VALENTINA
for REALbasic Reference

Paradigma (www.paradigmasoft.com)

Acknowledgments:

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

Contents

Contents

Global Methods

ValentinaDebugON([level])
ValentinaDebugOFF

Sets the level of debugging for Valentina.

Parameter ‘level’ can be one of the following:

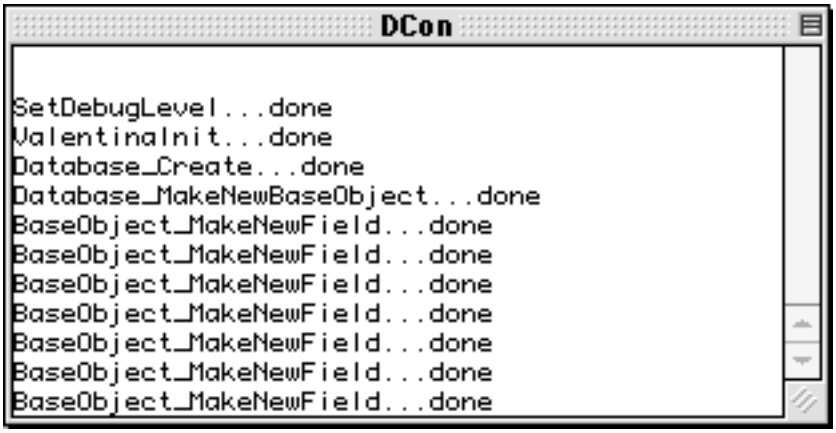
0 - no debug messages.
1 - message only if error was occurred.
2 - each function will produce message to the console window.

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 V4RB in the console window. Advnatage of this method is that even after crash of REALbasic and your app 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:

```
#if DebugBuild
    ValentinaDebugON( 2 )
#endif
```

• DO NOT FORGET to set the debuging level to zero for final release!



```
ValentinaInit(    CacheSize as Integer,
                  MacSerialNumber as String,
                  WinSerialNumber ) as Integer
```

To improve disk access, Valentina uses a cache mechanism. In the ValentinaInit method, you must define the size of the cache. It can be 100 kB if the database is tiny, or it can be several MB if the db is big. 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 for the MacOS version of or both MacOS and Windows versions in this method. Otherwise pass empty strings, 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.

ValentinaShutDown

Destroy cache and other allocations of Valentina kernel.

The Class Hierarchy

Valentina for REALbasic is implemented as set of classes which allow you to control a database:

class VDataBase
class VBaseObject
class **VField**
    class VBoolean
    class VByte
    class VShort
    class VUShort
    class VLong
    class VULong
    class VFloat
    class VDouble
    class VDate
    class VTime
    class VDateTime
    class VString
    class VarChar
    class VBLOB
        class VText
        class VPicture
class VCursor

Note, indention to the right shows inheritance of a class.

The class **VField** is shown as bold; this means it is an abstract class and captures the essence of it's subclasses. You can not create it by operator NEW. Only the subclasses can be created and used explicitly.

Class VDataBase

properties

```
Name                as String (r/o)
BaseObjectsCount    as Integer(r/o)

SchemaVersion       as intger           // Version of db Schema
Creator             as String           // Mac creator sign
ErrNumber           as Integer (r/o)    // Number of last error, 0 if OK.

DateFormat          as Integer          // specifies the format of date:
                                        // 0 - M/D/Y, 1 - D/M/Y, 2 -  Y/M/D
DateSep             as String           // separator for date, e.g. '/'
TimeSep             as String           // separator for time, e.g. ':'
```

methods

```
// Disk files methods
Create(Location as FolderItem, Mode as Integer, SegmentSize as Integer) as Boolean
Open(Location as FolderItem) as Boolean
Close
Flush

// BaseObject methods
BaseObject( Index as Integer ) as VBaseObject
BaseObject( Name as String ) as VBaseObject

// Database structure methods
CreateBaseObject( inName as String ) as VBaseObject
DeleteBaseObject( inBaseObject as VBaseObject )

// SQL Methods
SQLSelect( Query as String ) as VCursor
```

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

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().

Creator
With Macintosh applications, you can specify the creator's signature for database files. This allows you design an icon suite for your application. This assignment can be made using REALbasic and is usually done during the file SAVE operation.

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.

