VALENTINA 5

for Director Reference

Paradigma Software, Inc.
www.paradigmasoft.com

Contents

Introduction

Valentina for Macromedia Director Xtra (V4MD) brings the power of Paradigma Software's 'object-relational' database engine to Macromedia's flagship multimedia development tool, Director.

V4MD extends Director's features and helps you to build fast, database-driven multimedia titles. You can use V4MD to create interactive applications such as electronic catalogs, storybooks, kiosks, training materials, sales materials, games, and more. You can use it as a back-end to your multimedia projects, to efficiently manage a text, a numeric data, dates, images, sound clips as well as any type of media that Director can store in its members.

As a member of the Valentina line of object-relational database products, V4MD allows you to unify your database development, with the across-the-line standard database format.

V4MD employs the same cross-platform and a cross-product database format, shared by all products in the Valentina product line, allowing databases to built for use with one development environment to be used with any other development environment supported by Valentina:

- Valentina C++ SDK (Mac/Win32)
- Valentina .NET (C#, VB7, ...)
- Valentina COM (VB6, ...)
- Valentina for REALbasic
- Valentina for Macromedia Director (Mac/Win32)
- Valentina Studio (GUI, schema/mananment, data browsing, import/export, ODBC import/export)
- Valentina Server (Office and Embedded versions)

Getting Started

Valentina for Director Xtra is a scripting Xtra, i.e. it extends the Lingo with new functions and data types.

In this part we will describe SIMPLEST template of usage of Valentina for Director. Valentina offers very powerful and yet simple API to work with database. To start development a Lingo developer need to learn just ONE (!) important function (and 4 helper functions which can be inserted by copy-paste into your code).

The general template of work with Valentina database looks as simple as next:

```
Valentina.Init()
        gDB.Open()

        res = gDB.SqlQuery( "some SQL command1" )
        res = gDB.SqlQuery( "some SQL command2" )
        res = gDB.SqlQuery( "some SQL command3" )

        gDB.Close()
Valentina.Shutdown()
```

1) first of all you need to initialize Valentina engine:

```
global Valentina
Valentina = new( xtra "Valentina" )
Valentina.Init()
```

2) then you can create a database object and open an existed database:

```
global gDB
gDB = new ( xtra "VDatabase" )
gDB.Open( the moviepath & "mydb" )
```

3) Suppose your database have table Person with fields fname and lname. The following SQL SELECT command will find and return all parsons with name Jon:

```
res = gDB.SqlQuery( "SELECT * FROM Person WHERE fname = 'Jon' " )
put res
```

The result will be a property list with information about error (if any) and list of found records. For example it can look as:

```
[ #errNumber: 0, #errNumberHex: 0, #errString: void, #rowsChanged: 0,
 #columns: ["fname", "lname"], #rows: [ ["Jon", "Jonson"] ] ]
```

4) when you have finished working with the database you should close it.

```
gDB.Close()
```

5) finally you should shutdown database engine.

```
Valentina.Shutdown()
```

Now you know enough to start and finish work with Valentina for Director. Using single function SqlQuery() you can do practically anything with database: create/drop/alter tables and columns, insert/delete/update records, do searching and sorting. Read the next page for more information.

EXAMPLE on this function is located at V4MD/Examples/SQL_way/GettingStarted

What Next

As you have see in the previous part you can do practically all work with database using single sqlQuery() method. This is because this method uses the power of SQL language. Valentina supports strictly SQL92 standard, some features of SQL1999/2003 and adds own extensions.

To go by this SIMPLEST way you need read about sqlQuery() in this document, and check Valentina SQL as deeply as require your SQL skills.

Except this simplest way Valentina offer very reach and powerful API to work with database. Here we can expose 2 main advanced ways to work with Valentina.

1) SQL way that use VDatabase + VCursor Xtras.

In this way you still use SQL, but you get results of SELECT query as VCursor xtra that allows you to work with separate records and fields, add/delete/update records and other advanced features.

2) API way. This way allows you to do any operation with database without SQL absolutely.

To use this way you need become familiar with set of Xtras that V4MD offers. See the next part "Architecture of Valentina for Director" to get more info.

Architecture of Valentina for Director

Valentina for Director is represented as set of Xtras, each Xtra have set of handlers and properties that form its semantic.

The top Xtra is **Valentina Xtra**. You need create one and only one instance of this Xtra. It is good idea put it into global variable with name Valentina.

```
Valentina = new ( Xtra "Valentina" )
```

This Xtra has set of handlers that control the whole engine (Init(), Shutdown()) or are utility handlers (EscapeString()).

VConnection Xtra - represents a connection to a remote Valentina Server.

