

# **VALENTINA COM**

## **Reference**

Paradigma ([www.paradigmasoft.com](http://www.paradigmasoft.com))  
© 1999-2003

### **Acknowledgments:**

**Andy Bachorski, Andy Fuchs, Bill Mounce, Brian Blood, Craig A. Berry,  
David A. Bayly, Frank J. Schima, Guillermo Zaballa, Hideaki Iimori,  
John Roberts, Jochen Peters, Lynn Fredricks, Paul Shaap, Robert Brenstein.**

---

---

# Contents

<b>IValentina</b> .....	<b>3</b>
<b>Properties</b> .....	<b>3</b>
<b>Class IVDataBase</b> .....	<b>6</b>
<b>Class description</b> .....	<b>7</b>
<b>Properties description</b> .....	<b>7</b>
<b>Methods description</b> .....	<b>8</b>
<b>Database structure methods</b> .....	<b>10</b>
<b>BaseObject methods</b> .....	<b>11</b>
<b>SQL Methods</b> .....	<b>12</b>
<b>Encryption Methods</b> .....	<b>14</b>
<b>Dump methods</b> .....	<b>16</b>
<b>Class IVBaseObject</b> .....	<b>17</b>
<b>Class description</b> .....	<b>19</b>
<b>Record Methods</b> .....	<b>19</b>
<b>Navigation Methods</b> .....	<b>21</b>
<b>Field Methods</b> .....	<b>23</b>
<b>Database structure methods</b> .....	<b>24</b>
<b>Encryption methods</b> .....	<b>28</b>
<b>Class IVField</b> .....	<b>29</b>
<b>Properties Description</b> .....	<b>31</b>
<b>Encryption methods</b> .....	<b>39</b>
<b>BLOB Methods</b> .....	<b>40</b>
<b>Class IVCursor</b> .....	<b>42</b>
<b>Class description</b> .....	<b>43</b>
<b>Properties description</b> .....	<b>43</b>
<b>Field Methods</b> .....	<b>45</b>
<b>Navigation Methods</b> .....	<b>46</b>
<b>Record Methods</b> .....	<b>47</b>
<b>Import/Export</b> .....	<b>50</b>
<b>Class VServer</b> .....	<b>51</b>
<b>Class description</b> .....	<b>52</b>
<b>Properties description</b> .....	<b>52</b>
<b>Creation of VServer</b> .....	<b>54</b>
<b>Class VDatabaseInfo</b> .....	<b>59</b>
<b>Class VClientInfo</b> .....	<b>60</b>

## IValentina

### Properties

---

#### DebugLevel

Sets the level of debugging for Valentina.

Its value can be one of the following:

- 0 - no debug messages.
- 1 - message only if error was occurred.
- 2 - each function will produce message to the console window.

If on your MacOS computer is installed “**DCon**” which is included into MondoToConsole Package at <http://homepage.mac.com/vanhoek/>, or on your Windows computer is installed analog utility “**DbgView**” <http://www.sysinternals.com/dbgview.htm> then you can collect debugging messages from VCOM in the console window. Advnatage of this method is that even if VisualBasic crashes you still see log of execution, you see point of crash and you have time to think. Also this window will show you any Valentina error even if you have no check them explicitly.

#### **Example:**

```
Valentina.DebugLevel = 2
```

- DO NOT FORGET to set the debugging level to zero for final release!

## IValentina

---

`Init( inCacheSize as Integer,  
inSerialNumber as String )`

<b>Parameter:</b>	<b>Description:</b>
<code>inCacheSize</code>	The size of the database cache in MB.
<code>inSerialNumber</code>	The serial number for use under Windows or "" for demo mode.

To improve disk access, Valentina uses a cache mechanism. In the `ValentinaInit` method, you must define the size of the cache. You can place this parameter in the preferences of your application, so a user can define it manually. By default, it is a good idea to allocate half of free memory to the cache.

If you are registered user then specify your serial number. Otherwise pass the empty string, and Valentina will work in the time limited mode - 10 minutes per launch. After 10 minutes any request to the database will be ignored and Valentina will make 3 beeps.

### **Example:**

```
err = Valentina.Init( 5, "" )
```

---

### `ShutDown`

Destroys cache and other allocations of Valentina kernel.

### **Example:**

```
Valentina.Init( 5, "", "" )  
.....// some work here  
Valenitna.ShutDown()
```

## IValentina

---

[SetExtensions\(inDesc as string, inDat as string, inBlb as string, inInd as string\)](#)

<b>Parameter:</b>	<b>Description:</b>
inDesc	A new extension for ".vdb" file of Valentina
inDat	A new extension for ".dat" file of Valentina
inBlb	A new extension for ".blb" file of Valentina
inInd	A new extension for ".ind" file of Valentina

You can call this function BEFORE open/create database to inform Valentina kernel about extensions it must use for database files.

### **Example:**

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

---

[ValentinaEscapeString\(inStr as string, inForRegEx as Boolean\) As String](#)

<b>Parameter:</b>	<b>Description:</b>
inStr	The string you want to escape.
inForRegEx	Specify how you want escape string: for REGEX or for LIKE.

Utility function that allow you escape string (usually from user input) before you will concatenate that string into SQL query.

If inForRegEx parameter is false then only single quote will be escaped.

Otherwise will be escaped all chars that use RegEx syntax.

### **Example:**

```
res = ValentinaEscapeString( "Valentina's (day)", 0 )  
// res is "Valentina\'s (day)"  
  
res = ValentinaEscapeString( "Valentina's day", 1 )  
// res is "Valentina\'s \((day\)"
```

## Class IVDataBase

### properties

Name	as String	[r/o]	// The name of database.
Path	as String	[r/o]	// The full path of database.
SchemaVersion	as integer	[r/w]	// Version of db Schema
IsEncrypted	as boolean	[r/o]	// TRUE if the database is encrypted.
BaseObjectCount	as Integer	[r/o]	
DateFormat	as Integer	[r/w]	// specifies the format of date: // 0 - M/D/Y, 1 - D/M/Y, 2 - Y/M/D
DateSep	as String	[r/w]	// separator for date, e.g. '/'
TimeSep	as String	[r/w]	// separator for time, e.g. ':'
CenturyBound	as Integer	[r/w]	// default 20.