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

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

Methods description

Create(Location as FolderItem, Mode as Integer, SegmentSize as Integer) as Boolean
Valentina has its own internal file system, so it can store many logical files in a single disk file. Valentina can store databases in one, two, three or four files. This is determined when you specify the parameter “Mode” in the Create method. The choices are:

1 (description, data, BLOB, indexes)
2 description + (data, BLOB, indexes)
3 description + (data, BLOB) + indexes
4 description + data + BLOB + indexes

While each disk file can be up to 2GB (on MacOS up to 9.0) then total size of database will 2 GB for mode 1 and up to 8 GB for mode 4. Storing a tmp information in the separate disk file reduce chance of db corruption. Sometimes you may wish keep all db in the one disk file (this sometimes love MAC users). If you have separate description file then you can open it forsing Valentina to build a new empty database.

The parameter “SegmentSize” in the Create method defines the size of each segment of disk file assigned to the database. When Valentina needs additional disk space it allocates it in segments. A typical setting for this parameter is to 32 * 1024 bytes. However, if you want a tiny database in the your application, as in an address book, then you might set it to 2 or 4 KB. We don’t recommend using a segment size more than 64KB.

Note, the size of segment of 32 KB doesn’t limit the database to be 32KB. This is just the size of segment; for example if your database is 1 MB then it has 34 segments inside of the file. A segment of Valentina is like a disk cluster in the MacOS or Windows operating system.

Open(Location as FolderItem) as Boolean
Opens existing database at specified location.

Close
Closes the database.

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

Database structure methods

CreateBaseObject(inName as String) as VBaseObject
Creates a new empty Table in the database.
You need to add columns to this table using VBaseObject.CreateField() method.

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

BaseObject methods

BaseObject(Index as Integer) as VBaseObject
Returns a BaseObject by index.

BaseObject(Name as String) as VBaseObject
Returns BaseObject by name, case insensitive.

SQL Methods

SQLSelect(Query as String) as VCursor

Valentina for REALbasic uses SQL to search a database. See document “Valentina SQL” on information about SQL supported by Valentina.

The SQLSelect method is given a string as a paramemter (this is a SQLstring specifying what is to be searched for), resolves it, and returns the resulting table as a cursor of type VCursor. 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.

Class VBaseObject

properties

```
Name          as String
FieldCount    as Integer      // (r/o) number of fields in this BaseObject
RecordCount   as Integer      // (r/o) number of logical records in this BaseObject.
DataBase      as VDataBase    // Databaseof this BaseObject.
```

methods

```
// Field Methods:
Field( inIndex as Integer ) as VField
Field( inName as String ) as VField

// Record Methods:
SetBlank                    // Clears memory buffer of  BaseObject,
                            // set nullable fields to NULL

AddRecord                   // 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 Integer
GoToRecID(RecID as Integer) as Boolean

// Methods to change database structure:
CreateField( Name as String, Type as Integer,
        [Param1 as Integer], [Param2 as Integer], [Method as String] ) as VField

CreateField( Name as String, type as Integer,
        Target as VBaseObject, Control as Integer ) as VField

DeleteField( inFld as VField )

ChangeType( inFld as VField, NewType as Integer, Param as Integer ) as VField
```

Class description

Each VBaseObject manages a table of the database.
Each VBaseObject must have at least one field but not more than 65535 fields.

Record Methods

SetBlank
Each VBaseObject 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.

AddRecord
Adds a new record to the table with the current values in the memory buffer of this BaseObject. So at first we need to assign values to the fields for the new record and then call AddRecord():

```
thePerson.SetBlank
thePerson.FirstName.Value = “Jojn”
thePerson.LastName.Value = “Roberts”
thePerson.AddRecord
```

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

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

DeleteRecord as Boolean
Deletes the current record of BaseObject.
Returns TRUE if the operation was successful.
Returns FALSE is record was not deleted, e.g. it was locked or not exists.

If after deleted record exists next record it becomes current. Otherwise the previouse record become current. If BaseObject becomes empty then current record is undefined.

DeleteAllRecords
Deletes all records in the BaseObject.
BaseObject becomes empty, the current record – undefined.

Flush
This method flushes all unsaved information of this BaseObject from the cache to the disk.

Navigation Methods

The navigation methods of VBaseObject class will seldom be used. The VCursor 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.

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.

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.

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.

