeValue)

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

Example (Visual BASIC):

```
Dim v As Integer

v = database.CollationAttribute( EVColAttribute.kStrength )

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

```
ConnectionVariable(
        inConnVariableName as EVConnectionVariable ) as EVConnectionVariableValue

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

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

Example (Visual BASIC):

```
        Dim i As EVConnectionVariableValue

        i = database.ConnectionVariable(EVConnectionVariable.kFilesTransferMode)

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

DateFormat as EVDateFormat

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

Example (Visual BASIC):

```
dtf = db.DateFormat
```

DateSep as String

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

Example (Visual BASIC):

```
dts = db.DateSep
```

IndexCount as Integer (r\o)

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

Example (Visual BASIC):

```
count = db.IndexCount
```

IsEncrypted as Boolean (r\o)

Returns TRUE if this database is encrypted.

Example (Visual BASIC):

```
encrypted = db.isEncrypted
```

IsReadOnly as Boolean (r/o)

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

Example (Visual BASIC):

```
res = db.IsReadOnly
```

IsRemote as Boolean (r/o)

Returns TRUE if this database is remote.

Example (Visual BASIC):

```
res = db.IsRemote
```

IsOpen as Boolean (r\o)

Returns TRUE if this database is open now.

Example (Visual BASIC):

```
res = db.IsOpen
```

LastInsertedRecID as Integer (r/o)

Returns: Integer

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

This function is useful mainly if you execute

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

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

Function IVTable.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 (Visual BASIC):

```
recid = db.LastInsertedRecID
```

LinkCount as Integer (r\o)

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

Example (Visual BASIC):

```
count = db.LinkCount
```

LocaleName as String

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

Example (Visual BASIC):

```
locale = db.LocaleName
db.LocaleName = "en_US"
```

Mode as EVDbMode (r\o)

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

Example (Visual BASIC):

```
mode = db.Mode
```

Name as String (r\o)

The name of database.

Example (Visual BASIC):

```
name = db.Name
```

Path as String (r\o)

The full path to this database.

Example (Visual BASIC):

```
path = db.Path
```

SchemaVersion as Integer

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

Example (Visual BASIC):

```
ver = db.SchemaVersion
```

SegmentSize as Integer (r\o)

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

Example (Visual BASIC):

```
seg = db.SegmentSize
```

StorageEncoding as String

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

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

Example (Visual BASIC):

```
encoding = db.StorageEncoding
```

TableCount as Integer (r\o)

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

Example (Visual BASIC):

```
count = db.TableCount
```

TimeSep as String

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

Example (Visual BASIC):

```
tms = db.TimeSep
```

ResponseTimeOut as Integer

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

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

Example (Visual BASIC):

```
db.ResponseTimeOut = 100
```

Local vs Remote Creation

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

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

Parameter	Description
inStorageType	Storage type for this database

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

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

Example (Visual BASIC):

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

```
InitClient(
    inHost as String,
    inUserName as String,
    inUserPassword as String,
    inPort as Integer = 15432 ,
    inTimeOut as Integer = 5,
    inOptions as String = "" )
```

Parameter	Description
inHost	The IP-adress or DNS name of host
inUserName	The user name
inuserPassword	The password of user
inPort	The port number of Valentina Server On 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

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

If you do not use the optional inPort parameter, a connection using this object will be attempted using VServer's default port.

Note: If in your VSerer.ini file you have specified a non-default port value, then the parameter inPort is required in the IVDatabase constructor. The inPort parameter allows you (or other developers) to have more than one VServer available on a Host computer.

Example (Visual BASIC):

```
Set remote_db = new VDatabase
remote_db.InitClient( "somecompany.com", "sa", "sa" )
```

InitClientWithServer(inServer as VServer)

Parameter	**Description**
inServer	Instance of Vserver class.

This form of constructor should be used in the client-server mode only.

You can wish to use this form of constructor if you want access several databases in the scope of single connection. So at first you should create an instance of VServer class and establish connection then you create few IVDatabase class instances.

Example (Visual BASIC):

```
Dim vsrv As VServer
Dim db1 As VDatabase
Dim db2 As VDatabase

Set vsrv = new VServer
vsrv.Init( "myhost",  "name", "passw" )

vsrv.OpenSession()

Set db1 = new VDatabase
db1.InitClientWithServer( vsrv )

Set db2 = new VDatabase
db2.InitClientWithServer( vsrv )
```

Disk Methods

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

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

Creates a new, empty database on disk.

Note: After creation, the database is already open.

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

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

Example (Visual BASIC):

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

Example (Visual BASIC):

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

remote_db.Create( "My Database1", kDscDatBlb_Ind, 16 * 1024 )
```

Open(inLocation as String)

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

Opens an existing database at the specified location.

Example (Visual BASIC):

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

Example (Visual BASIC):

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

remote_db.Open( "My Database1" )
```

Close()

Closes the database.

Example (Visual BASIC):

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

ThrowOut()

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

Example (Visual BASIC):

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

Flush()

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

Example (Visual BASIC):

```
db.Flush()
```

IsRemote (r/o)

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

Example (Visual BASIC):

```
db.IsRemote
```

```
Clone1(
        inTargetDb as String,
        inLoadRecords as Boolean = true,
        inDoLog as Boolean = false )
```

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

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

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

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

Example (Visual BASIC):

```
        db.Clone1( newDbLocation )
```

```
Clone2(
        inTargetDb as IVDatabase,
        inLoadRecords as Boolean = true,
        inDoLog as Boolean = false )
```

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

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

Example (Visual BASIC):

```
        Set dbCloned = new VDatabase
        dbClone.InitLocal()

        // we create it with some other segment for example.
        dbCloned.Create( newDbLocation, kDscDatBlb_Ind, 8 * 1024 )

        db.Clone( dbCloned )
```

Structure Methods

```
CreateTable(
        inName as String,
        inTableKind as EVTableKind = kTblPermanent,
        inStorageType as EVStorageType = kStorageDefault ) as IVTable
```

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

Creates a new empty Table in the database.

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

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

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

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

Example (Visual BASIC):

```
Dim tbl As VTable

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

```
DropTable( inTable as IVTable )
```

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

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

Example (Visual BASIC):

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

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

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

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

To specify a link you need to define the following:

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

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

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

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

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

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

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

Example (Visual BASIC):

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

```
CreateForeignKeyLink(
        inName as String,
        inKeyField as IVField,
        inPtrField as IVField,
        inOnDelete as EVOnDelete = kSetNull,
        inOnUpdate as EVOnUpdate = kUpdCascade,
        inTemporary as Boolean = false ) as IVLink
```

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

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

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

Example (Visual BASIC):

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

```
DropLink( inLink as IVLink )
```

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

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

Example (Visual BASIC):

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

Table Methods

Table(inNameOrIndex as VARAINT) as IVTable

Parameter:	Description:
inNameOrIndex	The Name of a Table or The index of a Table in a database, starts from 1.

Returns a Table by name or by its index.

Note: The parameter inName is case insensitive.

Example (Visual BASIC):

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

Example (Visual BASIC):

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

Link Methods

Link(inNameOrIndex as VARAINT) as IVLink

Parameter:	Description:
inName	The Name of a Link or The index of a Link in a database, starts from 1.

Returns a Link by name.

Note: The parameter inName is case insensitive.

Example (Visual BASIC):

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

Example (Visual BASIC):

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

SQL Methods

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

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

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

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

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

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

You can set the following parameters with these values:

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

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

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

Example (Visual BASIC):

```
Dim curs As VCursor
Set curs = db.SqlSelect( "SELECT * FROM T " )
```

Example (Visual BASIC):

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

Example (Visual BASIC):

```
Dim arrBind(2) As Variant

arrBind(0) = "john"
arrBind(1) = 25

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

```
SqlExecute(
        inQuery as String,
        inBinds as VARIANT = null) as Integer
```

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

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

This returns the number of affect rows.

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

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

Example (Visual BASIC):

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

Example (Visual BASIC):

```
Dim Binds(2) As Variant

Binds(0) = "john"
Binds(1) = "jehn"

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

IndexStyle Methods

CreateIndexStyle(inName as String) as IVIndexStyle

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

Creates a new Index Style in the database.

Example (Visual BASIC):

```
Dim indStyle1 As VIndexStyle
Set IndexStyle1 = db.CreateIndexStyle( "myStyle" )
```

DropIndexStyle(inStyle as IVIndexStyle)

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

Deletes the specified index style from the database.