VDataBase Xtra - contains handlers to manage database and its sub-elements: Tables, Links, Indexes. Valentina can work with a multiple open databases. You will use one instance of VDataBase Xtra for a database.

```
db = new ( Xtra "VDatabase", #kLocal, #kDisk )
```

VCursor Xtra - contains handlers that allow to navigate and modify(!) records of the result of SQL SELECT query. You can have many cursors per database at the same time. Each cursor keep record-level locks. You create cursors using VDatabase.SqlSelect() handler:

```
curs = db.SqlSelect( "SELECT * FROM T1", #kServerSide, #kReadWrite )
```

NOTE: using VDatabase and VCursor you can do all job with database using SQL commands and Relational data model. But Valentina for Director also have another set of Xtras, which allow you manage database in NON-SQL way using Object-Relational data model of Valetnina. So let's take a look on these Xtras.

VTable Xtra - represents a table of database. It have handlers and properties to do any operation you need for tables: navigate records, read/write records, manage fields of tables, so on.

VField Xtra - represents a column/field of a database table. It have handlers and properties to manage Value of a current record and different column settings.

VSet Xtra - represents a set of found records as result of VField.Find() handlers. This Xtra also provides handlers to do set operations as Union, Intersection, Difference.

VSetIterator Xtra - is partner of VSet Xtra and allows to iterate the found set item by item.

VLink Xtra - represents a new abstraction of Valentina Object-Relational data model, that allow link tables. Valentina model offers Foreign Key, ObjectPtr and BinaryLink types of link currently. Read in the Valentina WIKI sections "Valentina Kernel" and "Valentina SQL" for details.

VServer, VDatabaseInfo, VClientInfo Xtras - allow to control a Valentina Sever.

VProject and VReport Xtras - allow to open Valentina project prepared in Valentina Studio PRO,

Valentina Symbols

Valentina for Director uses many symbol constants to simplify coding for a developer. Below you can see full list of symbols that Valentina understands.

Note, that numeric values are given only for information. You should never use numeric values.

OS types

#kOSDefault	= 0
#kOSMac	= 1
#kOSWindows	= 2
#kOSUnix	= 3

DateFormat types

#kMDY	= 0
#kDMY	= 1
#kYMD	= 2

DebugLel types

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

DbMode types

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

Flag types

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

OnDeletion types

#kOnDelete_NoAction	= 0
#kOnDelete_SetNull	= 1
#kOnDelete_Cascade	= 2
#kOnDelete_Restrict	= 3
#kOnDelete_Default	= 4

OnUpdate types

#kOnUpdate_NoAction	= 0
#kOnUpdate_SetNull	= 1
#kOnUpdate_Cascade	= 2
#kOnUpdate_Restrict	= 3
#kOnUpdate_Default	= 4

RecursionDirection types

#kFromParentToChild	= 0
#kFromChildToParent	= 1

StorageType types

#kStorage_Default	= 0
#kStorage_Disk	= 1
#kStorage_RAM	= 2

Tablekind types

#kTblPermanent	= 0
#kTblTemporary	= 1

CursorLocation types

#kClientSide	= 1
#kServerSide	= 2

LockType types

#kNoLocks	= 1
#kReadOnly	= 2
#kReadWrite	= 3

CursorDirection types

#kForwardOnly	=1
#kRandom	= 2

LinkType types

#kMany	= 0
#kOne	=1

FieldType types

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

DumpType types

#kSQL	= 1
#kXML	= 2

DumpData types

#kStructureOnly	= 1
#kStructureAndRecords	= 2
#kRecordsOnly	= 3

VerboseLel types

#kVerbose_None	= 0
#kVerbose_Low	= 1
#kVerbose_Normal	= 2
#kVerbose_High	= 3
#kVerbose_VeryHigh	= 4

ColAttribute types

#kFrenchCollation	= 0
#kAlternateHandling	= 1
#kCaseFirst	= 2
#kCaseLel	= 3
#kNormalizationMode	= 4
#kStrength	= 5
#kHiraganaQuaternaryMode	= 6
#kNumericCollation	= 7

ColAttributeValue types

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

PictType types

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

ReportPrintType

#kToPDF
#kToHTML
#kToPicture_JPG
#kToPicture_JPG
#kToPicture_JPG

Valentina Xtra

Properies

integer cacheSize (r/o)
integer databaseCount (r/o)
VDatabase database(inIndex) (r/o)
symbol debugLevel (r/w)
integer lastError (r/o)
string lastErrorString (r/o)
symbol lastErrorSymbol (r/o)
VConnection localConnection (r/o)
integer logToMessageWindow (r/w)
string version (r/o)

Initialization methods

```
init(   integer inCacheSize,
        string inMacSerialNumber = "",
        string inWinSerialNumber = "",
        string inLinSerialNumber = "" )

initClient()

initReports(
        string inMacSerialNumber = "",
        string inWinSerialNumber = "" )

shutDown()
shutDownClient()

convert_1_2(
        string inOldDb_Version1,
        string inNewDb_Version2,
        boolean inLoadRecords,
        integer inNewSegmentSize = 0 )
```