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

GoToRecID(inRecID as Integer) 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.

Field Methods

Field(inIndex as Integer) as VField
Field(inName as String) as VField
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 nil.

```
fld = theBaseObject.Field( 2 );
```

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

```
dim fld as VField
dim fldString as VString

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

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

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

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 into a new version)

The main purpouse for these methods – is to change the size of the record of the Table. 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.

```
CreateField( Name as String, Type as Integer,
            [Param1 as Integer], [Param2 as Integer], [Method as String] ) as VField

CreateField( Name as String, type as Integer,
            Target as VBaseObject, Control as Integer ) as VField
```

Method CreateField appends to this BaseObject (Table) a new field (column). This operation can be made even if there are records in the Table. This is a very fast operation.

For all **numeric** fields, **VDate** and **VTime** you should use first method with 2 parameters:

```
boPerson.CreateField( “BornDate”, kV_TypeDate  )
```

Constants kV_TypeBoolean, kV_TypeByte, ... are defined in the module “ValentinaUtilities”.

For **String** and **VarChar** field you must specify Param1 as MaxLength, and you can specify Param2 as Language (default value -1):

```
boPerson.CreateField( “FirstName”, kV_TypeString, 25  )
boPerson.CreateField( “FirstName”, kV_TypeString, 25, 3  )      // German
```

For **BLOB** field you should specify Param1 as the size of the segment in bytes:

```
boPerson.CreateField( “Photo”, kV_TypeBLOB, 50 * 1024  )
```

For **TEXT** field you should specify Param1 as the size of the segment in bytes and you can specify Param2 as Language:

```
boPerson.CreateField( “Notes”, kV_TypeText, 5 * 1024  )
boPerson.CreateField( “Notes”, kV_TypeText, 5 * 1024,  3  )
```

Finaly for **ObjectPtr** field you must use the second form of CreateField method in which you must specify the (parent) BaseObject to which it is pointing and deletion control (about deletion control see chapter “VObjectPtr class”):

```
boTask.CreateField( “person_ptr”, kV_TypeObjectPtr, thePerson,  kV_Cascade  )
```

If you want create a **Method of BaseObject** (i.e. virtual field) then you need specify the parameter ‘Method’:

```
boPerson.CreateField(, "FullName", kV_TypeVarChar,  504, -1,
                        "CONCATE(FirstName, ' ', LastName)" )
```

DeleteField(inFld as VField)
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.

ChangeType(inFld as VField, NewType as Integer, Param as Integer) as VField
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 and VarChar fields Param is MaxLength.
For BLOB an its subtypes (Text, Picture) Param is SegmentSize.
For the remaining types of fields, Param is ignored and should be zero.

Class VField

Properties

```
Name          as String          // up to 32 bytes
Type          as Integer (r/o)   // Constants are defined in the
                                 // ValentinaUtilities module

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
IsMethod      as Boolean         // true if the field is method.

IsNull        as Boolean         // (r/o) true if the current value of the field is
                                 //  NULL.
```

Methods

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

SetMethod( inText as String )    // set text of the Method
GetMethod as String              // returns text of the Method

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

Class Description

This is a base abstract class for all other types of fields. You must not create an instance of this class! Each field must have a unique name (case insensitive) in the BaseObject.

Using VBaseObject.GetField() or VCusor.Getfield() you can get reference of a VField. There is no any difference between VField of BaseObject and VField of Cursor.

If you need get access to properties of VField subclasses then you need do type casting to that class. For example, if you have got reference of string Field and want get access to property MaxLength of class VString you can write:

```
dim fld as VField
dim str_fld as VString

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

Properties Description

Name

Each field should have unique name in scope of BaseObject. Maximal length of name is 32 bytes.

Type

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

Each field has several flags, which define its behavior:

Indexed

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

Unique

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

Nullable

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.

IsMethod

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

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 column of table, IsNull is a property of the current record.

Methods description

SetBlank

Clear value of any field. For numeric field it set it to zero, for String fields it set it empty string. If a field is Nullable then set its value to NULL.

GetString as String
SetString(inValue as String)

These methods allow you to get or set a field value using strings regardless of the assigned field type. When assigning a value to a field, Valentina will convert the string to the appropriate type.

This is particularly useful for the VDate and VTime fields, because Valentina uses the settings of the control panels for conversion. This allows you to display the correct international format. You can change format of Date/Time using property VDataBase.DateFormat.