Example (Visual BASIC):

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

IndexStyle(inName as String) as IVIndexStyle

Parameter:	Description:
inName	The Name of a IndexStyle.

Returns an IndexStyle by name.

Note: The parameter Name is case insensitive.

Example (Visual BASIC):

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

Encryption Methods

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

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

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

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

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

Allows you to encrypt the database.

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

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

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

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

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

Note:

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

Example (Visual BASIC):

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

Example (Visual BASIC):

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

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

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

Allows to decrypt the database.

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

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

Example (Visual BASIC):

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

Example (Visual BASIC):

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

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

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

Allows you to change the encryption key for the database.

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

Example (Visual BASIC):

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

Example (Visual BASIC):

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

RequiresEncryptionKey() as Boolean

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

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

Example (Visual BASIC):

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

UseEncryptionKey(

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

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

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

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

Example (Visual BASIC):

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

Examp;e:

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

Dump Methods

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

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

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 (Visual BASIC):

```
Dim db As VDatabase
...
db.Dump( pathXML, EVDumpType.kXML )
```

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

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

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

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

Example (Visual BASIC):

```
Dim db As VDatabase
...
db.LoadDump( pathXML, pathNewDb, EVDumpType.kXML )
```

Utility methods

```
Diagnose(
        inFile as String = "",
        inVerboseLevel as EVVerboseLevel = kNone ) as Boolean
```

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

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

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

Parameter inVerboseLevel can accept the following values:

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

Example (Visual BASIC):

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

Interface IVTable

Properties

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

Database	as IVDatabase (r/o)	// Database of this Table.
FieldCount	as Integer (r/o)	// (r/o) number of fields in this Table
ID	as Integer (r/o)	
Name	as String	
IsEncrypted	as Boolean (r/o)	
LinkCount	as Integer (r/o)	
LocaleName	as String	
PhysicalRecordCount	as Integer (r/o)	
RecID	as Integer	
RecordCount	as Integer (r/o)	// (r/o) number of logical records in this Table.
StorageEncoding	as String	

Field Methods

Field(inNameOrIndex as VARAINT) as IVField

Link Methods

Link(inNameOrIndex as VARAINT) as IVLink

Record Methods

SetBlank(inAccess as EVValueAccess = forUpdate)	// Clears a memory buffer of a Table, // sets nullable fields to NULL.
AddRecord() as Integer	// Adds a new record with the current value of fields.
DeleteRecord()	// Deletes the current record.
DeleteAllRecords(inSet as IVSet = null)	// Delete all records of table or selection.
UpdateRecord()	// Updates an existing record with new values.
UpdateAllRecords(inSet as IVSet = null)	// Update all records of table or selection.

Cach Methods

Flush()	// Saves information of this Table only to disk.

Navigation Methods

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

RecordExists(inRecID as Integer) as Boolean

```
// Set of handy CreateXXXField()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

CreateObjectPtrField(
        inName as String,
        inTarget as IVTable,
        inOnDeletion as EVOnDelete = kCascade,
        inFlags as EVFlag = fNone ) as IVField
```

Structure Methods

```
DropField( inFld as IVField )

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

Encryption Methods

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

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

Dump Methods

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

LoadDump (
        inDumpFile as String,
        inDumpType as EVDumpType,
        inEncoding as String = "UTF-16" )
```

Selection Methods

```
SelectAllRecords as IVBitSet,
SelectNoneRecords as IVBitSet,

Sort(   inSet as IVSet,
        inField as IVField,
        inAcending as Boolean = true ) as IVArraySet

SortN(  inSet as IVSet,
        inItems as VARIANT ) as IVArraySet
```

Description

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

Properties

CollationAttribute(
    inColAttribute as EVColAttribute) as EVColAttributeValue

CollationAttribute(
    inColAttribute as EVColAttribute,
    inColAttributeValue as EVColAttributeValue)

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

Example (Visual BASIC):

```
Dim v As Integer

v = table.CollationAttribute( EVColAttribute.kStrength )

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

Database as IVDatabase (r/o)

Returns the parent database of this table.

Example (Visual BASIC):

```
Set db = table.Database
```

FieldCount as Integer (r/o)

Returns the number of custom fields in the table.

Example (Visual BASIC):

```
fldCount = table.FieldCount
```

ID as Integer (r/o)

Returns the unique identifer of the table.

Example (Visual BASIC):

```
id = table.ID
```

Name as String

The name of the table.

Example (Visual BASIC):

```
Dim sname As String

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

IsEncrypted as Boolean (r/o)

Returns TRUE if the database is encrypted.

Example (Visual BASIC):

```
encrypted = table.IsEncrypted
```

LinkCount as Integer (r/o)

Returns the number of links in the table.

Example (Visual BASIC):

```
Dim LinkCount As Integer
LinkCount = table.LinkCount
```

LocaleName as String

Specifies the locale name for this table.

Example (Visual BASIC):

```
Dim LocaleName As String
LocaleName = table.LocaleName

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

PhysicalRecordCount as Integer (r/o)

Returns the number of physical records in the table.

Example (Visual BASIC):

```
physRecCount = table.PhysicalRecordCount
```

RecID as Integer

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

Example (Visual BASIC):

```
recID = Table.RecID

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

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

RecordCount as Integer (r/o)

Returns the number of logical records in the table.

Example (Visual BASIC):

```
rcdCount = table.RecordCount
```

StorageEncoding as String

Specifies for this table the string encoding stored on disk.

Example (Visual BASIC):

```
Dim Encoding As String
Encoding = table.StorageEncoding

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

Field Methods

Field(inNameOrIndex as VARIANT) as IVField

Parameter:	**Description:**
inNameOrIndex	The name of the field or The index of the field. Starts from 1.

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

Also this method allows you to access fields of a Table by index. If the field with the specified index doesn’t exist then it returns null.

Example (Visual BASIC):

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

```
Dim fld As VFieId
Dim fldString As VString

Set fld = boPerson.Field(1)
If( fld.Type = kTypeString ) then
        Set fldString =  fld
        ' now you can access properties of VString field:
        ' MaxLength, Language,...  using fldString
End If
```

Example (Visual BASIC):

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

Link Methods

Link(inNameOrIndex as VARIANT) as IVLink

Parameter:	Description:
inNameOrIndex	The name of a link or The index of a link. Starts from 1.

Returns a link of this table by name or by numeric index.

Example (Visual BASIC):

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

Example (Visual BASIC):

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

Record Methods

SetBlank(inAccess as EVValueAccess = forUpdate)

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

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

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

Example (Visual BASIC):

```
Table.SetBlank()
```

AddRecord() as Integer

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

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

Example (Visual BASIC):

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

DeleteRecord()

Deletes the current record of a Table.

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

Example (Visual BASIC):

```
Table.DeleteRecord()
```

DeleteAllRecords(inSet as IVSet = null)

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

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

Example (Visual BASIC):

```
Table.DeleteAllRecords()
```

UpdateRecord()

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

Example (Visual BASIC):

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

UpdateAllRecords(inSet as IVSet = null)

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

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

Example (Visual BASIC):

```
Table.UpdateAllRecords()
```

Cache Methods

Flush()

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

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

Example (Visual BASIC):

```
Table.Flush()
```

Navigation Methods

FirstRecord() as Boolean

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

Example (Visual BASIC):

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

LastRecord() as Boolean

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

Example (Visual BASIC):

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

PrevRecord() as Boolean

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

Example (Visual BASIC):

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

NextRecord() as Boolean

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

Example (Visual BASIC):

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

RecordExists(inRecID as Integer) as Boolean

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

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

Example (Visual BASIC):

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

Structure Methods

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

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

Methods to create numeric fields

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

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

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

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

Example (Visual BASIC):

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

Methods to create string/varchar fields

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

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

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

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

Example

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

Methods to create fixed/var binary fields

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

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

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

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

Example

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

Method to create BLOB fields.

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

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

Create a BLOB (Binary Large Object) field.

• You need to specify the segment size in bytes.

Example

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

Method to create TEXT fields.

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

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

Create a Text field.

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

Example

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

Method to create Picture fields.

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

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

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

Example

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

Method to create ObjectPtr fields.