Utility methods

```
logToFile( integer inOn, integer inFlush )

escapeString( string inStr, integer inForRegEx )
setExtensions( list inExtensions )

getDatabaseFormatVersion( string inVdbFile )
getCurrentFormatVersion()

getSchemaVersion( string inVdbFile )
getDatabaseMode( string inVdbFile )
getIsStructureEncrypted( string inVdbFile )
```

Properties

integer cacheSize (r/o)

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

Example:

```
size = Valentina.cacheSize
```

symbol debugLevel (r/w)

This allows you to set the debug level in Valentina for Director.
Any debug level above 0 will create a file which outputs the results. The file will be named “V4MD_Log.txt”. It will be created in the same directory as the project. The only exception is for Mach-O builds in Mac OS X where it will be created one level inside the executable.
The valid values are DebugLel types:

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

Example:

```
Valentina.debugLevel = #kLogParams
-- do some work where you search for a bug
Valentina.debugLevel = #kLogErrors
```

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

• If you have a level of debugging >= 1 then Valentina will check all dbRef, boRef and CursorRef on validity (note, that fldRef Valentina will not check in any case).

integer lastError (r/o)

Returns the number of the last error for this database. If there is no error then this property returns 0. You can/should check an error after each operation of Valentina.

Example:

```
err = Valentina.lastError
```

integer databaseCount (r/o)

Returns: integer

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

Example:

```
db_count = Valentina.databaseCount
```

VDatabase database(inIndex) (r/o)

Returns: VDatabase

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

See also:

Valentina.database(inIntIndex)

Example:

```
db = Valentina.database( inIntIndex )
```

string lastErrorString (r/o)

Returns a string that describes the last error for this database.

Example:

```
errStr = Valentina.lastErrorString
```

symbol lastErrorSymbol (r/o)

Returns the error as symbol. This symbol is the same as error code sontant in the XML files located in the folder "vcomponents/vresources", for example it can be "ERR_TABLE_NAME_NOT_UNIQUE". This function allows you to write code that checks errors not by numbers but by symbol constants.

Example:

```
errSym = Valentina.lastErrorSymbol
if errSym = #ERR_TABLE_NAME_NOT_UNIQUE
...
```

integer logToMessageWindow (r/w)

If TRUE then Valentina prints debug messages into the Message Window of Director.

Example:

```
Valentina.logToMessageWindow = true
```

VConnection localConnection (r/o)

Returns: VConnection

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

See also:

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

Example:

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

string version (r/o)

Returns the version of the Valentina engine.

Example:

```
ver = Valentina.version
```

Initialization Methods

Init(
integer inCacheSize,
string inMacSerialNumber = "",
string inWinSerialNumber = "")

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

Returns: VOID

Before you call any handler of Valentina's xtras, you must initialize the Valentina kernel. The most obvious way to do this on the movie startup.

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

To improve the 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.

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

Example:

```
global Valetnina
on startMovie
	global Valentina

	Valentina = new (Xtra "Valentina" )
	Valentina.init(  9 * 1024 * 1024, "",  "" )
end
```

---

initClient()

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

Example:

```
global Valetnina
on startMovie
	global Valentina

	Valentina = new (Xtra "Valentina" )
	Valentina.initClient()
end
```

```
InitReports(
        string inMacSerialNumber = "",
        string inWinSerialNumber = "" )
```

Parameter	Description
inMacSerialNumber	The serial number for use under Mac OS or "" in the demo mode.
inWinSerialNumber	The serial number for use under Windows or "" in the demo mode.

Returns: VOID

Description:

Initializes the work with Valentina reports for your application.

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

Example:

```
global Valetnina
on startMovie
        global Valentina

        Valentina = new (Xtra "Valentina" )
        Valentina.init( 4 * 1024 * 1024, "",  "" )
        Valentina.initReports( "",  "" )
end
```

shutDown()

Returns: VOID

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

Example:

```
on stopMovie
	global Valentina

	Valentina.shutDown()
end
```

shutDownClient()

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

Пример:

```
Valentina.shutDownClient()
```

convert_1_2(
	string inVersion1DbPath,
	string inVersion2DbPath,
	integer inLoadRecords,
	integer inNewSegmentSize = 0)

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

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

Note: This function do not change the old Database.

Example:

```
db.convert_1_2( oldDB, newDB, true )
```

Utility Methods

logToFile(integer inOn, integer inFlush)

Parameter	Description
inOn	TRUE if you want log to file
inFlush	TRUE if each log to file must be flushed immediately

Returns: VOID

This function allow you set ON/OFF logging to file of Valentina debug messages. If parameter inFlush is TRUE then each log messages will be flushed to HDD immediately. This is useful in case you have crash in your application. But this slows down logging, so on default parameter inFlush is FALSE.