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.

SetMethod(inText as String)

If you want create in a BaseObject virtual field, i.e. Method then you must set text for this method after field constructor of field and BEFORE you will Create database on disk.

Example:

```
Person( inDB as VDatabase )
        mFirstName = new VVarChar( “FirtName”, 504  )
        mLastName = new VVarChar( “LastName”, 504 )
        mFullName = new VVarChar( “FullName”, 504 )
        mFullName.SetMethod( “CONCAT(FirstName, ‘ ‘, LastName)”  )
end sub
```

GetMethod as String

Returns string of text for BaseObject Method.

Numeric Fields

The numeric field classes are described below. They differ only in the type of property ‘Value’. For example, VBoolean value has type boolean while VDouble has type double. Each class has a constructor where you should specify the name of this field (column).

All of these classes are subclasses of VField. This means that they inherit all the properties and the methods of VField class. All these classes have constructors with 2 parameters: the name of the column and the Flags.

In the module “ValentinaUitilities” are defined the following constants:

```
kV_Indexed          = 1
kV_Unique           = 2
kV_Nullable         = 4
kV_IndexByWords     = 8
```

If you specify Flags as 0 then the field will not be indexed, will not be unique, and will not be nullable.

```
fld = new VByte( “byte_fld”,  0 )
fld = new VByte( “byte_fld”,  kV_Indexed )
fld = new VByte( “byte_fld”,  kV_Indexed + kV_Unique )
fld = new VByte( “byte_fld”,  kV_Indexed + kV_Nullable )
```

We use these Flags in the constructor just to write less code. If we did not use flags then we would need to write several lines of code instead of one:

```
fld = new VByte( “byte_fld” )
fld.indexed = true
fld.nullable = true
```

NOTE: constants kV_Indexed, kV_Uniuqe and kV_Nullable should be used in the constructors only. At runtime you should change properties of the field:

```
fld.indexed = true.
```

Class VBoolean

constructor VBoolean(Name as String, [Flags as Integer])
properties Value as Boolean

Class VByte

constructor VByte(Name as String, [Flags as Integer])
properties Value as Integer

Class VShort

constructor VShort(Name as String, [Flags as Integer])
properties Value as Integer

Class VUShort

constructor VUShort(Name as String, [Flags as Integer])
properties Value as Integer

Class VLong

constructor VLong(Name as String, [Flags as Integer])
properties Value as Integer

Class VULong

constructor VULong(Name as String, [Flags as Integer])
properties Value as Integer

Class VFloat

constructor VFloat(Name as String, [Flags as Integer])
properties Value as single

Class VDouble

constructor VDouble(Name as String, [Flags as Integer])
properties Value as double

Class VDate

constructor VDate(Name as String, [Flags as Integer])

properties

Year	as Integer	// any year between $-2^{22}..+2^{22}$
Month	as Integer	// 1.12
Day	as Integer	// 1..31

methods Set(year as Integer, month as Integer, day as Integer)

Class VTime

constructor VTime(Name as String, [Flags as Integer])

properties

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

methods Set(hour as Integer, minute as Integer, second as Integer)

The classes VDate and VTime differ from the group of numeric fields in that they have a complex “value” representing several properties. Also, they have the method Set() that allows the setting of all three properties in one call.

Class VDateTime

constructor VDateTime(Name as String, [Flags as Integer])

properties

Year	as Integer	// any year between $-2^{22}..+2^{22}$
Month	as Integer	// 1.12
Day	as Integer	// 1..31
Hour	as Integer	// 0..23
Minute	as Integer	// 0..59
Second	as Integer	// 0..59

methods SetDate(year as Integer, month as Integer, day as Integer)
SetTime(hour as Integer, minute as Integer, second as Integer)

Class VString
Class VVarChar

parent class VField

- VString and VVarChar classes have the same API (except constructor).

constructor
VString(Name as String, MaxLength as Integer, Language as Integer, [Flags as Integer])
VVarChar(Name as String, MaxLength as Integer, Language as Integer, [Flags as Integer])

properties

MaxLength	as Integer	// maximal length of string which can be stored
Language	as Integer	// number of corresponded itlb2 resource
IndexByWords	as Boolean	// if true then indexed by each word of the string
Value	as String	

methods
ReadRawData as String
WriteRawData(inData as String)

Class Description

This type of field is used for storing strings in a database.