### methods

#### // Disk files methods

Create(inPath as String, [inMode as Long], [inSegmentSize as Long] )

Open(inPath as String)

Close()

Flush()

#### // BaseObject methods

BaseObject( inIndexOrName as VARIANT ) as IVBaseObject

#### // Database structure methods

CreateBaseObject( inName as String ) as IVBaseObject

DeleteBaseObject( inBaseObject as IVBaseObject )

#### // SQL Methods

SQLSelect( inQuery as String ) as IVCursor

SQLExecute( inQuery as String, [Binds() as String] ) as Long

#### // Dump Methods

Dump(inDumpFile as String, inDumpType as DumpTypeEnum)

LoadDump( inDumpFile as String, inNewDb as String, inDumpType as DumpTypeEnum)

#### // Encryption Methods

ChangeStructureEncryption(inOldKey as String,inNewKey as String)

SetStructureEncryption(inKey as String)

ChangeEncryption(inOldKey as String,inNewKey as String)

SetEncryption(inKey as String)

---

## Class description

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

## Properties description

### Name

Contains the name of database file.  
For example: "customers.vdb"

### Path

Contains the full path of database.  
For example: "C:\work\databases\customers.vdb"

### SchemaVersion

Number of version of database schema. Initial value is 1.  
Can be used if you want to change database structure in the new version of your application.

### IsEncrypted

You can use this property of database to manage own version for database schema.

### BaseObjectsCount

You can use it to get the number of BaseObjects (Tables) in this database. Then you can iterate BaseObjects by index using method BaseObject().

### DateFormat

### DateSep

### TimeSep

### CenturyBound

Look into ValentinaKernel.pdf file for description of this properties.

---

## Methods description

---

### Create(

Location as FolderItem,  
 [Mode as Long = 4],  
 [SegmentSize as Long = 32\*1024] )

<b>Parameter:</b>	<b>Description:</b>
Location	The full path to the database on the disk.
Mode	How many files for databases will be used, range 1-8, default 4.
SegmentSize	The size of one cluster in the database file, default 32KB.

Creates a new empty database on the hard disk. Returns true on success. After creation database is open already.

As the Mode parameter you can specify one of:

- 1 // (description,data,BLOB,indexes)
- 2 // description + (data,BLOB,indexes)
- 3 // description + (data,BLOB) + indexes
- 4 // description + data + BLOB + indexes
- 5 // (description,data,BLOB) + indexes
- 6 // (description,data) + BLOB + indexes
- 7 // (description,data,indexes) + BLOB
- 8 // description + (data,indexes) + BLOB

### Example:

```
db.Create( file, 5, 32 * 1024 )
db.Create( file, 5 ) // last 2 parameters can be skipped.
```

---

### Open(Location as FolderItem)

<b>Parameter:</b>	<b>Description:</b>
Location	The full path to the database on the disk.

Opens existing database at specified location.

### Example:

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

[Close\(\)](#)

Closes the database.

**Example:**

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

---

[Flush\(\)](#)

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

**Example:**

```
db.Flush
```

---

## Database structure methods

---

### CreateBaseObject( inName as String ) as IVBaseObject

---

<b>Parameter:</b>	<b>Description:</b>
inName	Name of new BaseObject.

Creates a new empty Table in the database.  
You need to add columns to this table using IVBaseObject.CreateField() method.

**Example:**

```
dim BaseObject as IVBaseObject  
  
BaseObject = db.CreateBaseObject( "Person" )
```

---

### DeleteBaseObject( inBaseObject as IVBaseObject )

---

<b>Parameter:</b>	<b>Description:</b>
inBaseObject	reference of IVBaseObject to delete.

Removes the specified BaseObject (Table) from the database. This operation is undoable and instantaneous!

**Example:**

```
db.DeleteBaseObject( BaseObject )
```

---

## BaseObject methods

---

`BaseObject( inIndexOrName as VARIANT ) as IVBaseObject`

**Parameter:**

Index

**Description:**

index of BaseObject in database, start from 1.

Returns a BaseObject by index or by name.

**Example:**

```
BaseObject = db.BaseObject( i )
```

**Example:**

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

---

## SQL Methods

---

### SQLSelect(

inQuery as String  
 inLocation as integer,  
 inLockType as integer,  
 inDirection as integer ) as IVCursor

Parameter:	Description:
Query	The SQL string of query.
[inLocation]	Location of cursor.
[inLockType]	Lock type for records of cursor.
[inDirection]	Direction of cursor.

Valentina uses SQL to search a database. See document ValentinaSQL.pdf on information about SQL supported by Valentina.

The SQLSelect method is given a string as a parameter (this is a SQLstring specifying what is to be searched for), resolves it, and returns the resulting table as a cursor of type IVCursor. When you finish working with the cursor, you must assign it the value nil to destroy it and free memory.

If you pass wrong SQL string then cursor will be empty (no records). You should always check Valentina's error after query to see if operations was successful.

The last 3 optional parameters allow you to control behavior of cursor. See ValentinaKernel.pdf and VServer.pdf for more details.

This parameters can get the next values:

inLocation: kClient =1, kServer =2  
 inLockType: kNoLock =1 , kReadOnly =2 , kReadWrite = 3  
 inDirection: kForwardOnly = 1, kRandom = 2

On default the last 3 parameters get values:

kClient, kReadOnly, kForwardOnly

### Example:

```
dim curs as IVCursor
cur = db.SQLSelect( "SELECT * FROM T " )
```

### Example:

```
dim curs as VCursor
cur = db.SQLSelect( "SELECT * FROM T ",
                  kServer, kReadWrite, kRandom )
```

---

---

`SQLExecute( Query as String, [Binds() as String] ) as Long`

<b>Parameter:</b>	<b>Description:</b>
Query	The SQL string of query.
Binds()	Optional array of strings -- binded values.

You can use this function to execute any SQL command supported by Valentina except a command that returns cursor as result (e.g. SELECT). See document ValentinaSQL.pdf on information about SQL supported by Valentina.

Returns the number of affect rows.

For commands INSERT and UPDATE you can specify list of strings binded to query. See ValentinaSQL.pdf for details.

**Example:**

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

**Example:**

```
dim Binds() as String  
  
Binds.append 'john'  
Binds.append 'jehn'  
  
recCount = db.SQLExecute(  
    "UPDATE person SET name = :1 WHERE name = :2", Binds )
```

---

## Encryption Methods

Class IVDataBase has encryption methods that allow you encrypt the whole database or only the database structure, so any other Valentina-enabled application will not be able to open your database. Usually you will use only one of this way of database encryption, although it is possible to combine both.

---

### ChangeStructureEncryption(inOldKey as String,inNewKey as String)

Parameter:	Description:
inOldKey	Old encryption key
inNewKey	New encryption key

Changes the encryption key of database structure file (.vdb). This function must be used:

- to convert not encrypted file to encrypted. ( inOldKey = "" )
- to change encryption key of encrypted.
- to cancel encryption. ( inNewKey = "" )

Usually this functions is called:

- just after creation of database.
- if user want to change the encryption key.

**Example:**

```
db.Create()  
db.ChangeStructureEncryption( "", "key12345" )  
....  
db.Close()
```

---

### SetStructureEncryption(inKey as String)

Parameter:	Description:
inKey	Encryption key

Informs an encrypted database about the structure encryption key to be used. It must be called just before IVDataBase.Open() function.