Example:

```
Valentina.logToFile( true, false )
```

escapeString(string inStr, integer inForRegEx)

Parameter	Description
inStr	The string to be escaped.
inForRegEx	Specifies if a string will be used with RegEx search.

Returns: string

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

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

Example:

```
res = Valentina.escapeString(  "Valentina's ( day)", 0 )
-- res is "Valentina\'s ( day)"

res = Valentina.escapeString(  "Valentina's ( day)", 1 )
-- res is "Valentina\'s \( day\)"
```

setExtensions(list inExtensions)

Parameter	**Description**
inExtensions	The list of extensions.

Returns: VOID

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

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

The list can contains up to 4 string items.

First item will be used for description file.
Second item will be used for data file.
Third item will be used for BLOB file.
Forth item will be used for index file.

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

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

Example:

```
Valentina.setExtensions(  [ "vdb", "dat", "blb", "ind"] )

Valentina.setExtensions(  [ "vdb", "", "", "tre"] )
```

getDatabaseFormatVersion(string inVdbFile)

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

Returns: integer

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

Example:

```
vers = Valentina.getDatabaseFormatVersion( path )
```

getCurrentFormatVersion()

Returns: integer

Returns the current format version of database file.

Example:

```
vers = Valentina.getCurrentFormatVersion
```

getSchemaVersion(string inVdbFile)

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

Returns: integer

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

Example:

```
SchemaVersion = Valentina.getSchemaVersion( path )
```

getDatabaseMode(string inVdbFile)

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

Returns: integer

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

Example:

```
dbMode = Valentina.getDatabaseMode( path )
```

getIsStructureEncrypted(folderItem inVdbFile)

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

Returns: boolean

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

Example:

```
fi = getFolderItem( "MyDatbase.vdb" )
isEncrypted = Valentina.getStructureEncrypted( fi )
```

VConnection Xtra

Properties

```
boolean     isConnected (r/o)
string      hostName (r/o)
integer     port (r/o)
string      userName (r/o)
```

Construction Methods

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

Methods

```
open()
close()

useSSL()
```

SQL Methods

```
sqlQuery(
        string   inQuery,
        symbol inCursorLocation  = #kClientSide,
        symbol inLockType = #kReadOnly,
        symbol inCursorDirection = #kRandom,
        list       inBinds() = [] )

sqlSelect(
        string   inQuery,
        symbol inCursorLocation  = #kClientSide,
        symbol inLockType = #kReadOnly,
        symbol inCursorDirection = #kRandom,
        list       inBinds() = [] )

sqlExecute(
        string inQuery,
        list inBinds() = [] )
```

Properties

boolean isConnected (r/o)

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

Example:

```
res = connection.IsConnected
```

string hostName (r/o)

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

Example:

```
hName = connection.HostName
```

integer port (r/o)

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

Example:

```
port = connection.Port
```

string userName (r/o)

Returns user name of this connection.

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

Example:

```
userName = connection.UserName
```

Construction Methods

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

Returns: VOID

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

Description:

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

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

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

Example:

```
connection = new ( Xtra "VConnection", "localhost", "sa", "sa" )
connection.open()
```

Example:

```
connection = new ( Xtra "VConnection", "123.456.789.123", "sa", "sa" )
```

Connection Methods

open()

Returns: VOID

Establishes a connection to a Valentina Server.

Errors:

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

Example:

```
connection = new ( Xtra "VConnection", "localhost", "sa", "sa" )
connection.open()
```

close()

Returns: VOID

Closes the connection with the server. After this any objects created in the scope of this connection (VDatabase, VTable, VCursor, …) becomes invalid and you should not try to use it, otherwise most probably you will get ERR_STREAM_XXXX error.

NOTE: VConnection.Open() and .Close() methods are similar to Init/ShutDown methods in means that you cannot reuse any objects created between these calls in the scope of this connection. Instead on the next Open() you need to create all objects again starting from VDatabase object.

Example:

```
connection = new ( Xtra "VConnection", "localhost", "sa", "sa" )
connection.open()
...
connection.close()
```

useSSL()

Returns: VOID

You must call this method right BEFORE VConnection.Open() method if you want establish a secure connection to Valentina Server. Note that VServer should listen for SSL port to be able accept such connection.