Properties Description

MaxLength
Maximal length of the String and VarChar field can be in range 1 .. 65535 bytes.

Language
You can specify the language which will be used for indexing this field.

EXAMPLE:

```
dbReference = new VString( “REFERENCE”,  9, -1, kV_Indexed + kV_Unique)
dbVerse     = new VString( “VERSE”,  400, -1 )

dbReference = new VVarChar( “REFERENCE”,  9, -1, kV_Indexed + kV_Unique)
dbVerse     = new VVarChar( “VERSE”,  400, -1 )
```

Using the flag IndexByWords, you can specify that the String or VarChar field should be indexed by each word.

To set/get the value of String field you should use the property Value:

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

Methods Description

ReadRawData() as String
WriteRawData(outData as String)

These methods allow you to use VString field to store raw data (which contains 0 chars).

NOTE:
1) if you use this methods then you must not index this field,
2) This field will not be correctly displayed in the Valentina DBMS application.

Class VBLOB

parent class VField

constructor
VBLOB(Name as String, SegmentSize as Integer)

properties
SegmentSize as Integer // in bytes, N * 1024

methods

GetDataSize as Integer
DeleteData

ReadRawData as String
ReadRawData(inHowMuch as Integer, inOffset as Integer) as String

WriteRawData(inData as String)
WriteRawData(inData as String, inOffset as Integer)

SetPicture(inData as Picture)
GetPicture as Picture

Class Description

This type of field is intended for storing large chunks of data (e.g., pictures, text, movies, etc.).

Constructors of BLOB fields don't have parameter Flags, because a BLOB can't be indexed, unique or nullable.

Properties Description

SegmentSize
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 10 KB.

Methods Description

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

DeleteData
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.

ReadRawData as String
ReadRawData(inHowMuch as Integer, inOffset as Integer) as String

WriteRawData(inData as String)
WriteRawData(inData as String, inOffset as Integer)

- These methods allow you to store in the BLOB field any raw 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:

```
// we will use second form here
    dim s1, s2, s3, s4 as String
    s1 = "aaaaaa"          // 6 chars
    s2 = "bbbbbbbb"        // 8 chars

    blob_fld.WriteRawData( s1 )
    blob_fld.WriteRawData( s2, 6 )
    ...
    s3 = blob_fld.ReadRawData( 6, 0 )
    s4 = blob_fld.ReadRawData( 8, 6 )
```

Adding and updating of records with BLOB

There are 2 cases:
1) if the value of a reference in the table-record is NULL, then assigning a value to it will cause a new BLOB-record to be created;
2) if the value of a reference in the Table-record is not NULL, i.e. current record has some BLOB-context for this field, then assigning a new value will cause the old BLOB-record to be updated with the new value.

So, if you want to add a new record to the databasewith a BLOB-field, you must be sure that the referenced BLOB-field is NULL. For this you must call 'SetBlank' for the BaseObject to clear the buffer of the record, or call 'SetBlank' for that BLOB-field, so the value of that one field will be cleared.

If you want to update a value of a BLOB-field of the current record then you just assign the new context to it. After that you should call 'UpdateRecord' of the BaseObject.

If you update the context of a field with a NULL value, the old context of the field will be deleted and the value of the BLOB-field will become NULL.
NOTE: the table record still exists!

Example:

To add a record:

```
mfPhoto.SetBlank
mfPhoto.SetPicture( thePicture )
AddRecord
```

To update a record:

```
mfPhoto.SetPicture( theNewPicture )      // replaces oldPicture
UpdateRecord
```

This clears the context of the current VBLOB-record but doesn't delete the table-record:

```
mfPhoto.DeleteData     // there is no more Photo associated  with this Person.
UpdateRecord
```

Class VText

parent class VBLOB

constructor
VText(Name as String, SegmentSize as Integer, Language as Integer, [Flags as Integer])

properties
Language as Integer
Value as String

Class Description

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

It can be indexed like VString but it has no limit on the size of the content because it's type is like VBLOB. If a VText field is indexed, then it is always indexed by words; so there is no property 'IndexByWords'.

You can specify the language which will be used for indexing this field. This feature allows you to develop international (e.g., Japanese) ready applications. To determine the local language, the MacOS uses the ID of the itl2 system resource. For example, the value of this resource is 3 for the German language. You can use toolbox routines to get the ID of the itl2 resource by its name (like "German" or "Japanese").