```
CreateObjectPtrField(
        inName as String,
        inTarget as IVTable,
        inOnDeletion as EVOnDelete = kCascade,
        inFlags as EVFlag = fNone ) as IVField
```

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

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

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

DropField(inFld as IVField)

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

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

Example (Visual BASIC):

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

ChangeType(

```
    inFld as IVField,
    inNewType as EVFieldType,
    inParam1 as Integer ) as IVField
```

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

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

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

Example (Visual BASIC):

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

Encryption Methods

The IVTable class has a set of functions for encryption analog to functions of the IVDatabase and IVField classes.

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

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

Encrypt(inKey as String)

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

Allows you to encrypt the Table.

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

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

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

Example (Visual BASIC):

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

Decrypt(inKey as String)

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

Allows to decrypt the Table.

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

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

Example (Visual BASIC):

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

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

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

Allows you to change the encryption key fot the Table.

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

Example (Visual BASIC):

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

RequiresEncryptionKey() as Boolean

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

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

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

Example (Visual BASIC):

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

UseEncryptionKey(inKey as String)

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

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

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

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

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

Example (Visual BASIC):

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

tbl.UseEncryptionKey( "key12345" )
```

Dump Methods

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

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

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

Example (Visual BASIC):

```
Dim tbl As VTable
...
tbl.Dump( pathXML, EVDumpType.kXML )
```

```
LoadDump(
        inDumpFile as String,
        inDumpType as EVDumpType,
        inEncoding as String = "UTF-16" )
```

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

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

Example (Visual BASIC):

```
Dim tbl As VTable
...
tbl.loadDump( pathXML,  EVDumpType.kXML )
```

Selection Methods

SelectAllRecords as IVBitSet

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

Example (Visual BASIC):

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

SelectNoneRecords as IVBitSet

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

Example (Visual BASIC):

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

```
Sort(
        inSet as IVSet,
        inField as IVField,
        inAscending as Boolean = true ) as IVArraySet
```

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

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

Returns a new sorted selection as an ArraySet.

Example (Visual BASIC):

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

```
SortN( inSet as IVSet,
        inItems as VARIANT ) as IVArraySet
```

Parameter:	Description:
inSet	The set to be sorted.
inItems	Array of items

Executes sorting of a table selection inSet on several fields.

The second parameter is array that specify which fileds to sort and, optionally, order of sorting for each field. You can specify Name of Field or Index of Field in the Table.

Example (Visual BASIC):

```
Dim SortItems As Variant

SortItems(0) = "fldname1"
SortItems(1) = "fldname2"

Set SortedSet = table.Sort( allRecs,  SortItems )
```

Also you can specify order of sorting for each and any field. For this you can add boolean parameter right after the field.

Example (Visual BASIC):

```
Dim SortItems As Variant

SortItems(0) = "fldname1"
SortItems(0) = True                 // Ascending
SortItems(1) = "fldname2"
SortItems(1) = False                // Descending

Set SortedSet = table.Sort( allRecs,  SortItems )
```

Interface IVField

Properties

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

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

Value methods

SetBlank()	// clear the value of the field.
GetString() as String	// returns a value of the Field as a String
SetString(inValue as String)	// store a String value in the Field

Search methods

```
ValueExists(
        inValue as Variant,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as Boolean

ValueExistsWithCount(
        inValue as Variant,
        ByRef outCount as Integer,
        inSelection as IVSet = nulll,
        inSearchPref as EVSearch = kPreferIndexed ) as Boolean
```

```
FindValue(
        inValue as Variant,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet

FindValueAsArraySet(
        inValue as Variant,
        inSelection as IVSet = null,
        inMaxCount as Integer = &hffffffff,     // ulong_max
        inSearchPref as EVSearch = kPreferIndexed ) as IVArraySet

FindRange(
        inLeftInclude as Boolean,
        inLeftValue as Variant,
        inRightValue as Variant,
        inRightInclude as Boolean,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet

FindRangeAsArraySet(
        inLeftInclude as Boolean,
        inLeftValue as Variant,
        inRightValue as Variant,
        inRightInclude as Boolean,
        inSelection as IVSet = null,
        inMaxCount as Integer = &hffffffff,     // ulong_max
        inSearchPref as EVSearch = kPreferIndexed ) as IVArraySet

FindSingleValue(
        inValue as Variant,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as Integer

FindNulls(
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet

FindNotNulls(
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

```
FindStartsWith(
        inValue as String,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet

FindContains(
        inValue as String,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet

FindEndsWith(
        inValue as String,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet

FindRegEx (
        inValue as String,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet

FindLike(
        inValue as String,
        inEscapeChar as String = "\",
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet

FindDistinct(
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

Encryption methods

```
RequiresEncryptionKey() as Boolean

Decrypt(inKey as String)
Encrypt(inKey as String)
UseEncryptionKey(inKey as String)

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

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

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

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

```
Dim fld As VField
Dim str_fld As VString

Set fld = Person.Field( “Name” )
Set str_fld =  fld
If Not str_fld Is Nothing
   maxLen = str_fld.MaxLength
End If
```

Properties

CollationAttribute(inColAttribute as EVColAttribute) as EVColAttributeValue

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

The value of the specified collation attribute for this table.

Example (Visual BASIC):

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

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

DefaultValue as 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 null.

Example (Visual BASIC):

```
v = fld.DefaultValue
```

ID as Integer (r/o)

Return the unique identifier of the field.

Example (Visual BASIC):

```
id = fld.ID
```

IndexStyle as IVIndexStyle

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 (Visual BASIC):

```
fld.IndexStyle = style1

Set currStyle = fld.IndexStyle
```

IsEncrypted as Boolean (r/o)

Returns TRUE if the database is encrypted.

Example (Visual BASIC):

```
encrypted = fld.IsEncrypted
```

IsIndexed as Boolean

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

Example (Visual BASIC):

```
fld.IsIndexed = False
        ... ' add many records for example
fld.Indexed = True
```

IsMethod as Boolean (r/o)

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

Example (Visual BASIC):

```
If( fld.IsMethod )
```

IsNullable as Boolean

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

Example (Visual BASIC):

```
fld.IsNullable = True
If( fld.Nullable )
```

IsNull as Boolean

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

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

Example (Visual BASIC):

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

IsUnique as Boolean

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

Example (Visual BASIC):

```
fld.IsUnique = True
If( fld.Unique )
```

LocaleName as String

Specifies the locale name for this field.

Example (Visual BASIC):

```
LocaleName = fld.LocaleName

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

MethodText as String

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

Example (Visual BASIC):

```
method = fld.MethodText

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

Name as String

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

Example (Visual BASIC):

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

StorageEncoding as String

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

Example (Visual BASIC):

```
Encoding = fld.StorageEncoding

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

Table as IVTable (r/o)

Returns the Table of this field.

Example (Visual BASIC):

```
Set t = fld.Table
```

Type as EVFieldType (r/o)

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

Each field has several flags, which define its behavior:

Example (Visual BASIC):

```
case fld.Type
```

See also: IVTable.ChangeType

TypeString as String (r/o)

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

Example (Visual BASIC):

```
strType = fld.TypeString
```

Value as Variant

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

Example (Visual BASIC):

```
Dim f As VField
Dim iv As Integer

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

Value Methods

SetBlank()

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

Example (Visual BASIC):

```
fld.SetBlank()
```

GetString() as String

Returns the value of the field as a string.

Example (Visual BASIC):

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

SetString(inValue as String)

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

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

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

Example (Visual BASIC):

```
str = "aaaaa"
...

fld.SetString( str )
```

Search Methods

```
ValueExists(
        inValue as Variant,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as Boolean
```

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

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

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

Example (Visual BASIC):

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

```
ValueExistsWithCount(
        inValue as Variant,
        ByRef outCount as Integer,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as Boolean
```

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

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

Example (Visual BASIC):

```
        Dim count As Integer
        found = fld.ValueExists( 5, count )
        found = fld.ValueExists( 5, count, S )
```

```
FindValue(
        inValue as Variant,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

If inSelection is null 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 (Visual BASIC):

```
        Dim s1 As VBitSet
        Dim s2 As VBitSet
        Set s1 = fld1.FindValue(5)
        Set s2 = fld2.FindValue( 7, s1 )
```

```
FindValueAsArraySet(
        inValue as Variant,
        inSelection as IVSet = null,
        inMaxCount as Integer = &hffffffff,
        inSearchPref as EVSearch = kPreferIndexed ) as IVArraySet
```

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

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

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

Example (Visual BASIC):

```
        Dim s1 As VArraySet
        Dim s2 As VArraySet
        Set s1 = fld1.FindValueAsArraySet(5)
        Set s2 = fld2.FindValueAsArraySet( 7, s1 )
```

```
FindRange(
        inLeftInclude as Boolean,
        inLeftValue as Variant,
        inRightValue as Variant,
        inRightInclude as Boolean,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

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

If inSelection is null 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 (Visual BASIC):

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

```
FindRangeAsArraySet(
        inLeftInclude as Boolean,
        inLeftValue as Variant,
        inRightValue as Variant,
        inRightInclude as Boolean,
        inSelection as IVSet = null,
        inMaxCount as Integer = &hffffffff,
        inSearchPref as EVSearch = kPreferIndexed ) as IVArraySet
```

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

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

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

Example (Visual BASIC):

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

```
FindSingleValue(
        inValue as Variant,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as Integer
```

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

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

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

Example (Visual BASIC):

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

```
FindDistinct(
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

Returns selection that contains only distinct values.