Example:

```
connection = new ( Xtra "VConnection", "localhost", "sa", "sa" )
connection.useSSL()
connection.open()
...
connection.close()
```

SQL Methods

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

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

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

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

```
sqlQuery(
        string   inQuery,
        symbol inCursorLocation  = #kClientSide,
        symbol inLockType = #kReadOnly,
        symbol inCursorDirection = #kRandom,
        list      inBinds() = [] )
```

See description of VDatabase.SqlQuery() method.

```
sqlSelect(
        string   inQuery,
        symbol inCursorLocation  = #kClientSide,
        symbol inLockType = #kReadOnly,
        symbol inCursorDirection = #kRandom,
        list      inBinds() = [] )
```

See description of VDatabase.SqlSelect() method.

```
sqlExecute(
        string inQuery,
        list inBinds() = [] )
```

See description of VDatabase.SqlExecute() method.

VDatabase Xtra

Properties

list	dateTimeFormat
integer	indexCount (r/o)
string	ioEncoding
boolean	isEncrypted (r/o)
boolean	isOpen (r/o)
boolean	isReadOnly (r/o)
boolean	isRemote (r/o)
boolean	isStructureEncrypted (r/o)
integer	lastInsertedRecID (r/o)
integer	linkCount (r/o)
string	locale
symbol	mode
string	name (r/o)
string	path (r/o)
integer	schemaVersion
string	storageEncoding
integer	tableCount (r/o)

// for CLIENT only:

Integer	responseTimeOut

Construction Methods

```
VDatabase( #kLocal, symbol inStorageType = #kDefault )
VDatabase( #kClient, VConnection inConnection )
```

Disk Methods

```
create(
        string inPath,
        symbol inMode,
        integer inSegmentSize )

open( string inPath )

close()
throwOut()
flush()

clone( string inNewDbPath,     integer inLoadRecords = TRUE, integer inDoLog = TRUE )
clone( object inNewDatabase, integer inLoadRecords = TRUE, integer inDoLog = TRUE )
```

Database Schema Methods

```
createTable(
        string inName,
        symbol inTableKind = #kTblPermanent,
        symbol inStorageType = #kDefault )

dropTable( object inTable )

createForeignKeyLink(
        string inName,
        object inKeyField,
        object inPtrField,
        symbol inOnDelete = #kOnDelete_SetNull,
        symbol inOnUpdate = #kOnUpdate_Cascade,
        integer inTemporary = FALSE )
```

```
createBinaryLink(
        string   inName,
        object  inLeftTable,
        object  inRightTable,
        symbol inLeftPower = #kOne,
        symbol inRightPower = #kMany,
        symbol inOnDelete = #kOnDelete_SetNull,
        symbol inStorageType = #kStorage_Default,
        integer inTemporary = FALSE

dropLink( object inLink )
```

Table Methods

```
table( any inIndexOrName )
```

Link Methods

```
link( any inIndexOrName )
```

SQL Query Methods

```
sqlQuery(
        string   inQuery,
        symbol inCursorLocation  = #kClientSide,
        symbol inLockType = #kReadOnly,
        symbol inCursorDirection = #kRandom,
        list       inBinds() = [] )

sqlSelect(
        string   inQuery,
        symbol inCursorLocation  = #kClientSide,
        symbol inLockType = #kReadOnly,
        symbol inCursorDirection = #kRandom,
        list       inBinds() = [] )

sqlExecute(
        string inQuery,
        list inBinds() = [] )
```

IndexStyle Methods

createIndexStyle(string inName)
dropIndexStyle(object inStyle)
IndexStyle(string inName)

Collation Methods

getCollationAttribute(symbol inColAttribute)
setCollationAttribute(symbol inColAttribute, symbol inValue)

Connection Variable Methods

getConnectionVariable(inConnVariableName as integer) as Integer
setConnectionVariable(inConnVariableName as integer, inConnVariableValue as integer)

Encryption Method

encrypt(string inKey, symbol dataKind = #kRecordsOnly)
decrypt(string inKey, symbol dataKind = #kRecordsOnly)
changeEncryption(string inOldKey, string inNewKey, symbol dataKind = #kRecordsOnly)

requiresEncryptionKey()
useEncryptionKey(string inKey, symbol dataKind = #kRecordsOnly)

Dump Methods

```
dump(
        string inDumpFilePath,
        symbol inDumpType,
        symbol inDumpData = #kStructureAndRecords,
        int inFormatDump = false,
        string inEncoding = "UTF-16" )

loadDump(
        string inDumpFile,
        string inNewDb,
        symbol inDumpType,
        string inEncoding = "UTF-16" )
```

Description

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

Properties

list dateTimeFormat

Set Date/Time format for this database. A Parameter are next in the list:

DateFormat	0 is MDY, 1 is DMY, 2 is YMD
DateSep	Character to be used as date separator, e.g. "/"
TimeSep	Character to be used as time separator, e.g. ":"
CenturyBound	Default is 20. Define behaviour of auto-correction feature (see ValentinaKernel.pdf for details).