If the language for field is -1, then a byte-to-byte method of comparison is used. This is the fastest method but it can be used only for English and some Romance languages.

String and Text fields can be searched using a regular expression search. The syntax of RegEx is similar to the syntax of RegEx found in Userland's Frontier. Its description can be found in the folder "Syntax of RegEx". To search by RegEx you use the "LIKE" word in the SQL query.

Adding and updating of records with a Text field

A VText field has a property ‘Value’ which works a little differently from the property ‘Value’ of usual fields (see the VBLOB field description also). There are 2 cases:

1) if the ‘Value’ is NULL and a new value is assigned to it, then a new VBLOB-record will be created;
2) if the ‘Value’ is not NULL and you assign a new value, then the existing VBLOB-record will be updated with the content of the new value.

So, if you want to add a new record to the database with a VText-field, you must be sure that value of table-record of VText-field is NULL. To do this you must first call ‘SetBlank’ for BaseObject to clear buffer of record, or call ‘SetBlank’ for that VText-field so that the value of that one field will be cleared.

If you want to update the value of the VText-field of the current record then you just assign the new content to it. After that, as usual, you should call ‘UpdateRecord’ for the BaseObject.

If you update the context of a VText-field with NULL then the old content of the field will be deleted and the value of VText-field becomes NULL.
NOTE: the table record still exists!

class VPicture

parent class VBLOB

constructor
VPicture(Name as String, SegmentSize as Integer)

methods

SetPicture(inData as Picture, [Quality as Integer])
GetPicture as Picture

Class Description

Picture field is a special BLOB field which can storepictures in the different formats. On default it stores Pictures with JPG compression.

Advanced information
1) This field must get and returns back:
- on MacOS PICT handle.
- on Windows DIB handle.

2) This is the regular BLOB field which stores data in the next format:
* PicType (4 bytes)
* PicSize (4 bytes)
* Picture itself.

Methods Description

SetPicture(inData as Picture, Quality as Integer)
Stores Mac PICT to the database with JPG compression. Parameter Quality can be in range 0..100 and specify quality of jpeg compression.

GetPicture as Picture
Read picture with JPG compression from database and returns it as PICT with JPG comression, i.e. this methid don't decomress PICT. You can use it as normal picture because MacOS will work with it correctly.

NOTES: this methods use QuickTime for compression.

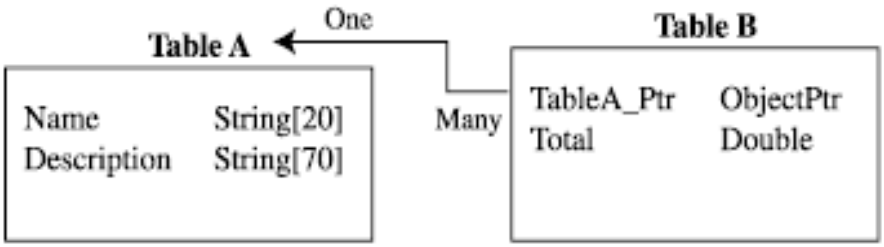
Class VObjectPtr

parent class VField

constructor
VObjectPtr(Name as String, PointedObject as VBaseObject, DeletionControl as Integer)

properties

Value	as Integer	// here is stored RecID of pointed record
PointedObject	as VBaseObject	
DeletionControl	as Integer	



Parent Table,
One Table

Child Table,
Many Table

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

- In SQL this is called a FOREIGN KEY
- In 4Dimention, FileMaker and MS Access this is called as Relation.