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

Example (Visual BASIC):

```
        Dim bset As VBitSet
        Set bset = fld.FindDistinct()
```

```
FindNulls(
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

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

Example (Visual BASIC):

```
        Dim bset As VBitSet
        Set bset = fld.FindNulls()
```

```
FindNotNulls(
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

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

Example (Visual BASIC):

```
        Dim bset As VBitSet
        Set bset = fld.FindNotNulls()
```

String Search Methods

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

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

```
FindStartsWith(
        inValue as String,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

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

Example (Visual BASIC):

```
Dim bset As VBitSet
Set bset = fld.FindStartsWith( "Jo" )
```

```
FindContains(
        inValue as String,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

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

Example (Visual BASIC):

```
Dim bset As VBitSet
Set bset = fld.FindContains( "Jo" )
```

```
FindEndsWith(
        inValue as String,
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

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

Example (Visual BASIC):

```
Dim bset As VBitSet
Set bset = fld.FindEndsWith( "hn" )
```

```
FindLike(
        inValue as String,
        inEscapeChar as String = "\",
        inSelection as IVSet = null,
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

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

Example (Visual BASIC):

```
Dim bset As VBitSet
Set bset = fld.FindLike( "%eter" )
```

```
FindRegEx (
        inValue as String,
        inSelection as IVSet = null
        inSearchPref as EVSearch = kPreferIndexed ) as IVBitSet
```

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

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

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

Example (Visual BASIC):

```
        Dim bset As VBitSet
        Set bset = fld.FindRegEx( "Pe?" )
```

Encryption Methods

The IVField class has a set of functions for encryption analog to functions of the IVDatabase and IVTable classes.

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

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

Encrypt(inKey as String)

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

Allows you to encrypt the separate Field in the table.

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

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

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

Example (Visual BASIC):

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

Decrypt(inKey as String)

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

Allows to decrypt the Field in the table.

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

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

Example (Visual BASIC):

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

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

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

Allows you to change the encryption key for the Field.

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

Example (Visual BASIC):

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

RequiresEncryptionKey() as Boolean

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

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

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

Example (Visual BASIC):

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

UseEncryptionKey(inKey as String)

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

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

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

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

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

Example (Visual BASIC):

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

fld.UseEncryptionKey( "key12345" )
```

Interface IVDate

Properties

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

Methods

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

Methods

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

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

Set the value of date field.

Example (Visual BASIC):

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

Interface IVTime

Properties

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

Methods

```
Set(
    inHour as Integer = 0,
    inMinute as Integer = 0,
    inSecond as Integer = 0 )
```

Methods

```
Set(
        inHour as Integer = 0,
        inMinute as Integer = 0,
        inSecond as Integer = 0 )
```

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

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

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

Example (Visual BASIC):

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

Interface IVDateTime

Properties

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

Methods

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

SetTime(
        inHour as Integer = 0,
        inMinute as Integer = 0,
        inSecond as Integer = 0 )
```

Value Methods

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

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

Sets the day, month and year.

Example (Visual BASIC):

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

```
SetTime(
        inHour as Integer = 0,
        inMinute as Integer = 0,
        inSecond as Integer = 0 )
```

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

Sets the time of day.

Example (Visual BASIC):

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

Interface IVString

Properties

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

Methods

```
SplitToWords(
        inStr as String ) as String
```

Interface IVString

Description

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

Properties

MaxLength as Integer

The Maximum length of a field can be in the range of values 1 .. 65535 bytes. It can be applied to VString 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 (Visual BASIC):

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

IndexByWords as Boolean

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

Example (Visual BASIC):

```
fldString.IndexByWords = True
```

Interface IVBLOB

Properties

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

Methods

```
DeleteData()

ReadData as String
WriteData( inData as String )

FromFile( inLocation as String )
ToFile( inLocation as String )
```

Description

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

Constructors of BLOB fields do not have parameter Flags.

Properties

DataSize as Integer (r/o)

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

Example (Visual BASIC):

```
Dim size As Long
size = fldBLOB.DataSize()
```

IsCompressed as Boolean

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

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

Example (Visual BASIC):

```
fldBlob.IsCompressed = True
```

SegmentSize as Integer (r/o)

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

Example (Visual BASIC):

```
segment = fldBlob.SegmentSize
```

Methods

DeleteData()

Deletes BLOB data of the field.

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

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

Example (Visual BASIC):

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

ReadData as String

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

Example (Visual BASIC):

```
Dim blobValue As String
blobValue = fldBLOB.readData()
```

WriteData(inData as String)

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

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

Example (Visual BASIC):

```
Dim s1 As String
s1 = “aaaaaa”                    // 6 chars
blob_fld.WriteData( s1 )
```

FromFile(inLocation as String)

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

Reads the whole file into the BLOB field.

Example (Visual BASIC):

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

ToFile(inLocation as String)

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

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

Example (Visual BASIC):

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

Interface IVText

Properties

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

Methods

```
SplitToWords(
        inStr as String ) as String
```

Description

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

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

String and Text fields can be searched using regular expressions.

Interface IVPicture

Properties

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

Methods

```
ReadPicture() as VARIANT

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

Description

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

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

Methods

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

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

Stores a Picture into VPicture field using the specified format.

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

This method expect that Picture is
DIB on Windows.

Example (Visual BASIC):

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

```
ReadPicture() as Picture
```

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

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

Example (Visual BASIC):

```
Dim pict As Variant
pict = fldPicture.ReadPicture()
```

Interface IVObjectPtr

Properties

Target	as IVTable

Method

```
ConvertFromRDB(
        inPrimaryKey as IVField,
        inForeignKey as IVField )
```

The field of the type ObjectPtr is intended to establish a “many to one” relation [M:1] between two Tables by ‘direct pointer’.

Note; In SQL this is called a FOREIGN KEY

It stores references to the related parent record (“One” record). The value of an ObjectPtr field is an unsigned long number (4 bytes, ulong) and it is the physical record number of the parent table. To set the Value of this field you must get the RecID of the record in the TargetTable:

```
mObjectPtr.Value = boPerson.GetRecID
```

Sometimes you may wish to relate a record of Table B to a non-current record of Table A, in this case you can save the RecID to a variable and use it later:

```
Dim RecID As Integer
RecID = TableA.GetRecID
TableA.GoToRecord( SomeOtherRecord )
...
TableB.TableA_Ptr.Value = RecID
```

• RecID is 1-based, zero is used for the ID of the undefined record.

The ObjectPtr field must know the pointed object (a parent object) and a deletion control to work correctly.

The Target must be defined when you create the field. There is no reason to change the Target at runtime.

The DeletionControl regulates a record deletion in the “Many” table when a record is deleted in the “One” table. It can be changed at runtime. This is the rule, which defines the behavior on deletion of a record. There are three methods for deleting records.

Leave related Many records:
From the database the record of the parent table is only deleted. The ObjectPtr field of the related child-records is automatically set to 0.

Delete related Many records:
The “One” and “Many “ components are all deleted. If a Many record also has some related Many records in a third Table, then they are also deleted in a cascade delete.

Can not delete if related Many:
The deletion of the One record is not allowed if there is at least one related Many record.

The ObjectPtr field can be used to establish a MANY to ONE relation, but it also can be used to establish a ONE to ONE relation. For this you should specify the ObjectPtr field as unique. Valentina can use this information to optimize a query.

Besides using the ObjectPtr field you can establish a Many to Many relation between two tables. For this you need to create an additional third table - Link as shown on the picture.

Properties

Target as IVTable

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 (Visual BASIC):