**Example:**

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

---

---

**ChangeEncryption(inOldKey as String,inNewKey as String)**

<b>Parameter:</b>	<b>Description:</b>
inOldKey	Old encryption key
inNewKey	New encryption key

Changes the encryption key of the whole database. This function must be used:

- to convert not encrypted database to encrypted. ( inOldKey = "" )
- to change encryption key.
- to cancel encryption. ( inNewKey = "" )

This function can work with not empty database. Usually this functions is called:

- just after IVDataBase.Create()
- if user want to change encryption key.

**Example:**

```
db.Create()
db.ChangeEncryption( "", "key12345" )
....
db.Close()
```

---

**SetEncryption(inKey as String)**

<b>Parameter:</b>	<b>Description:</b>
inKey	Encryption key

Informs an encrypted database about the encryption key to be used. It must be called just before IVDataBase.Open() function.

**Example:**

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

---

## Dump methods

---

### Dump(inDumpFile as String, inDumpType as DumpTypeEnum)

Parameter:	Description:
inDumpFile	The location of dump file.
inDumpType	Type of dump.

Dump all possible information about database into dump file. You can use this file to recreate database in new place.

inDumpType can be one of the following:

- 1 - SQL dump. Text file contains set of INSERT command.
- 2 - XML dump. Text file is information of database in XML form.

NOTE:

- XML dump is VERY useful because it allows safely DUMP a database with ObjectPtr fields. On LOAD of this dump into new database, Valentina will automatically correct values of ObjectPtr fields in related tables.
- You can use XML dump and load to compact your database.

**Example:**

```
dim db as IVDataBase
...
db.Dump(fiXML, 2) // do XML dump
```

---

### LoadDump( inDumpFile as String, inNewDb as String, inDumpType as DumpTypeEnum)

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

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

NOTE: you must use variable of type IVDataBase, but NOT your subclass of IVDataBase!  
After load, you need close IVDataBase and open it again as your subclass.

**Example:**

```
dim db as IVDataBase
...
db.LoadDump(fiXML,fiNewDb, 2) // do XML dump
```

## Class IVBaseObject

### properties

Name as String  
FieldCount as Integer // (r/o) number of fields in this BaseObject  
RecordCount as Long // (r/o) number of logical records in this BaseObject.  
DataBase as IVDataBase // Database of this BaseObject.  
IsEncrypted as Boolean

### methods

// Field Methods:

Field( inIndexOrName as VARIANT) as IVField

// Record Methods:

SetBlank() // Clears memory buffer of BaseObject,  
// set nullable fields to NULL  
AddRecord() as Long // Adds a new record with current value of fields  
UpdateRecord() // Updates existing record with new values  
DeleteRecord() as Boolean // Deletes current record  
DeleteAllRecords() // Makes table empty, very fast.  
Flush() // Saves on disk information of this BaseObject only.

// Navigation Methods:

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

GetRecID() as Long  
GoToRecID(RecID as Long) as Boolean

// Encryption methods:

ChangeEncryption(inOldKey as String,inNewKey as String)  
SetEncryption(inKey as String)

// Methods to change database structure:

DeleteField( inFld as IVField )  
ChangeFieldType( inField as IVField, inType as FieldTypeEnum,  
inParam as Long ) as IVField

continued on next page...

## Class IVBaseObject

---

// Methods to create a field in BaseObject.

```
CreateBooleanField ( inName as String, [inMethod as String] ) as IVField
CreateByteField    ( inName as String, [inMethod as String] ) as IVField
CreateShortField   ( inName as String, [inMethod as String] ) as IVField
CreateUShortField  ( inName as String, [inMethod as String] ) as IVField
CreateMediumField  ( inName as String, [inMethod as String] ) as IVField
CreateUMediumField ( inName as String, [inMethod as String] ) as IVField
CreateLongField    ( inName as String, [inMethod as String] ) as IVField
CreateULongField   ( inName as String, [inMethod as String] ) as IVField
CreateLLongField   ( inName as String, [inMethod as String] ) as IVField
CreateULLongField  ( inName as String, [inMethod as String] ) as IVField
CreateFloatField   ( inName as String, [inMethod as String] ) as IVField
CreateDoubleField  ( inName as String, [inMethod as String] ) as IVField
CreateDateField    ( inName as String, [inMethod as String] ) as IVField
CreateTimeField    ( inName as String, [inMethod as String] ) as IVField
CreateDateTimeField ( inName as String, [inMethod as String] ) as IVField
```

```
CreateFixedBinaryField( inName as String, inMaxLength as Long) as IVField
CreateVarBinaryField  ( inName as String, inMaxLength as Long) as IVField
```

```
CreateStringField( inName as String, inMaxLength as Long,
                  inLanguageName as String, [inMethod as String] ) as IVField
CreateVarCharField( inName as String, inMaxLength as Long
                  inLanguageName as String, [inMethod as String] ) as IVField
```

```
CreateBLOBField( inName as String, inSegmentSize as Long) as IVField
CreatePictureField( inName as String, inSegmentSize as Long) as IVField
```

```
CreateTextField( inName as String, inSegmentSize as Long,
                inLanguageName as String, [inMethod as String] ) as IVField
```

```
CreateObjectPtrField(
    inName as String,
    inTarget as IVBaseObject,
    [inDeletionControl as DeletionControlEnum = kV_SetNULL] ) as IVField
```

---

## Class description

Each IVBaseObject manages a table of the database.

Each IVBaseObject must have at least one field but not more than 65535 fields.

## Record Methods

---

### SetBlank()

Each IVBaseObject 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 the empty string. If any fields are nullable then they get the NULL value.

**Example:**

```
BaseObject.SetBlank()
```

---

### AddRecord() as Long

Adds a new record to the table with the current values in the memory buffer of this BaseObject. At first you need assign values to the fields for the new record and then call AddRecord(). Returns RecID of new record.

**Example:**

```
thePerson.SetBlank  
thePerson.FirstName.Value = "Jojn"  
thePerson.LastName.Value = "Roberts"  
thePerson.AddRecord()
```

---

### UpdateRecord()

This method stores new modified values of fields of CURRENT record.

**Example:**

```
thePerson.GotoRecID( SomeRecID )  
thePerson.FirstName.Value = "Brian"  
thePerson.LastName.Value = "Blood"  
thePerson.UpdateRecord()
```

### DeleteRecord() as Boolean

Deletes the current record of BaseObject.

If after deleted record exists the next record it becomes current. Otherwise the previous record becomes current. If Cursor becomes empty then current record is undefined.

Returns FALSE is record was not deleted, e.g. it was locked or not exists.

**Example:**

```
res = BaseObject.DeleteRecord()
```

---

### DeleteAllRecords()

Deletes all records in the BaseObject.

BaseObject becomes empty, the current record – undefined.