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

```
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. Therefore, you should never define ObjectPtr field as nullable, sinse it is possible to search for records where ptr =0.

The ObjectPtr field must know the pointed object (parent object) and deletion control to have correct behavior.

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

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 the rule, which defines the behavior on deletion of record. There are three ways deletions are handled.

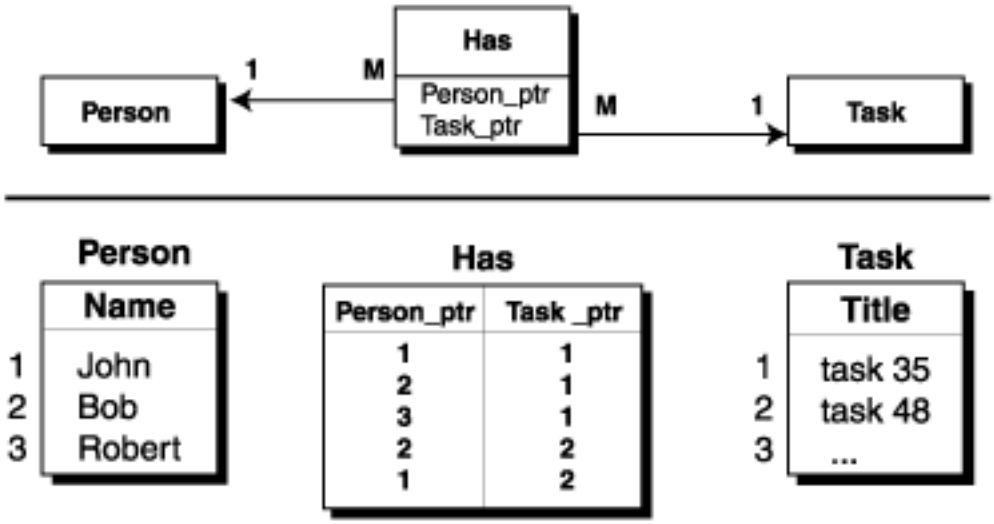
Leave related Many records:
From the database only the “One” record is deleted, and the ObjectPtr field of the related many records is automatically set to 0. In the “ValentinaUtilities” module the defined constant kV_SetNull is for this case.

Delete related Many records:
The “One” and “Many “ components are all deleted. If a Many record also have related Many records in the third Table(s) then they are also deleted in a "cascade delete". In the “ValentinaUtilities” module defined constant kV_Cascade is for this case.

Can not delete if related Many:
The deletion of the One record is not allowed if there is at least one related Many record. In the “ValentinaUtilities” module defined constant kV_Restrict is for this case.

As we have noted above ObjectPtr field can be used to establish MANY to ONE relation. Actually it can be used to establish ONE to ONE relation also. For this you should specify ObjectPtr field as unique. Valentina can use this information to optimize query resolving.

Besides using ObjectPtr field you can establish a Many to Many relation between 2 tables. For this you need to create additional third table - Link as shown on the picture:



Class VCursor

properties

```
DataBase        as VDataBase    // Databaseof this Cursor.
SQLstring       as String       // SQL string of this Cursor

FieldCount      as Integer  // (r/o) number of selected fields for this Cursor.
RecordCount     as Integer  // Number of selected records, can be reduced.

CurrentPosition as Integer  // Current position in the cursor

ReadOnly        as Boolean  // (r/o) true if records in this cursor can't be changed,
                            // i.e. you can't add/update/delete records.
```

methods

```
VCursor( inDatabase as VDatabase, inSQLstr as String )

// Field Methods:
Field( Index as Integer ) as VField
Field( Name as String ) as VField

// Navigation Methods:
FirstRecord as Boolean
LastRecord as Boolean
PrevRecord as Boolean
NextRecord as Boolean

// Record Methods:
SetBlank                // blank the memory buffer of the record
Add                     // adds a new record to the cursor
Update                  // updates current record of cursor
Delete as Boolean       // deletes current record of cursor
DeleteAll               // deletes all records of cursor

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

ImportText(     File as FolderItem,
                FieldDelimter as String,
                LineDelimter as String )

ExportText(     File as FolderItem,
                FieldDelimter as String,
                LineDelimter as String )
```

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.

Creation of cursor

VCursor(inDatabase as VDatabase, inSQLstr as String)
This constructor provides you the second way to create Cursor (the first one is via the method VDatabase.SQLSelect(). If you want to define a subclass of VCursor than you need use constructor of VCursor.

Example:

```
sub myCursor
        VCursor(inDB, inSQL)        // init parent class.
end sub
```

Properties description

FieldCount
Here you can find how many columns this cursor has.

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

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 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 only; otherwise it is false.

Field Methods

Field(Index as Integer) as VField
Field(Name as String) as VField
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.