```
Set tbl = fldPtr.Target
```

```
ConvertFromRDB(
        inPrimaryKey as IVField,
        inForeignKey as IVField)
```

Parameter:	**Description:**
inPrimaryKey	The field of the target table that plays role of the PRIMARY KEY field.
inForeignKey	The field of the table of this ObjectPtr field that plays role of the FOREIGN KEY.

Converts a RDB-link between 2 tables into an ObjectPtr-link.

Example (Visual BASIC):

```
        fldPtr.ConvertFromRDB( fldPersonID, fldPersonPtr )
```

Interface IVCursor

Properties

BOF	as Boolean (r/o)	
EOF	as Boolean (r/o)	
DataBase	as IVDatabase	// (r/o) Database of this Cursor.
FieldCount	as Integer	// (r/o) number of selected fields for this Cursor.
Position	as Integer	
RecordCount	as Integer	// (r/o) Number of selected records, it can be reduced.
ReadOnly	as Boolean	// (r/o) TRUE if records can't be changed // i.e. you can't add/update/delete records.

Field Methods

Field(inNameOrIndex) as IVField

Navigation Methods

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

Record Methods

SetBlank()	// blank the memory buffer of the record
AddRecord() as Integer	// adds a new record to a cursor
UpdateRecord()	// updates the current records of the cursor
UpdateAllRecords()	// updates ALL records of a cursor with a new value.
DeleteRecord()	// deletes the current record of a cursor
DeleteAllRecords()	// deletes all records of a cursor
DropRecord()	// removes the current record from a cursor // but don't delete it from the original Table.

Import/Export Methods

```
ImportText(
        inFile as String,
        inFieldDelimiter as String = chr(09),
        inLineDelimiter as String = LE,
        inEncoding as String = "UTF-16",
        inHasColumHeader as Boolean = FALSE,
        inMaxRecordsToImport as Integer = 0 )

ExportText(
        inFile as String,
        inFieldDelimiter as String = chr(09),
        inLineDelimiter as String = LE,
        inEncoding as String ="UTF-16",
        inHasColumHeader as Boolean = FALSE )
```

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 different records.

Properties

BOF as Boolean (r/o)

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

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

Example (Visual BASIC):

```
Do
    ...
    curs.PrevRecord()
Loop Until curs.BOF
```

EOF as Boolean (r/o)

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

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

Example (Visual BASIC):

```
Do
    ...
    curs.NextRecord()
Loop Until curs.EOF
```

Database as IVDatabase

Returns the database of this cursor.

Example (Visual BASIC):

```
Set db = fld.Database
```

FieldCount as Integer

Returns the number of fields of this cursor.

Example (Visual BASIC):

```
fldCount = curs.FieldCount  // get local shortcut to avoid of calling in loop
For i = 1 To fldcount
    ...
Next
```

Position as Integer

The current position in the cursor. You can set or get the current position of cursor using this property.

The valid range of values is from 1 to the.

When you assign a new value to the Position, Valentina loads a record from the disk to the memory buffer.

Note: If you try to assign a wrong value then the current record is not changed.

Example (Visual BASIC):

```
For i = 1 To curs.RecordCount
    curs.Position = i
Next
```

RecordCount as Integer

Returns the number of records of cursor.

Example (Visual BASIC):

```
// store into a local variable to avoid of calling it loop
recCount = curs.RecordCount

For i = 1 To fldcount
        ...
Next
```

ReadOnly as Boolean

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

Example (Visual BASIC):

```
If( curs.ReadOnly )
        ....
```

Field Methods

Field(inNameOrIndex as VARIANT) as IVField

Parameter:	**Description:**
inNameOrIndex	The Name of the field or The Index of the field. Starts from 1.

You can use these method to access a field 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 (Visual BASIC):

```
Dim i As Integer
Dim  Records as Integer
Dim LastName As String
Dim cur As VCursor

Set cur = db.SqlSelect( “select * from person where name like ‘john’ no_case”)

Records = cur.RecordCount
For i = 1 To Records
        cur.CurrentPosition = i
        LastName = cur.Field( “last_name” ).GetString
Next
```

Navigation Methods

FirstRecord() as Boolean

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

Example (Visual BASIC):

```
res = curs.FirstRecord()
```

LastRecord() as Boolean

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

Example (Visual BASIC):

```
res = curs.LastRecord()
```

PrevRecord() as Boolean

Go to the previous record of a Cursor if it exists. Returns TRUE if the previous record is found. Otherwise, it returns FALSE and this means we are at the first logical record in the Cursor or the Cursor is empty.

Example (Visual BASIC):

```
res = curs.PrevRecord()
```

NextRecord() as Boolean

Go to the next logical record of a Cursor if it exists. Returns TRUE if the next record is found. Otherwise it returns FALSE, which means we are at the last logical record in the Cursor.

Example (Visual BASIC):

```
If( myCursor.FirstRecord() )
        Do
                // work here
        Loop    Until myCursor.NextRecord() = False
End If
```

You can also do this with the ‘Position property’ in conjunction with ‘RecordCount’, but NextRecord() is more efficient.

Example (Visual BASIC):

```
If( myCursor.RecordCount > 0 )
        myCursor.Position = 1
        For i = 1 To myCursor.RecordCount  // work here
                myCursor.Position = myCursor.Position + 1
        Next
End If
```

Record Methods

SetBlank()

Each Cursor has a RAM buffer for field values of the current record. This buffer can be cleared by the SetBlank() method, i.e. all numeric fields become zero, all string fields get the empty string. If a field is Nullable then it will get a NULL value.

Example (Visual BASIC):

```
curs.SetBlank()
        curs.LongField(1).Value = i
        curs.ShortField(2).Value = i
res = curs.AddRecord()
```

AddRecord() as Integer

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

Returns RecID of added records.

IMPORTANT: it returns RecID of original table where record was inserted! Valentina can do this because cursor that allows adding of new records always is built on single table.

Example (Visual BASIC):

```
curs.SetBlank()
        curs.LongField(1).Value = i
        curs.ShortField(2).Value = i
newRecID = curs.AddRecord()
```

UpdateRecord()

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

It throws error if a record cannot be updated, e.g. cursor is ReadOnly.

Example (Visual BASIC):

```
curs.currentRecord = i
        curs.LongField(1).Value = i + 100
        curs.ShortField(2).Value = i + 100
curs.UpdateRecord()
```

UpdateAllRecords()

Updates ALL records of a Cursor with new values. This function can update several fields of the cursor at once. Valentina will only update fields with new values (dirty fields). It is not important what record is current when you, assign new values.

This function is much faster than an iteration of the cursor records in a loop to assign new values.

It throws error if a record cannot be updated, e.g. cursor is ReadOnly.

Example (Visual BASIC):

```
curs.LongField(1).Value = 145
curs.ShortField(2).Value = 200

curs.UpdateAllRecords()
```

DeleteRecord()

Deletes the current record of a cursor. The next record becomes the current record. Otherwise the previous record becomes current. If a Cursor becomes empty then the current record is undefined.

Returns FALSE if the record cannot be deleted, e.g. it was locked or does not exist, or a cursor is read only.

Example (Visual BASIC):

```
curs.DeleteRecord()
```

DeleteAllRecords()

Deletes all records of the Cursor. The Cursor becomes empty, the current record becomes undefined.

Returns FALSE if the records cannot be deleted (e.g. cursor is ReadOnly).

Example (Visual BASIC):

```
curs.DeleteAllRecords()
```

DropRecord()

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

Example (Visual BASIC):

```
curs.DropRecord()
```

Import/Export Methods

```
ImportText(
        inFile as String,
        inFieldDelimiter as String = chr(09),
        inLineDelimiter as String = LE,
        inEncoding as String = ”UTF-16",
        inHasColumHeader as Boolean = FALSE,
        inMaxRecordsToImport as Integer = 0 )
```

Parameter:	**Description:**
inFile	File to be imported.
inFieldDelimiter	Character to be used as a field delimeter, default is a tab-chr(0x09).
inLineDelimiter	Character to be used as a record delimeter, default is the OS Line Ending.
inEncoding	Encoding of the imported file.
inHasColumnHeader	TRUE if the import file has a column header line.
inMaxRecordsToImport	The maximum number of records to import.

Imports the specified text file into the fields of the Cursor.

Note: The Cursor must have the flag CanBeUpdated set to TRUE.

The parameters FieldDelimiter and LineDelimiter are optional, i.e. you may specify one of them or both . By default they are TAB (09) and the OS Line Ending correspondingly.

If the cursor represents a subset of the table-fields, then the omitted fields will be filled with NULL values if the field in NULLABLE or blank values otherwise.

Importing text to a Cursor works for a single Table only.

Example (Visual BASIC):