Example:

```
dtf = db.dateTimeFormat

db.dateTimeFormat = [  #MDY, "/", ":", 20 ]
```

integer indexCount (r/o)

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

Example:

```
count = db.indexCount
```

boolean isEncrypted (r/o)

Returns TRUE if this database is encrypted.

Example:

```
encrypted = db.isEncrypted
```

boolean isOpen (r/o)

Returns TRUE if this database is open now.

Example:

```
res = db.isOpen
```

boolean isReadOnly (r/o)

Returns TRUE if this database is read only.

Example:

```
res = db.isReadOnly
```

boolean isRemote (r/o)

Returns TRUE if this database is remote.

Example:

```
res = db.isRemote
```

lastInsertedRecID()

Returns: integer

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

This function is useful mainly if you execute
db.SqlExecute("INSERT INTO T ...")

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

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

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

Example:

```
recid = db.lastInsertedRecID
```

integer linkCount (r/o)

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

Example:

```
count = db.linkCount
```

string locale

Defines the Locale string for this database. Tables and fields of this database will inherit this parameter on default.

Example:

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

symbol mode

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

You can use symbols from the DbMode types.

Example:

```
db.mode = #kDsc_Dat_Blb_Ind        -- db will have 4 files.
```

Example:

```
mode = db.mode
```

string name (r/o)

The name of database.

Example:

```
name = db.name
```

string path (r/o)

The full path to this database.

Example:

```
path = db.path
```

integer schemaVersion

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

Example:

```
ver = db.schemaVersion
```

string storageEncoding

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

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

Example:

```
encoding = db.storageEncoding
db.storageEncoding = "UTF-16"
```

integer tableCount (r/o)

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

Example:

```
count = db.tableCount
```

integer responseTimeOut

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

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

Example:

```
db.responseTimeOut = 100
```

Construction Methods

The VDatabase Xtra constructor has two forms. The first is for a LOCAL database and the second for a CLIENT database. This is the only place where your code differ for local and client/server model - when you choose type of database object.

VDatabase(#kLocal, symbol inStorageType = #kDefault)

Parameter	Description
#kLocal	Dispatcher symbol
inStorageType	Storage type for this database (see StorageType types)

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

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

Example:

```
db = new ( Xtra "VDatabase", #kLocal )
```

Example:

```
-- this example create RAM-database.
db = new ( Xtra "VDatabase", #kLocal, #kRam )
```

```
VDatabase(
        #kClient,
        VConnection inConnection )
```

Parameter	Description
#kServer	Dispatcher symbol
inConnection	VConnection object.

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

Example:

```
conn = new( Xtra "VConnection", "somecompany.com", "sa", "sa" )
conn.Open()

remote_db = new( Xtra "VDatabase", #kClient,  conn )
```

Disk Methods

```
create(
        string inPath,
        symbol inMode,
        integer inSegmentSize )
```

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

Returns: VOID

Creates a new, empty database on disk.

Note: After creation, the database is already open.

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

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

Example:

```
db.create( file, #kDscDatBlb_Ind, 32 * 1024 )
```

Example:

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

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

open(string inPath)

Parameter:	**Description:**
inPath	VDatabase path.

Returns: VOID

Opens an existing database at the specified location.

Example:

```
res = db.open( file )
```

Example:

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

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

close()

Returns: VOID

Closes the database.

Example:

```
db.open()
        ....
db.close()
```

throwOut()

Returns: VOID

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

Example:

```
db.close()
        ....
db.throwOut()
```

flush()

Returns: VOID

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

Example:

```
db.flush()
```

---

clone(
string inNewDbPath,
integer inLoadRecords = true,
integer inDoLog = false)

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

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

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

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

Example:

```
db.clone( newDbLocation )
```

---

clone(
VDatabase inNewDatabase,
inLoadRecords as Boolean = true,
integer inDoLog = false)

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

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

Example:

```
dbCloned.create( newDbLocation, #kDscDatBlb_Ind, 8 * 1024 )
db.clone( dbCloned )
```

Structure Methods

createTable(
    string inName,
    symbol inTableKind = #kTblPermanent,
    symbol inStorageType = #kStorage_Default)

Parameter	Description
inName	The Name of a new Table.
inTableKind	The kind of Table. See Tablekind types.
inStorageType	Storage type for this database. See StorageType types.

Returns: VTable

Creates a new empty Table in the database.