```
dim i, Records as Integer
LastName as String
dim cur as VCursor

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

Records = cur.RecordCount
for i = 1 to Records
        cur.CurrentPosition = i
        LastName = cur.Field( "last_name" ).GetString
next
```

Type casting Methods

After you get the field as VField you can use type casting to get a reference on the actual class of the field.

As described in the paragaph "VField" you may need do type casting
a) to access value of field not as string but as number to be about 20 times faster.
b) to access properties of VField subclasses.

To save yout type class VCursor has set of methods which do this type casting for you.

BooleanField(Index as Integer) as VBoolean
BooleanField(Name as String) as VBoolean

ByteField(Index as Integer) as VByte
ByteField(Name as String) as VByte

ShortField(index as integer) as VShort
ShortField(name as string) as VShort

UShortField(index as integer) as VUShort
UShortField(name as string) as VUShort

LongField(index as integer) as VLong
LongField(name as string) as VLong

ULongField(index as integer) as VULong
ULongField(name as string) as VULong

LLongField(index as integer) as VLLong
LLongField(name as string) as VLLong

ULLongField(index as integer) as VULLong
ULLongField(name as string) as VULLong

FloatField(index as integer) as VFloat
FloatField(name as string) as VFloat

DoubleField(index as integer) as VDouble
DoubleField(name as string) as VDouble

DateField(index as integer) as VDate
DateField(name as string) as VDate

TimeField(index as integer) as VTime
TimeField(name as string) as VTime

DateTimeField(index as integer) as VDateTime
DateTimeField(name as string) as VDateTime

StringField(index as integer) as VString
StringField(name as string) as VString

VarCharField(index as integer) as VVarChar
VarCharField(name as string) as VVarChar

BlobField(index as integer) as VBlob
BlobField(name as string) as VBlob

TextField(index as integer) as VText
TextField(name as string) as VText

PictureField(index as integer) as VPicture
PictureField(name as string) as VPicture

ObjectPtrField(index as integer) as VObjectPtr
ObjectPtrField(name as string) as VObjectPtr

So you have several ways to work with fields of cursor:
Let you have variables, which defined as:

```
dim fld as VField
dim fldLong as VLong
```

now you can write on of the following:

```
fld = curs.Field( “long_fld” )
VLong( fld ).value = 5
```

```
VLong( curs.Field( “long_fld” ) ).value = 5
```

```
curs.LongField(“long_fld”).value = 5
```

As you can see the last form is the shortest.

Navigation Methods

FirstRecord as Boolean
Go to the first logical record of Cursor. Returns true if the first record is found.

LastRecord as Boolean
Go to the first logical record of Cursor. Returns true if the last record is found

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.

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
                // work here
        Loop Until myCursor.NextRecord = false
end if
```

Just for you info:
The result of this methods can be reached also with using of property ‘CurrentPosition’ in conuction with ‘RecordCount’, but this is less effective way.

Example:

```
if( myCursor.RecordCount > 0 )
        myCursor.CurrentPosition = 1
        For i = 1 to myCursor.RecordCount
                // work here
                myCursor.CurrentPosition = myCursor.CurrentPosition + 1
        Next
end if
```

Import/Export

ImportText(File as FolderItem,
FieldDelimter as String,
LineDelimter as String)

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

The parameters FieldDelimter and LineDelimter 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.

ExportText(File as FolderItem,
FieldDelimter as String,
LineDelimter as String)

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.

Appendix A: Valentina Utilities module

You must drag the module Valentina Utilities into your project to have access to the constants defined in it.

Constants of types of fields

kV_TypeBoolean	= 2
kV_TypeByte	= 3
kV_TypeShort	= 4
kV_TypeUShort	= 5
kV_TypeLong	= 6
kV_TypeULong	= 7
kV_TypeLLong	= 27
kV_TypeULLong	= 28
kV_TypeFloat	= 8
kV_TypeDouble	= 9
kV_TypeString	= 10
kV_TypeVarChar	= 26
kV_TypeDate	= 12
kV_TypeTime	= 13
kV_TypeDateTime	= 14
kV_TypeBLOB	= 21
kV_TypeText	= 17
kV_TypePicture	= 19
kV_TypeObjectPtr	= 15

Constants for parameter Flags in the constructors of the fields

kV_Indexed	= 1
kV_Unique	= 2
kV_Nullable	= 4
kV_IsMethod	= 16

Constants for DeletionControl of field ObjectPtr

kV_SetNull	= 0
kV_Cascade	= 1
kV_Restrict	= 2