```
curs.ImportText( fileToImport, chr(09), chr(13) )
```

```
ExportText(
        inFile as String,
        inFieldDelimiter as String = chr(09),
        inLineDelimiter as String = LE,
        inEncoding as String =”UTF-16",
        inHasColumHeader as Boolean = FALSE )
```

Parameter:	**Description:**
inFile	The file to be imported.
inFieldDelimiter	The character to be used as a field delimeter, default is tab-chr(0x09).
inLineDelimiter	The character to be used as a record delimeter, default is the OS Line Ending.
inEncoding	Encoding of the imported file.
inHasColumHeader	TRUE if import file has colum header line.

This command exports the fields and records of a Cursor to the designated text file. Using the SELECT statement, you can define the fields to export and their order, as well as the records to be exported.

Example (Visual BASIC):

```
        curs.ExportText( fileToExport, chr(09), chr(13) )
```

Interface IVSet

Properties

Count	as Integer (r/o)
IsSortedByRecID	as Boolean
IsEmpty	as Boolean (r/o)

Element methods

Append(inValue as Integer)
Remove(inValue as Integer)
Include(inValue as Integer) as Boolean

MakeNewIterator() as IVSetIterator
SortByRecID()

Properties

Count as Integer (r/o)

The number of items in the Set.

Example (Visual BASIC):

```
count = set1.Count
```

IsSortedByRecID as Boolean (r/o)

Returns TRUE if the Set is sorted by RecID values.

Example (Visual BASIC):

```
sorted = set1.isSortByRecID
```

IsEmpty as Boolean (r/o)

Returns TRUE if the Set is empty.

Example (Visual BASIC):

```
empty = set1.IsEmply
```

Element Methods

Append(inValue as Integer)

Parameter:	Description:
inValue	A value.

Appends a new value to the Set.

Example (Visual BASIC):

```
set.Append( rec )
```

Remove(inValue as Integer)

Parameter:	Description:
inValue	A value.

Removes the specified value from the Set.

Example (Visual BASIC):

```
set.Remove( rec )
```

Include(inValue as Integer) as Boolean

Parameter:	Description:
inValue	A value.

Returns TRUE if the Set contains the specified value.

Example (Visual BASIC):

```
found = set.Include( rec )
```

MakeNewIterator() as IVSetIterator

Creates and returns a new Iterator for this Set.

Example (Visual BASIC):

```
Set iter = s1.MakeNewIterator()
```

SortByRecID()

Sorts the Set.

Example (Visual BASIC):

```
s1.SortByRecID()
```

Interface IVArraySet

Methods

InitWithCount(inCount as Integer)
InitWithArraySet(inArraySet as IVArraySet)
InitWithBitSet(inBitSet as IVBitSet)

GetItemAt(inPosition as Integer) as Integer
SetItemAt(inPosition as Integer, Assigns inValue as Integer)

Set operations

Union	(inRightSet as IVArraySet) as IVArraySet
Intersection	(inRightSet as IVArraySet) as IVArraySet
Difference	(inRightSet as IVArraySet) as IVArraySet
SymmetricDifference	(inRightSet as IVArraySet) as IVArraySet

Methods

InitWithCount(inCount as Integer)

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

Constructor. Creates an ArraySet with the specified reserved size.

Note: inCount is not the maximum limit. It is just an initial size. If the ArraySet will require more space then it reallocates more RAM automatically.

Example (Visual BASIC):

```
Dim as1 As New VArraySet
as1.InitWithCount( 50 )
```

InitWithArraySet(inArraySet as IVArraySet)

Parameter:	**Description:**
inArraySet	Another ArraySet

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

Example (Visual BASIC):

```
Dim as2 As New VArraySet
as2.InitWithArraySet( as1 )
```

InitWithBitSet(inBitSet as IVBitSet)

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

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

Example (Visual BASIC):

```
Dim as3 As New VArraySet
as3.InitWithBitSet( bitSet1 )
```

Methods

GetItemAt(inPosition as Integer) as Integer

Parameter:	**Description:**
inPosition	Position of item in the array set.

Returns the item of the set at the specified position.

Example (Visual BASIC):

```
recID = as1.GetItemAt( 5 )
```

SetItemAt(inPosition as Integer, inValue as Integer)

Parameter:	**Description:**
inPosition	Position of item in the array set.
inValue	A value.

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

Example (Visual BASIC):

```
as1.SetItemAt( 5, recID)
```

Set Methods

Union(inRightSet as IVArraySet) as IVArraySet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

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

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

Example (Visual BASIC):

```
s1.Union( s2 )
```

Intersection(inRightSet as IVArraySet) as IVArraySet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

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

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

Example (Visual BASIC):

```
s1.Intersection( s2 )
```

Difference(inRightSet as IVArraySet) as IVArraySet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

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

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

Example (Visual BASIC):

```
s1.Difference( s2 )
```

SymmetricDifference(inRightSet as IVArraySet) as IVArraySet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

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

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

Example (Visual BASIC):

```
s1.SymmetricDifference( s2 )
```

Interface IVBitSet

Methods

InitWithCount(inMaxCount as Integer)
InitWithArraySet(inMaxCount as Integer, inArraySet as IVArraySet)

Set operations

Union	(inRightSet as IVBitSet) as IVBitSet
Intersection	(inRightSet as IVBitSet) as IVBitSet
Difference	(inRightSet as IVBitSet) as IVBitSet
SymmetricDifference	(inRightSet as IVBitSet) as IVBitSet

Methods

InitWithCount(inMaxCount as Integer)

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

Constructor. Creates a BitSet of the specified size.

Example (Visual BASIC):

```
Dim bs1 As New VBitSet
bs1.InitWithCount( 50 )
```

InitWithArraySet(inMaxCount as Integer, inArraySet as IVArraySet)

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

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

Example (Visual BASIC):

```
Dim bs2 As New VBitSet
bs2.InitWithArraySet( as1 )
```

Set Methods

Union(inRightSet as IVBitSet) as IVBitSet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

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

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

Example (Visual BASIC):

```
s1.Union( s2 )
```

Intersection(inRightSet as IVBitSet) as IVBitSet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

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

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

Example (Visual BASIC):

```
s1.Intersection( s2 )
```

Difference(inRightSet as IVBitSet) as IVBitSet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

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

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

Example (Visual BASIC):

```
s1.Difference( s2 )
```

SymmetricDifference(inRightSet as IVBitSet) as IVBitSet

Parameter:	**Description:**
inRightSet	The set to be used in the operation.

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

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

Example (Visual BASIC):

```
s1.SymmetricDifference( s2 )
```

Interface IVSetIterator

Properties

Value	as Integer (r\o)

Methods

FirstItem() as Integer
LastItem() as Integer
NextItem() as Integer
PrevItem() as Integer

Properties

Value as Integer (r\o)

Returns the current value of the iterator.

Example (Visual BASIC):

```
v = iter.Value
```

Methods

FirstItem() as Integer

Moves the iterator to the first item of the Set.
Returns the value of the item if it is found, else returns 0.

Example (Visual BASIC):

```
v = iter.FirstItem
```

LastItem() as Integer

Moves the iterator to the last item of the Set.
Returns the value of the item if it is found, else returns 0.

Example (Visual BASIC):

```
v = iter.LastItem
```

NextItem() as Integer

Moves the iterator to the next item of the Set.
Returns the value of the item if it is found, else returns 0.

Example (Visual BASIC):

```
v = iter.NextItem
```

PrevItem() as Integer

Moves the iterator to the prev item of the Set.
Returns the value of the item if it is found, else returns 0.

Example (Visual BASIC):

```
v = iter.PrevItem
```

Interface IVLink

Properties

BranchCount	as Integer (r/o)
ID	as Integer (r/o)
IsTemporary	as Boolean (r/o)
Name	as String
OnDelete	as EVOnDelete
OnUpdate	as EVOnUpdate
Owner	as IVTable

Table methods

```
IsBetween(
        inTableA as IVTable,
        inTableB as IVTable ) as Boolean

Table( inIndex as Integer ) as IVTable

Flush( inFlushTables as Boolean = true )
```

Search methods

```
FindLinked(
        inRecID as Integer,
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild ) as IVArraySet

FindLinkedAsBitSet(
        inSet as IVSet,
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild ) as IVBitSet

FindExclusivelyLinked(
        inRecID as Integer,
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild ) as IVArraySet

FindAllLinked(
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild ) as IVBitSet
```

Linking methods