**Example:**

```
BaseObject.DeleteAllRecord()
```

---

### Flush()

This method flushes all unsaved information of this BaseObject from the cache to the disk. This method can be faster of IVDataBase.Flush() because affected data only one BaseObject.

**Example:**

```
BaseObject.Flush()
```

## Navigation Methods

The navigation methods of IVBaseObject class will seldom be used. The IVCursor class provides more powerful methods for selecting, sorting and navigation of the records.

---

### FirstRecord() as Boolean

Go to the first logical record of BaseObject. Reads record from disk to the memory buffer. of BaseObject.

Returns TRUE if the first record is found.

Returns FALSE if the current record already was first or BaseObject is empty.

**Example:**

```
res = BaseObject.FirstRecord()
```

---

### LastRecord() as Boolean

Go to the last logical record of BaseObject. Reads record from disk to the memory buffer. of BaseObject.

Returns TRUE if the last record is found.

Returns FALSE if the current record already was last or BaseObject is empty.

**Example:**

```
res = BaseObject.LastRecord()
```

---

### PrevRecord() as Boolean

Go to the previous logical record of BaseObject. Read record from disk to the memory buffer. of BaseObject.

Returns TRUE if the previous record is found.

Returns FALSE if the current record was the first or BaseObject is empty.

**Example:**

```
res = BaseObject.PrevRecord()
```

NextRecord() as Boolean

Go to the next logical record of BaseObject. Reads record from disk to the memory buffer of BaseObject.

Returns TRUE if the next record is found.

Returns FALSE if the current record was the last or BaseObject is empty.

**Example:**

```
res = BaseObject.NextRecord()
```

---

GetRecID() as Long

Returns RecID of the current record. Range is 1..N, 0 - if current record is undefined.

**Example:**

```
recID = BaseObject.GetRecID()
```

---

GoToRecID( inRecID as Long) as Boolean

Makes a record with the specified RecID current. Reads record from disk to the memory buffer of BaseObject. This is the fastest way to access a record.

Returns TRUE if record found. Otherwise returns FALSE, i.e. record is deleted.

**Example:**

```
found = BaseObject.GoToRecID( recID )
```

---

## Field Methods

---

[Field\( inIndexOrName as VARIANT\) as IVField](#)

<b>Parameter:</b>	<b>Description:</b>
inIndexOrName	The index or name of the field. Index start from 1.

This methods allows you to access fields of BaseObject by index or name. The index starts from 1. If the field with specified index or name doesn't exist then it returns NULL.

**Example:**

```
fld = BaseObject.Field( i )
```

**Example:**

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

---

## Database structure methods

You may need these methods if:

- 1) you are developing an application with a dynamic database structure;
- 2) you want to change the database structure of your existing application (for example in a new version).

The main purpose for these methods – is to change the size of the Table record. If a Table has records then disk files must be transformed. Valentina will perform these operations with the help of temporary files: so if computer crashes for any reason, the database will not be corrupted.

### Creation of fields.

Valentina has big enough set of functions to create fields, but you can easily see that they have simple names and are easy to use.

The first big group of methods has 2 parameters and are used to create numeric and date/time fields. Parameter `inMethod` is optional and must be `NULL` in case you want a normal field. If you need BaseObject method (calculation field) you need specify the second parameter.

```

CreateBooleanField ( inName as String, [inMethod as String] ) as IVField
CreateByteField   ( inName as String, [inMethod as String] ) as IVField
CreateShortField  ( inName as String, [inMethod as String] ) as IVField
CreateUShortField ( inName as String, [inMethod as String] ) as IVField
CreateMediumField ( inName as String, [inMethod as String] ) as IVField
CreateUMediumField ( inName as String, [inMethod as String] ) as IVField
CreateLongField   ( inName as String, [inMethod as String] ) as IVField
CreateULongField  ( inName as String, [inMethod as String] ) as IVField
CreateLLongField  ( inName as String, [inMethod as String] ) as IVField
CreateULLongField ( inName as String, [inMethod as String] ) as IVField
CreateFloatField  ( inName as String, [inMethod as String] ) as IVField
CreateDoubleField ( inName as String, [inMethod as String] ) as IVField
CreateDateField   ( inName as String, [inMethod as String] ) as IVField
CreateTimeField   ( inName as String, [inMethod as String] ) as IVField
CreateDateTimeField ( inName as String, [inMethod as String] ) as IVField

```

### Example:

```

fldBornDate = boPerson.CreateDateField( "BornDate" )

fldAge = boPerson.CreateUShortField(
    "Age", "Year(currentDate) - Year(BornDate)")

```

```
CreateFixedBinaryField( inName as String, inMaxLength as Long ) as IVField  
CreateVarBinaryField ( inName as String, inMaxLength as Long ) as IVField
```

These 2 methods are used to create FixedBinary or VarBinary fields. These fields do not have method, but have new parameter -- maximal length [1..65535].

**Example:**

```
fldInfo = boVector.CreateFixedBinaryField( "vector", 100 )
```

---

```
CreateStringField( inName as String, inMaxLength as Long,  
                  inLanguageName as String, [inMethod as String] ) as IVField  
CreateVarCharField( inName as String, inMaxLength as Long  
                   inLanguageName as String, [inMethod as String] ) as IVField
```

These 2 methods are used to create String of fixed length or VarBinary fields. These methods require that you specify name and maximal length. Optionally you may specify BaseObject method formula.

**Example:**

```
fldFirstName = boPerson.CreateVarCharField( "FirstName", 504 )  
fldLastName = boPerson.CreateVarCharField( "LastName", 504 )  
fldFullName = boPerson.CreateVarCharField( "LastName", 504,  
                                           "CONCAT(FirstName, ' ', LastName)" )
```

---

```
CreateBLOBField( inName as String, inSegmentSize as Long) as IVField  
CreatePictureField( inName as String, inSegmentSize as Long) as IVField
```

```
CreateTextField( inName as String, inSegmentSize as Long,  
                inLanguageName as String, [inMethod as String] ) as IVField
```

These methods create BLOB, TEXT and Picture fields.  
A BLOB field require a special parameter -- SegmentSize (in bytes).

**Example:**

```
fldText = boPerson.CreateTextField( "Notes", 512, "English" )
```

---

```
CreateObjectPtrField(  
    inName as String,  
    inTarget as IVBaseObject,  
    [inDeletionControl as DeletionControlEnum = kV_SetNULL] ) as IVField
```

Creates an ObjectPtr field. You must provide target BaseObject, i.e. it must exists already. In case you need structure with circle dependencies you may provide NULL as Target and after creation of all BaseObjects set it to correct target.

Optional parameter 'Deletion Control' specify how records will be deleted. It can have values: kV\_SetNULL, kV\_Cascade, kV\_Restrict. Look in ValentinaKernel.pdf for more details.