```
CountLinked(
        inRecID as Integer,
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild ) as Integer

LinkRecords( inRecID() as Variant )

UnlinkRecords( inRecID() as Variant )

DeleteLinkedRecords(
        inRecID as Integer,
        inTableA as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )

DeleteAllLinkedRecords( inTableA as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )

IsLinked( inLeftRecID as Integer, inRightRecID as Integer ) as Boolean
```

Properties

BranchCount as Integer (r/o)

Returns the number of branches for this link.

Example (Visual BASIC):

```
brc = Link.BranchCount
```

ID as Integer (r/o)

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

Example (Visual BASIC):

```
link_id = Link.ID
```

IsTemporary as Boolean (r/o)

Returns TRUE if this link is temporary.

Example (Visual BASIC):

```
tmp = Link.IsTemporary
```

Name as String

Returns the name of the link.

Example (Visual BASIC):

```
s = L ink.Name
```

OnDelete as EVOnDelete

The behavior on deletion of the record-owner.

Example (Visual BASIC):

```
v = Link.OnDelete
```

OnUpdate as EVOnUpdate

The behavior on update of the record-owner.

Example (Visual BASIC):

```
v = Link.OnUpdate
```

Owner as IVTable

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

Note, you need specify this property only if you are going to use the DeletionControl for this link.

Example (Visual BASIC):

```
Link.Owner = tblPerson
```

Table Methods

```
IsBetween(
        inTableA as IVTable,
        inTableB as IVTable ) as Boolean
```

Parameter:	**Description:**
inTableA	Left table of link.
inTableB	Right table of link.

Returns TRUE if this Link links both specified Tables.

Example (Visual BASIC):

```
res = Link.IsBetween( TablA, TablB )
```

```
Table( inIndex as Integer ) as IVTable
```

Parameter:	**Description:**
inIndex	The index of table.

Returns a table of the link by index.

Example (Visual BASIC):

```
Set tbl = Link.Table( i )
```

```
Flush( inFlushTables as Boolean = true )
```

Parameter:	**Description:**
inFlushTables	TRUE if Tables of Link also should flush.

Flushes new or modified information of Link. On default it also pass flush() command to Tables of Link. You can set parameter to be FALSE, in this case Tables are not touched.

Example (Visual BASIC):

```
Set tbl = Link.Flush()
```

Search Methods

```
FindLinked(
        inRecID as Integer,
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as IVArraySet
```

Parameter:	**Description:**
inRecID	The RecID of a record of the left table.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns the records from inTableB linked to record with inRecID from inTableA. If zero records are found then returns Null.

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 (Visual BASIC):

```
        Set res = Link.FindLinked( rec, TblA, TblB )
```

```
FindLinkedAsBitSet(
        inSet as IVSet,
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as IVArraySet
```

Parameter:	**Description:**
inSet	Selection of records.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns the records from inTableB linked to any record specified by inSet from inTableA. If zero records are found then returns Null.

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 (Visual BASIC):

```
        Set res = Link.FindLinked( rec, TblA, TblB )
```

```
FindExclusivelyLinked(
        inRecID as Integer,
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as IVArraySet
```

Parameter:	**Description:**
inRecID	The RecID of a record of the left table.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns the records from inTableB linked to the record inRecID of inTableA and only to it. If zero records are found then returns Null.

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 (Visual BASIC):

```
        Set res = Link.FindExclusivelyLinked( rec, TblA, TblB )
```

```
FindAllLinked(
        inTableA as IVTable,
        inTableB as IVTable,
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as IVArraySet
```

Parameter:	**Description:**
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

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

Example (Visual BASIC):

```
        Set res = Link.FindAllLinked( TblA, TblB )
```

Linking Methods

```
CountLinked(
        inRecID as Integer,
        inTableA as IVTable,
        inTableB as IVTable
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
as Integer
```

Parameter:	Description:
inRecID	The RecID of a record of the left table.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns the number of records of table inTableB linked to the record inRecID of table inTableA.

For a recursive link you should specify the parameter inRecursionDirection. If you specify kFromParentToChild then the function will use child records of the inRecID record. Otherwise it will use parent record(s) of the inRecID record.

Example (Visual BASIC):

```
        res = Link.CountLinked( rec, TblA, TblB )
```

LinkRecords(inRecID() as Integer)

Parameter:	Description:
inRecID	The RecID of a record of the left table.

Establishes a link between records of linked Tables, specified as an array of RecID values (Valentina 2.0 supports 2-branch links only, so 2 records must be specified).

The array must contains the correct number of values, in the order of branches of this link. The order of branches corresponds to the order of Tables on link creation.

Example (Visual BASIC):

```
Dim recs(1) as Variant                    // allocate array with 2 items.

// Link record 1 of the left table to record 3 of the right table of the Link.
recs(0) = 1
recs(1) = 3
Link.LinkRecords( recs )
```

Example (Visual BASIC):

```
// The same task in syntax:
Link.LinkRecords( Array(1, 3) )
```

UnlinkRecords(inRecID() as Variant)

Parameter:	Description:
inRecID	The RecID of a record of the left table.

Breaks the link between records of the linked Table specified as an array of RecID values.

The array must contain the correct number of values, in the order of branches of this link. The order of branches corresponds to the order of Tables on link creation.

Example (Visual BASIC):

```
Link.UnlinkRecords( Array(1, 3) )
```

DeleteLinkedRecords(

```
DeleteLinkedRecords(
        inRecID as Integer,
        inTableA as IVTable
        inRecursionDirection as EVRecursionDirection = kFromParentToChild )
```

Parameter:	**Description:**
inRecID	The RecID of a record of the left table.
inTableA	Left table of link.
inRecursionDirection	The direction of movement for a recursive link.

Removes all records that are linked by this Link to the record inRecID of table inTableA.

The action of this function depends on the DeletionControl parameter of the link, which can be { refuse, delete some records, update some records }.

ERRORS: errRestrict.

Example (Visual BASIC):

```
        Link.DeleteLinkedRecords( rec, TblA )
```

DeleteAllLinkedRecords(inTableA as IVTable)

Parameter:	**Description:**
inTableA	Left table of link.

Removes all records linked by this Link to the any record of table inTableA.

The action of this function depends on the DeletionControl parameter of the link, which can be { refuse, delete some records, update some records }.

ERRORS: errRestrict.

Example (Visual BASIC):

```
        Link.DeleteAllLinkedRecords( TblA )
```

IsLinked(inLeftRecID as Integer, inRightRecID as Integer) as Boolean

Параметр:	**Описание:**
inLeftRecID	The RecID of a record of the left table.
inRightRecID	The RecID of a record of the right table.

Returns TRUE, if the two specified records are linked.

Example (Visual BASIC):

```
        res = Link.IsLinked( 3, 2 )
```

Interface IVLink2

Properties

LeftType	as EVLinkType (r/o)
RightType	as EVLinkType (r/o)

Properties

LeftType as EVLinkType (r\o)

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

Example (Visual BASIC):

```
lt = Link.LeftType
```

RightType as EVLinkType (r\o)

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

Example (Visual BASIC):

```
rt = Link.RightType
```

Interface IVServer

Only for Client.

Properties

Available	as Integer	// (r/o) Returns TRUE if the server is available.
ConnectionCount	as Integer	// (r/o) The number of active connections to a server.
DatabaseCount	as Integer	// (r/o) The number of databases that the server recognizes.
HostName	as String	// (r/o) The name of the host where the server is located.
Port	as Integer	// (r/o) Returns the port number of the server host.
UserName	as String	// (r/o) The name of the current user.
UserCount	as Integer	// (r/o) Returns the count of registered users.
Version	as String	// (r/o) Version of the server.

Methods

```
Init(
        inHost as String,
        inUserName as String,
        inUserPassword as String,
        inPort as Integer = 15432,
        inTimeOut as Integer = 5 )
```

Connection Methods

```
OpenSession()
CloseSession()

Restart()
Shutdown()
CancelConnection ( inConnectionID as Integer )
Refresh()
```

INI-File Methods

```
GetVariable( inName as String ) as String
SetVariable( inName as String, inValue as String )
```

Master databases methods

```
RegisterDatabase( inDbName as String, inServerPath as String = "" )
UnregisterDatabase( inDbName as String ) as Boolean
```

User Methods

```
AddUser(
        inUserName as String,
        inPassword as String,
        isAdmin as Boolean = False )

RemoveUser(inUserName as String)