**Example:**

```
fldPtr = boPerson.CreateObjectPtrField( "Father", boPerson, kV_SetNULL )
```

---

---

**DeleteField( inFld as IVField )**

<b>Parameter:</b>	<b>Description:</b>
inFld	The field that should be deleted.

Removes the referenced field (column) from the BaseObject. This operation is undoable! It can be made in about 0 seconds for a BaseObject with any number of records.

**Example:**

```
BaseObject.DeleteField( fld )
```

---

**ChangeFieldType(**  
inFld as IVField,  
inNewType as FieldTypeEnum,  
inParam as Integer ) as IVField

<b>Parameter:</b>	<b>Description:</b>
inFld	The field, which type should be changed.
inNewType	New type for a field.
inParam	Additional parameter.

Sometimes you may need to change the type of the 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 change its type to VULong.

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

**Example:**

```
fld = BaseObject.ChangeType( fld, kV_TypeString,40 )
```

---

## Encryption methods

Class IVBaseObject has 2 functions for encryption. You may wish to use this functions if you want to encrypt only one or several BaseObjects of database. This give you faster solution than encryption of the whole database.

You can use this function also to specify for a BaseObject a special key, different from the key of its BaseObject. Unlikely you will do this, but it is possible.

---

### ChangeEncryption(inOldKey as String,inNewKey as String)

Parameter:	Description:
inOldKey	Old encryption key
inNewKey	New encryption key

Changes the encryption key of BaseObject. This function must be used:

- to convert not encrypted BaseObject to encrypted. ( inOldKey = "" )
- to change encryption key of encrypted BaseObject.
- to cancel encryption. ( inNewKey = "" )

This function can work with not empty BaseObject. Usually this functions is called:

- just after IVDataBase.Create()
- if user want to change the encryption key.

#### Example:

```
db.Create()
bo.ChangeEncryption( "", "key12345" )
db.Close()
```

---

### SetEncryption(inKey as String)

Parameter:	Description:
inKey	Encryption key

Informs an encrypted database about encryption key to be used for this BaseObject. It must be called just after VDatabase.Open() function.

#### Example:

```
db.Open()
bo.SetEncryption( "key12345" )
```

#### Example:

```
db.SetEncryption( "key1" )
db.Open()
bo.SetEncryption( "key12345" ) // for this BaseObject we specify special key
```

## Class IVField

### Properties

```
Name          as String          // Up to 504 bytes.
Type          as FieldTypeEnum [r/o] // The type of field.

// Flags:
Indexed       as Boolean        // TRUE if the field is indexed.
Unique        as Boolean        // TRUE the field has unique values only.
Nullable      as Boolean        // TRUE if the field accepts NULL values
Compressed    as Boolean        // TRUE if the BLOB field uses compression.
IndexByWords  as Boolean        // TRUE if the string field is indexed by words.

// Method properties:
Method        as String         // The method of field.
IsMethod      as Boolean [r/o]  // TRUE if the field is method.

// String properties:
MaxLength     as Long           // The max length of field in bytes. [1-65535]
Language      as String         // The language name of field.

// BLOB properties:
SegmentSize   as Long           // The segment size for a BLOB field in bytes..

// ObjectPtr properties:
Target        as IVBaseObject   // The taeget BaseObject of this ObjectPtr field..
DeletionControl as DeletionControlEnum // The deletion control.

// Value properties:
IsNull        as Boolean        // TRUE if the current value of field is NULL.
Value         as VARIANT        // Value of field.
StrValue      as String         // Value of field in string form.

// Encryption properties:
IsEncrypted   as Boolean
```

continued on next page...

## Class IVField

---

### Methods

// BLOB methods:

GetBLOBDataSize() as Long

ReadBLOBData( inHowMuch as Long, inOffset as Long) as VARIANT

WriteBLOBData( inValue as VARIANT, inOffset as Long)

DeleteBLOBData()

// ObjectPtr

ConvertFromRDB( inPrimaryKey as IVField, inForeignKey as IVField )

// Encryption methods:

ChangeEncryption(inOldKey as String,inNewKey as String)

SetEncryption(inKey as String)

---

## Properties Description

---

### Name as String

---

Each field must have name that is unique in the scope of BaseObject. Maximal length of name is 504 bytes.

**Example:**

```
name = fld.Name  
fld.Name = "First Name"
```

### Type as FieldTypeEnum

---

Each field has a type that defines the context of the data which can be stored in it. The type of field is assigned when you create field in BaseObject.

**Example:**

```
case fld.Type
```

**See also:** IVBaseObject.ChangeType

### Nullable as Boolean

---

If TRUE then this field can have the NULL value. Because this feature adds 1 bit per record for the field, the default is FALSE.

**Example:**

```
fld.Nullable = TRUE  
kIsNullable = fld.Nullable
```

### Unique as Boolean

---

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

**Example:**

```
fld.Unique = TRUE  
kIsUnique = fld.Unique
```

### Indexed as Boolean

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

**Example:**

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

---

### IndexByWords as Boolean

using this flag you can specify that the String or VarChar field should be indexed by each word.

**Example:**

```
fldString.IndexByWords = TRUE
```

---

### Compressed as Boolean

This flag specify is BLOB field is compressed. You can set it true to make BLOB field compress its data storing them on disk.

**Example:**

```
kIsCompressed = fldBlob.Compressed
```

---

**BaseObject Method properties**

---

**IsMethod as Boolean**

True if the field is virtual, i.e. it is BaseObject Method.  
Read Only.

**Example:**

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

---

**Method as String**

This property contain text of BaseObject method. You can use it to read this text. Also you can use it to change method's formula.

NOTE: you can not use it to convert normal field into BaseObject method. You must create BaseObject method from start using BaseObject.CreateXXXField() functions.

**Example:**

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

**Example:**

```
fld.Method = "CONCAT( FirstName, ' ', LastName )"
```

---

**String properties**

---

[MaxLength as Long](#)

Maximal length of field can be in range 1 .. 65535 bytes.

Can be applied to VString, VVarChar, VFixedBinary, VVarBinary fields.

IF you change maximal length of field, you change size of table records. This means that Valentina must rebuild table, so this operation require some time.

**Example:**

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

---

[Language as String](#)

You can specify the language that will be used for indexing of this field.

IF you change language then Valentina must rebuilt index.

**Example:**

```
fldString.Language = "German"
```

---

**BLOB properties**

---

[SegmentSize as Long](#)

The parameter `SegmentSize` is used by Valentina only once - when it creates the BLOB-file. This parameter can't be changed at runtime. By default it is 1 KB.

**Example:**

```
segment = fldBlob.SegmentSize
```

**ObjectPtr properties**

`ObjectPtr` field is used to establish relation 1 : 1 or 1 : M between 2 tables, see details in [ValentinaKernel.pdf](#).

---

[Target as IVBaseObject](#)

The target `BaseObject` for this `ObjectPtr` field.

The `Target` must be defined when you create the field. Usually there is no reason to change `Target` runtime.

**Example:**

```
dim pointed as VBaseObject  
pointed = fldPtr.target
```

---

[DeletionControl as DeletionControlEnum](#)

The `DeletionControl` regulates record deletion in the "Many" table when a record is deleted in the "One" table. It can be changed at runtime. This is a rule, which defines the behavior on deletion of record.

**Example:**

```
fldPtr.DeletrionCotrol = kV_SetNULL
```

**Value properties**

---

**IsNull**

This is a record property. It is TRUE if the value of this field for the current record of the table is NULL. Don't confuse it with the property Nullable! Nullable is a property of the table column, IsNull is a property of the current record.

**Example:**

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

---

**StrValue as String**

These methods allow you to get or set a field value using strings regardless of the field type.

Valentina will convert the string to/from an appropriate type. In fact it is possible to convert to/from string practically ANY type except BLOB, Picture and FixedBinary/VarBinary.

**Example:**

```
fldLong.StrValue = "100000"  
fldDate.StrValue = "03/06/2002"  
fldString.StrValue = "Jumping fox"
```

---

---

**Value as VARIANT**

You can use this property to SET / GET value of field. This property has type VARIANT, so it can be used to transfer value of practically any type.

You CAN use this property:

**1) to set /get value of a numeric field:****Example:**

```
fldLong.Value = 555  
fldFloat.Value = 44.12345
```

**2) to set / get value of a String/VarChar field:****Example:**

```
fldString.Value = "jumping fox"
```

**3) to set/get value of a Date, Time or DateTime field.**

Value is returned as Date type of visualBasic (VT\_DATE type of VARIANT for C++).

**4) to set / get value of a BLOB field.**

Notice, that this way of access a BLOB field read/write BLOB value into single step as single block. IF you have huge BLOB that do not fit into RAM you should use methods of BLOB field to read/write BLOB value by parts.

**Example:**

```
dim arr(1 to 1000) as Byte  
// fill array here  
boT1.Field("fldBLOB").Value = arr()  
boT1.AddRecord()
```

**5) to set/get value of a FixedBinary / VarBinary field.****Example:**

```
dim arr(1 to 100) as Byte  
// fill array here  
boT1.Field("fldFixedBinary").Value = arr()  
boT1.AddRecord()
```

**6) to set/get value of a Picture field.**

In VisualBasic you just assign an image to Value of field:

**Example:**

```
fldPicture.Value = image
```

If look deeply into C++ code, then VARIANT has type VT\_DISPATCH and contains IPicture interface. So if you use VCOM in C++ application you must put IPicture into VARIANT and assign such value to the field. Valentina also returns you VARIANT that contains IPicture. Note that Picture you can read / write in the single step only.

**[ NOTE FOR C++ ]**

It is obvious that for C++ application that already has DIB HANDLE it is very annoying to pack DIB into IPicture interface. Besides this slow down the work because Valentina also must spend time on extracting of DIB from IPicture.

To simplify this VCOM provides you the following TIP. You can just put DIB HANDLE into VARIANT and set type of VARIANT to be VT\_I4.

**Example:**

```
void foo( void )
{
    VARIANT v;
    v.lVal = (long) myDibHandle;    // force handle to be long.
    v.vt = VT_I4;
    Field.put_Value( v );
}
```

To get DIB from Valentina in the same way you must inform it that you want get DIB as HANDLE but not as IPicture. To do this, you need set type of VARIANT as VT\_I4 before send it to get\_Value().

**Example:**

```
void foo( void )
{
    VARIANT v;
    v.vt = VT_I4;    // ask to return DIB directly.
    Field.get_Value( &v );

    // now we can have normal DIB HANDLE:
    HANDLE pict = (HANDLE) v.lVal;
}
```

---

## Encryption methods

Class IVField has 2 functions for encryption. You may wish to use this functions if you want encrypt only one or several fields of database. This give you faster solution than encryption of the whole database.

You can use this function also to specify for a field a special key, different from the key of its BaseObject. Unlikely you will do this, but it is possible.

---

### ChangeEncryption(inOldKey as String,inNewKey as String)

<b>Parameter:</b>	<b>Description:</b>
inOldKey	Old encryption key
inNewKey	New encryption key

Changes the encryption key of Field. This function must be used:

- to convert not encrypted field to encrypted. ( inOldKey = "" )
- to change encryption key of encrypted field.
- to cancel encryption. ( inNewKey = "" )

This function can work with not empty BaseObject. Usually this functions is called:

- just after IVDataBase.Create()
- if user want to change the encryption key.

#### Example:

```
db.Create()
fld.ChangeEncryption( "", "key12345" )
db.Close()
```

---

### SetEncryption(inKey as String)

<b>Parameter:</b>	<b>Description:</b>
inKey	Encryption key

Informs an encrypted database about encryption key to be used for this Field. It must be called just after IVDataBase.Open() function.

#### Example:

```
db.Open
fld.SetEncryption( "key12345" )
```

#### Example:

```
db.SetEncryption( "key1" )
db.Open
fld.SetEncryption( "key12345" ) // for this field we specify other key
```

---

## BLOB Methods

---

### [GetBLOBDataSize\(\) as Long](#)

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

**Example:**

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

---

### [DeleteBLOBData\(\)](#)

Deletes the BLOB data of the field. After this function you must Update record of BaseObject to store in the table new reference of BLOB record (NULL). This method is useful for you if you want delete BLOB data, but don't want delete records of BaseObject.

**Example:**

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

---

### [ReadBLOBData\( inHowMuch as Long, \[inOffset as Long = 0\] \) as VARIANT](#)

<b>Parameter:</b>	<b>Description:</b>
inHowMuch	The number of bytes to read.
inOffset	The offset to start reading BLOB data.

Read from BLOB field data inHowMuch bytes starting from inOffset. Returns data packed into ByteArray of VARIANT.

**Example:**

```
dim blobValue() as byte
...
blobValue = fldBLOB.readBLOBData( size )
```

---

---

`WriteBLOBData(inValue as VARAINT, [inOffset as Long = 0] )`

**Parameter:**

inValue  
inOffset

**Description:**

Array of bytes to be stored in BLOB field.  
The offset in BLOB field to start write this part of info.

These methods allow you to store in the BLOB field any data using String of REALbasic as exchange structure between REALbasic and Valentina.

Second form of methods allow you random access to the context of BLOB field.

**Example:**

```
dim s1(1 to 600) as byte
```

```
fldBLOB.WriteBLOBData( s1 )
```

```
fldBLOB.WriteBLOBData( s1, 600 ) // append it again
```

## Class IVCursor

### properties

DataBase as IVDataBase [r/o] // Database of this Cursor.  
SQLstring as String [r/o] // SQL string of this Cursor.