ChangeUserPassword(
        inUserName as String,
        inNewPassword as String)

GetUserName( inUserIndex as Integer) as String
GetUserIsAdmin( inUserIndex as Integer) as Boolean
```

DatabaseInfo methods

```
DatabaseInfo( inIndex as Integer) as IVDatabaseInfo
```

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

Available as Integer (r/o)

Returns TRUE if the server is available.

Example (Visual BASIC):

```
res = server.Available
```

ConnectionCount as Integer (r/o)

Returns the number of all active connections to the server.

Example (Visual BASIC):

```
connCount = server.ConnectionCount
```

DatabaseCount as Integer (r/o)

Returns the number of databases that a server knows about. In other words, this is the number of databases registered in the Master Database of the VServer.

Example (Visual BASIC):

```
dbCount = server.DatabaseCount
```

HostName as String (r/o)

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

Example (Visual BASIC):

```
version = server.HostName
```

Port as Integer (r/o)

Returns the port number of the server host.

Example (Visual BASIC):

```
port = server.Port
```

UserName as String (r/o)

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 (Visual BASIC):

```
userName = server.UserName
```

UserCount as Integer (r/o)

Returns the number of registered users.

Example (Visual BASIC):

```
count = server.UserCount
```

Version as String (r/o)

Returns a string that contains the VServer version number.

Example (Visual BASIC):

```
version = server.Version
```

Creation of VServer

```
VServer(
        inHost as String,
        inUserName as String,
        inUserPassword as String,
        inPort as Integer = 15432,
        inTimeOut as Integer = 5 )
```

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.

This method 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 (Visual BASIC):

```
        Dim server As New VServer
        server.Init( "localhost", "sa", "sa" )
```

Connection Methods

OpenSession()

Establishes a connection to the server.

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

Example (Visual BASIC):

```
Dim server As VServer
server.Init( "localhost", "sa", "sa" )
server.OpenSession()
```

CloseSession()

Closes the connection with the server .

Example (Visual BASIC):

```
Dim server As VServer
server.Init( "localhost", "sa", "sa" )
server.OpenSession
...
server.CloseSession()
```

CancelConnection(inConnectionID as Integer)

Parameter	Description
inConnectionID	The connection ID.

Cancels an existing connection by its ID.

Example (Visual BASIC):

```
server.CancelConnection( connID )
```

Restart()

Forces a restart of the VServer.

Example (Visual BASIC):

```
server.Restart()
```

Refresh()

This method allows you to refresh the list of DatabaseInfo objects. This method sends a request to the Valentina Server.

Example (Visual BASIC):

```
server.Refresh()
```

Shutdown()

Shuts down the VServer.

Note: After this operation there is no way to restart VServer from the application. If you want to restart the VServer, use Restart().

Example (Visual BASIC):

```
server.Shutdown()
```

INI-File Methods

GetVariable(inName as String) as String

Parameter:	Description:
inName	The name of server variable.

This method allows you to read a value of the specified Server Variable. The name of the variable is case insensitive. With names of variables you can use constants of the INI-file of VServer. For more information, refer to the Valentina Server documentation.

Example (Visual BASIC):

```
cache = server.GetVariable( "CacheSize" )
```

SetVariable(

```
SetVariable(
        inName as String,
        inNewValue as String )
```

Parameter:	Description:
inName	The name of the server variable.
inNewValue	New value for this variable.

This method allows you to change a value of the specified Server Variable. The name of variable is case insensitive. With names of variables you can use constants of the INI-file of VServer. For more information, refer to the Valentina Server documentation.

NOTE: Some variables require a restart of VServer to affect changes.

Example (Visual BASIC):

```
server.SetVariable( "CacheSize",  8 )
```

Master Database Methods

```
RegisterDatabase(
        inDbName as String,
        inServerFullPath as String = "" )
```

Parameter:	**Description:**
inDbName	The name of the database.
inServerFullPath	The full path of the database located on the server computer.

You can use this method to register in Vserver some existed database. This command adds a new record to the Master Database.

Usually you need just to drop a database into the folder pointed by .ini varable "System-Catalog", and call this method specifying only the name of database. Also it is possible to specify the full path of database on the server computer.

Note: For a MacOS X version of Valentina Server, use a UNIX path.

Errors:

The Database Name already exists.

Example (Visual BASIC):

```
server.RegisterDatabase( "DbName"  )
```

This assumes that a database with name "DbName" or "DbName.vdb" exists in the "data-bases" folder of VServer.

Example (Visual BASIC):

```
server.RegisterDatabase( "Accounting", "C:\SomeCompany\account2002.vdb" )
```

```
UnregisterDatabase( inDbName as String ) as Boolean
```

Parameter:	**Description:**
inDbName	The name of a database.

If you want to remove some database from the scope of the VServer, you need to remove the record about it from the Master Database. You can do this using this method.

Errors:

Database Name not found.

Example (Visual BASIC):

```
server.UnregisterDatabase( "Accounting" )
```

User Methods

```
AddUser(
        inUserName as String,
        inPassword as String,
        isAdmin as Boolean = False )
```

Parameter:	**Description:**
inUserName	The user name.
inPassword	The password for this user.
IsAdmin	TRUE if this user has administrator permissions.

An Administrator can add new users to the Master Database.

Errors:

The user name already exists.

Example (Visual BASIC):

```
server.AddUser( "Peter", "a1234fteg4" )
```

```
RemoveUser( inUserName as String )
```

Parameter:	**Description:**
inUserName	The user name.

An administrator can remove users from the Master Database.

Errors:

The user name is not found.

Example (Visual BASIC):

```
server.RemoveUser( "Peter" )
```

ChangeUserPassword(inUserName as String, inNewPassword as String)

```
ChangeUserPassword(
    inUserName as String,
    inNewPassword as String )
```

Parameter:	**Description:**
inUserName	The user name.
inNewPassword	New password for this user.

An administrator can change the password of a user.

Errors:

The user name is not found.

Example (Visual BASIC):

```
server.ChangeUserPassword( "Peter", "rvsa3341" )
```

GetUserName(inUserIndex as Integer) as String

Parameter	**Description**
inUserIndex	The user index.

Returns the name of the user by index.

Example (Visual BASIC):

```
server.GetUserName()
```

GetUserIsAdmin(inUserIndex as Integer) as Boolean

Parameter:	**Description:**
inUserIndex	The user index.

Returns TRUE if the specified user is an administrator.

Example (Visual BASIC):

```
res = server.GetUserIsAdmin( i )
```

DataBaseInfo Methods

DatabaseInfo(inIndex as Integer) as IVDatabaseInfo

Parameter:	**Description:**
inIndex	1-based index

This method allows you to iterate through the collection of DatabaseInfo objects.

The Vserver instance obtains a list of the DatabaseInfo upon OpenSession(). You can periodically refresh this list using the Refresh() method.

Example (Visual BASIC):

```
Dim dbi As VDatabaseInfo

For i = 1 To server.DatabaseCount
        dbi = server.DatabaseInfo( i )
        ....
Next
```

Interface IVDatabaseInfo

Only for VCOM Client.

Properties

```
ClientCount   as  Integer    // (r/o) The number of connected clients.
CursorCount   as Integer     // (r/o) The number of cursors currently on this database.
Name          as String      // (r/o) The name of the database.
Path          as String      // (r/o) The full path of the database on the server.
```

Methods

```
ClientInfo( inIndex as Integer ) as IVClientInfo

Refresh()
```

Methods

ClientInfo(inIndex as Integer) as IVClientInfo

Parameter:	Description:
inIndex	The index of ClientInfo object.

This method allows you to iterate through the collection of ClientInfo objects.

The object of a DatabaseInfo gets the list of ClientInfo on its creation. You can periodically refresh this list using the Refresh() method.

Example (Visual BASIC):

```
Dim ci As VClientInfo

For i = 1 To dbi.ClientCount
        ci = dbi.DatabaseInfo
        ....
Next
```

Refresh()

This method allows you to refresh the list of ClientInfo objects. This method sends a request to the Valentina Server.

Example (Visual BASIC):

```
dbi.Refresh()
```

Interface IVClientInfo

Only for a VCOM Client.

Properties

Address	as String	// (r/o) The IP address of the client computer.
ConnectionID	as Integer	// (r/o) The ID of this connection.
CursorCount	as Integer	// (r/o) The number of cursors of this connection.
Login	as String	// (r/o) The login of this connection.
Port	as Integer	// (r/o) The port number of the client computer..

Methods

Refresh()