FieldCount as Integer [r/o] // The number of fields in Cursor.  
RecordCount as Long // The number of selected records, it can be reduced.

CurrentPosition as Long // The current position in the cursor.

ReadOnly as Boolean [r/o] // TRUE if records can't be changed  
// i.e. you can't add/update/delete records.

### methods

// Field Methods:

Field( IndexOrName as VARIANT ) 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 Boolean // adds a new record to the cursor

UpdateRecord() as Boolean // updates current record of cursor

UpdateAllRecords() as Boolean // updates ALL records of cursor by new value.

DeleteRecords() as Boolean // deletes current record of cursor

DeleteAllRecords() as Boolean // deletes all records of cursor

DropRecord() // removes the current record from cursor  
// but don't delete it from original BaseObject.

ImportText( inFile as String,  
[inFieldDelimiter as String = '\t'], [inLineDelimiter as String = '\r'] )

ExportText( inFile as String,  
[inFieldDelimiter as String = '\t'], [inLineDelimiter as String = '\r'] )

---

## Class description

This class provides the result of an execution of SQL's SELECT statement. Valentina offers a cursor with random access to records.

Each cursor has independent memory buffer, so you can have many cursors at the same time for the same BaseObject, which point on different records.

You create a cursor object using IVDataBase.SqlSelect().

## Properties description

---

### FieldCount

Here you can find how many columns this cursor has.

**Example:**

```
fldCount = curs.FieldCount // get local shortcut to avoid of calling in loop
for i = 1 to fldcount
    ...
next
```

---

### RecordCount

Equal to the number of records found as a result of SQL query.

**Example:**

```
recCount = curs.RecordCount // get local shortcut to avoid of calling in loop
for i = 1 to fldcount
    ...
next
```

---

---

### CurrentPosition

To navigate through the records of the cursor you can use the property CurrentPosition. Position in the Cursor is not the same as CurrentRecord in BaseObject. For example, the first record of the Cursor can be 125th in the BaseObject.

**Example:**

```
CurrentPosition = 1           // go to the first record of cursor.
CurrentPosition = RecordCount // go to the last record of cursor.

// go to the NextRecord:
if( CurrentPosition < RecordCount )
    CurrentPosition = CurrentPosition + 1
end if
```

**NOTE:** when you assign a new value to the CurrentPosition you force Valentina to load a record from disk to the memory buffer. So this is not “just variable”. On the other hand, if the record was read before and it is in the cache, then repeated reading of its data is a “no time” operation.

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

---

### ReadOnly

True if the records of Cursor can be readed, but cannot be modified; otherwise it is FALSE.

**Example:**

```
if( curs.ReadOnly )
    ....
```

---

## Field Methods

---

[Field\( IndexOrName as VARIANT\) as IVField](#)

<b>Parameter:</b>	<b>Description:</b>
IndexOrName	The index or name of the field. The index start from 1.

You can use these methods to access fields of the cursor and their values. The order of fields in the cursor is the same as the order of fields in the SELECT statement of query.

When you have a field you can call for it any function of class IVField.

### Example:

```
LastName as String
dim curs as IVCursor

curs = gDataBase.SQLSelect("select * from person where name like 'john' no_case")

curs.FirstRecord
Do
    LastName = curs.Field( "last_name" ).StrValue
while curs.NextRecord
```

---

## Navigation Methods

---

### FirstRecord() as Boolean

---

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

**Example:**

```
res = curs.FirrstRecord()
```

### LastRecord() as Boolean

---

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

**Example:**

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

### PrevRecord() as Boolean

---

Go to the previous logical record of 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.

**Example:**

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

### NextRecord() as Boolean

---

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

**Example:**

```
if( myCursor.FirstRecord() )  
  Do  
    // some work here  
  WHILE myCursor.NextRecord()  
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 get the NULL value.

**Example:**

```
curs.SetBlank()
    curs.Field(1).Value = i
    curs.Field(2).Value = i
res = curs.Add()
```

---

### [AddRecord\(\) as Boolean](#)

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

Returns FALSE if record was not added, e.g. cursor is ReadOnly.

**Example:**

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

UpdateRecord() as Boolean

Updates current record of Cursor by new values.

Retuns FALSE if record was not updated, e.g. cursor is ReadOnly.

**Example:**

```
curs.currentRecord = i
    curs.Field(1).Value = i + 100
    curs.Field(2).Value = i + 100
res = curs.UpdateRecord()
```

---

UpdateAllRecords() as Boolean

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

This function is much much faster then iteration of cursor records in loop to assign new values.

Retuns FALSE if records was not updated, e.g. cursor is ReadOnly.

**Example:**

```
curs.Field(1).Value = 145
curs.Field(2).Value = 200

res = curs.UpdateAllRecords()
```

DeleteRecord() as Boolean

Deletes the current record of cursor.

If after deleted record exists the next record it becomes current. Otherwise the previous record becomes current. If Cursor becomes empty then current record is undefined.

Returns FALSE is record was not deleted, e.g. it was locked or not exists, or cursor is read only.

**Example:**

```
res = curs.DeleteRecord()
```

---

DeleteAllRecords() as Boolean

Deletes all records of the Cursor.

Cursor becomes empty, the current record – undefined.

Returns FALSE if record was not added (e.g. cursor is ReadOnly).

**Example:**

```
res = curs.DeleteAllRecords()
```

---

DropRecord()

Removes the current record from Cursor, but do not delete it from original BaseObject.

**Example:**

```
curs.DropRecord()
```

---

## Import/Export

---

```
ImportText( inFile as String,
           [inFieldDelimiter as String = '\t'],
           [inLineDelimiter as String = '\r'] )
```

<b>Parameter:</b>	<b>Description:</b>
inFile	The full path to file to be imported.
inFieldDelimiter	The character to be used as field delimiter, default is 0x09.
inLineDelimiter	The character to be used as field delimiter, default is 0x13.

Imports the specified text file into the fields of the Cursor. The Cursor must be designed to have the flag CanBeUpdated TRUE.

The parameters FieldDelimiter and LineDelimiter are optional, i.e. you may specify either or both of them. On default they are TAB (09) and CR(13) respectively. If the cursor represents a subset of the table-fields, then the omitted fields will be filled with blank values.

In the current version supported importing to the Cursor designed for a single Table only.

### Example:

```
curs.ImportText( fileToImport, "\t", "\r" )
```

---

```
ExportText( inFile as String,
           [inFieldDelimiter as String = '\t'],
           [inLineDelimiter as String = '\r'] )
```

<b>Parameter:</b>	<b>Description:</b>
inFile	The full path to file to be imported.
inFieldDelimiter	The character to be used as field delimiter, default is 0x09.
inLineDelimiter	The character to be used as field delimiter, default is 0x13.

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.

The current version of Valentina will export data only into a cursor based on a single table.

### Example:

```
curs.ExportText( fileToExport, "\t", "\r" )
```

## Class VServer

Only for V4RB Client.

### properties

Version	as String	// (r/o) version of server
HostName	as String	// (r/o) the name of host where server is located.
UserName	as String	// (r/o) the name of current user
DatabaseCount	as Integer	// (r/o) the number of databases that know server.
ConnectionCount	as Integer	// (r/o) the number of active connections to server.

### methods

VServer( inHost as string,  
inUserName as String,  
inUserPassword as String,  
[inPort as Integer] )

OpenSession()  
CloseSession()

// INI-File Methods:

GetIniProperty( Name as String ) as String  
SetIniProperty( Name as String, Value as String )

// Server databases methods:

RegisterDatabase(inDbName as string, inServerPath as String)  
UnregisterDatabase(inDbName as string)

// User Methods:

AddUser(inUserName as String, inPassword as String)  
RemoveUser(inUserName as String)  
ChangeUserPassword( inUserName as String,  
inNewPassword as String)

// DatabaseInfo Methods:

DatabaseInfo(index as integer) As VDatabaseInfo  
Refresh

### Class description

This class is needed only if you develop Server Part of your own Server Application. This class allows you develop your own frontend for VServer. It allows for a user which have administraton rights manage parameters of Server, locally or remotely.

### Properties description

---

#### Version as String

Returns the string that contains version of VServer.

**Example:**

```
version = server.version
```

---

#### hostName as String

Returns the string that contains name of server host.

**Example:**

```
version = server.version
```

---

#### UserName as String

Returns the string that contains the name of the current user, i.e. administrator self.

**Example:**

```
version = server.version
```

## Class IVServer

---

---

### DatabaseCount as Integer

Returns the count of databases that server knows about. In other words, this is the number of databases registered in the Master Database of VServer.

**Example:**

```
dbCount = server.DatabaseCount
```

---

### ConnectionCount as Integer

Returns the count of all active connections to server.

**Example:**

```
connCount = server.ConnectionCount
```

## Creation of VServer

---

```
VServer( inHost as string,  
        inUserName as String,  
        inUserPassword as String,  
        [inPort as Integer] )
```

<b>Parameter:</b>	<b>Description:</b>
inHost	The IP-address or DNS name of host.
inUserName	The user name.
inUserPassword	The password of user.
inPort	The port number that listen Server on inHost. On default - standard port of Valentina Server.

This method construct VServer object. Note that only Administrator User(s) can use this object. Constructor simply stores parameters and do not try connect yet. The real connection will happens on OpenSession().

### Example:

```
dim server as VServer  
server = new VServer ("localhost", "sa", "sa" )
```

---

### OpenSession()

Establish conenction to server.

Possible errors: Wrong user name, Wrong password,  
User is not adminstrtror, Connection cannot be established.

### Example:

```
dim server as VServer  
server = new VServer ("localhost", "sa", "sa" )  
server.OpenSession
```

## Class IVServer

---

---

### CloseSession()

Close conenction to server.

#### Example:

```
dim server as VServer
server = new VServer ("localhost", "sa", "sa" )
server.OpenSession
...
server.CloseSession()
```

---

### GetVariable( inName as String ) as String

This method allows you to read value of the specified Server Variable. The name of variable is case insensetive. As names of variables you can use constants of INI-file of VServer. See docs of VServer.

#### Example:

```
cache = server.GetVariable( "CacheSize" )
```

---

### SetVariable( inName as String, inNewValue as String )

This method allows you to change value of the specified Server Variable. The name of variable is case insensetive. As names of variables you can use constants of INI-file of VServer. See VServer.pdf for details.

NOTE: Some variables require the restart of VServer to affect changes.

#### Example:

```
server.SetVariable( "CacheSize", 8 )
```

### [RegisterDatabase\(inDbName as string, inServerPath as String\)](#)

You need this function if you have some Valentina database and you want put it under Valentina Server. You need then register this database in the Master Database. For registration you should specify the name of database and its full path on server. This command adds a new record to Master Database.

Since Method accept full path, you can keep database at any location on Server HDD.

**Errors:**

Database Name already exists.

**Example:**

```
res = server.RegisterDatabase( "Accounting",  
                              "C:\SomeCompany\account2002.vdb" )
```

---

### [UnregisterDatabase\(inDbName as string\)](#)

If you want remove some database from the scope of VServer, you need remove record about it from Master Database. You can do this using this method.

**Errors:**

Database Name not found.

**Example:**

```
server.UnregisterDatabase( "Accounting" )
```

## Class IVServer

---

---

`AddUser( inUserName as String, inPassword as String )`

Administrator can add new users to the Master Database.

**Errors:**

User Name already exists.

**Example:**

```
res = server.AddUser( "Peter", "a1234fteg4" )
```

---

`RemoveUser( inUserName as String )`

Administrator can remove users from the Master Database.

**Errors:**

User Name not found.

**Example:**

```
server.RemoveUser( "Peter" )
```

---

`ChangeUserPassword( inUserName as String,  
inNewPassword as String)`

Administrator can change password of a user.

**Errors:**

User Name not found.

**Example:**

```
res = server.ChangeUserPassword( "Peter", "rvsa3341" )
```

## Class IVServer

---

### [DatabaseInfo\(index as integer\) As VDatabaseInfo](#)

This method allows you to iterate the collection of DatabaseInfo objects.

Object of Vserver get list of DatabaseInfo on OpenSession. You can periodically refresh this list using Refresh() method.

#### **Example:**

```
dim dbi as VDatabaseInfo

for i = 1 to server.DatabaseCount
    dbi = server.DatabaseInfo
    ....
next
```

---

### [Refresh\(\)](#)

This method allows you refresh list of VDatabaseInfo objects at any time you want. This method send a request to the VServer.

## Class VDatabaseInfo

Only for V4RB Client.

### properties

Name	as String	// (r/o) The name of database.
Path	as String	// (r/o) The full path of database on server.
ClientCount	as Integer	// (r/o) The number of connected clients.

### methods:

ClientInfo( Name as String ) as VClientInfo  
Refresh()

---

ClientInfo( Name as String ) as VClientInfo

This method allows you to iterate the collection of ClientInfo objects.

Object of Vserver get list of DatabaseInfo on its creation. You can periodically refresh this list using Refresh() method.

### Example:

```
dim ci as VclientInfo

for i = 1 to dbi.ClientCount
    ci = dbi.DatabaseInfo
    ...
next
```

---

Refresh()

This method allows you refresh list of ClientInfo objects at any time you want. This method send a request to the VServer.

## Class VClientInfo

Only for V4RB Client.

**properties**

Address	as String	// (r/o) The IP address of client computer.
Port	as String	// (r/o) The port number of client computer..
ConnectionID	as Integer	// (r/o) The ID of this connection.