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

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

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

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

Example:

```
dim tbl as VTable

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

dropTable(object inTable)

Parameter	Description
inTable	The Table, which must be deleted.

Returns: VOID

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

Example:

```
db.dropTable( tbl )
```

```
createForeignKeyLink(
        string inName,
        object inKeyField,
        object inPtrField,
        symbol inOnDelete = #kOnDelete_SetNull,
        symbol inOnUpdate = #kOnUpdate_Cascade,
        integer inTemporary = FALSE )
```

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

Returns: VLink

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

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

Example:

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

```
createBinaryLink(
        string inName,
        object inLeftTable,
        object inRightTable,
        symbol inLeftPower    = #kOne,
        symbol inRightPower   = #kMany,
        symbol inOnDelete     = #kOnDelete_SetNull,
        symbol inStorageType = #kStorage_Default,
        integer inTemporary   = FALSE )
```

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

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

To specify a link you need to define the following:

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

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

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

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

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

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

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

Example:

```
linkPersonPhone = db.createBinaryLink(
        "PersonPhone", tblPerson, tblPhone,
        #kMany, #kMany )
```

dropLink(object inLink)

Parameter	**Description**
inLink	The Link, which must be deleted.

Returns: VOID

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

Example:

```
db.dropLink( lnk )
```

Table Methods

table(any inIndexOrName)

Parameter	Description
inIndexOrName	The name or the index of a Table.

Returns: VTable

Returns the reference to the table by its name or the index.

Note: The parameter inIndexOrName is case insensitive.

Example:

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

Link Methods

link(any inIndexOrName)

Parameter	Description
inIndexOrName	The name or the index of a Table.

Returns: VLink

Returns the reference to the link by its name or the index.

Note: The parameter Name is case insensitive.

Example:

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

SQL Methods

VDatabase Xtra allow you execute SQL commands. The syntax of supported SQL is described in the ValentinaSQL document.

VDatabase Xtra offer you 3 methods to do SQL commands:
1) sqlExecute() - executes all commands except SELECT.
2) sqlSelect() - executes SELECT command and return VCursor Xtra with result.
3) sqlQuery() - is combination of above 2. It executes any SQL command and return result as property list.

The sqlQuery() command is easy for understanding and use. Most Lingo developers will prefer to use it. Developers which want more flexibility and performance may prefer to use SqlSelect + VCursor Xtra.

```
sqlExecute(
        string inQuery,
        list inBinds() = [] )
```

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

Returns: integer

You can use this function to execute any SQL command except SELECT command.

Returns the number of affected rows. Commands that change the database schema return zero always, because they do not affect records.

For commands that have an EXPR (expression) clause in the syntax, you can define an array of binded parameters. This is an array of strings for V4MD.

See for details the Valentina SQL Reference section in the Valentina Wiki
http://valentina-db.com/dokuwiki/

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

Example:

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

Example:

```
recCount = db.sqlExecute(
        "UPDATE person SET name = :1 WHERE name = :2",
        ['john', 'jehn']  )
```

```
sqlSelect(
        string inQuery,
        symbol inCursorLocation = #kClientSide,
        symbol inLockType = #kReadOnly,
        symbol inCursorDirection = #kRandom,
        list inBinds() = [] )
```

Parameter	**Description**
inQuery	The SQL string of a query.
inCursorLocation	The location of cusror. See CursorLocation types.
inLockType	The lock type for records of a cursor. See LockType types.
inCursorDirection	The direction of a cursor. See CursorDirection types.
inBinds()	The list of binded parameters.

Returns: VCursor

sqlSelect() method gets an SQL query string, resolves it and returns the result table as a VCursor Xtra. Now you can use methods of VCursor Xtra to work with records of result.

Note: You should destroy VCursor as soon as possible when you do not need it any more. For this use code: cursor = VOID

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

You can set the following parameters with these values:

inCursorLocation:	#kClientSide, #kServerSide, #kServerSideBulk
inLockType:	#kNoLocks , #kReadOnly, #kReadWrite
inCursorDirection:	#kForwardOnly, #kRandom

On default these parameters have the following values:
#kClientSide, #kReadOnly, #kForwardOnly

For the SELECT command you can define array of binded parameters. This is a list of strings for V4MD.

See for details the Valentina SQL Reference section in the Valentina Wiki
http://valentina-db.com/dokuwiki/

Example:

```
curs = db.sqlSelect( "SELECT * FROM T " )
```

Example:

```
curs = db.sqlSelect( "SELECT * FROM T ",
                #kServerSide, #kReadWrite, #kRandom )
```

Example:

```
curs = db.sqlSelect( "SELECT * FROM T WHERE f1 = :1, f2 > :2",
                #kServerSide, #kReadWrite, #kRandom,
                [ "john", "25" ] )
```

```
sqlQuery(
        string inQuery,
        symbol inLockType = #kReadOnly,
        list inBinds() = [] )
```

Parameter	Description
inQuery	The SQL string of a query.
inLockType	The lock type for records of a cursor. See LockType types.
inBinds()	The list of binded parameters.

Returns: property list.

sqlQuery() method gets an SQL query string, resolves it and returns the result as property list.

sqlQuery() can execute all and any SQL commands. So returned property list depend on command.

If you execute any command except SELECT, then property list looks as:

```
[ #errNumber: 0, #errNumberHex: 0, #errString: void, #rowsChanged: 1 ]
```

If you execute command SELECT, then property list also contains found records:

```
[ #errNumber: 0, #errNumberHex: 0, #errString: void, #rowsChanged: 0,
  #columns: ["fname", "lname"],
  #rows: [ ["Jon", "Jonson"], ["Peter", "Petrov"], ["Jak", "Davis"] ]
]
```

To access found records in property list you can use syntax as next:

```
put res.rows.count      -- to output the number of records
put res.rows[1]         -- to output the first record.
put res.rows[1][2]      -- to output the second field/column of the first record.
```

The optional parameter inLockType allow you to control the behavior of record locks during execution of SQL command. On default it is #kReadOnly. You may wish sometimes to use #kNoLocks. Note, for sqlQuery() there is no sense to use #kReadWrite.

The inBinds parameter allows you to define an array of binded parameters. This is a list of strings for V4MD.

See for details the Valentina SQL Reference section in the Valentina Wiki
http://valentina-db.com/dokuwiki/

Example:

```
res = db.sqlQuery( "CREATE TABLE ( fname string(40), lname string(40))" )
-- [ #errNumber: 0, #errNumberHex: 0, #errString: void, #rowsChanged: 1 ]
```

Example:

```
res = db.sqlQuery( "INSERT INTO T1(fname, lname) VALUES ('Jon', 'Jonson')" )
-- [ #errNumber: 0, #errNumberHex: 0, #errString: void, #rowsChanged: 1 ]
```

Example:

```
res = db.sqlQuery( "SELECT * FROM T1)" )
-- [ #errNumber: 0, #errNumberHex: 0, #errString: void, #rowsChanged: 0,
     #columns: ["fname", "lname"], #rows: [ ["Jon", "Jonson"] ] ]
```

Example:

```
res = db.sqlQuery( "SELECT * FROM T1 WHERE fname = :1)", #kReadOnly, ["Jon"] )
-- [ #errNumber: 0, #errNumberHex: 0, #errString: void, #rowsChanged: 0,
     #columns: ["fname", "lname"], #rows: [ ["Jon", "Jonson"] ] ]
```

IndexStyle Methods

createIndexStyle(string inName)

Parameter	Description
inName	The name of an index style.

Returns: VindexStyle

Creates a new Index Style in the database.

Example:

```
IndexStyle1 = db.createIndexStyle( "myStyle" )
```

dropIndexStyle(object inStyle)

Parameter	Description
inStyle	The index style to be deleted.

Returns: VOID

Deletes the specified index style from the database.

Example:

```
db.dropIndexStyle( IndexStyle1 )
```

IndexStyle(string inName)

Parameter	Description
inName	The Name of a IndexStyle.

Returns: VIndexStyle

Returns an IndexStyle by name.

Note: The parameter inName is case insensitive.

Example:

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

Collation Methods

getCollationAttribute(symbol inColAttribute)

Parameter: **Description:**
inColAttribute A collation attribute (see ColAttribute types)

Returns: symbol (see ColAttributeValue types)

Returns the value of the specified collation attribute of this dabase.

Example:

```
v = db.getCollationAttribute( #kStrength )
```

setCollationAttribute(symbol inColAttribute, symbol inValue)

Parameter: **Description:**
inColAttribute A collation attribute to be changed (see ColAttribute types)
inValue A new value for the dabase (see ColAttributeValue types)

Set new value for the specified collation attribute of this database.

Example:

```
db.setCollationAttribute( #kStrength, #kPrimary )
```

Connection Methods

getConnectionVariable(integer inConnVariableName)

Parameter:	Description:
inConnVariableName	The name of connection variable.

Returns: integer

Returns the value of the specified collation attribute of this dabase.

Example:

```
v = db.getConnectionVariable( #kStrength )
```

setConnectionVariable(integer inConnVariableName, integer inConnVariableValue)

Parameter:	Description:
inConnVariableName	The name of connection variable.
inConnVariableValue	The value of connection variable.

Returns: VOID

Set new value for the specified collation attribute of this database.

Example:

```
db.setConnectionVariable( #kStrength, #kPrimary )
```

Encryption Methods

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

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

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

```
encrypt(
        string inKey
        symbol inDataKind = #kRecordsOnly )
```

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

Allows you to encrypt the database.

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

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

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

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

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

Note:

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

Example:

```
db.open()
db.encrypt ( "key12345" )
```

Example:

```
db.open()
db.encrypt ( "key12345", #kStructureOnly )
```

```
decrypt(
        string inKey
        symbol inDataKind = #kRecordsOnly )
```

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

Allows to decrypt the database.

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

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

Example:

```
db.open()
db.decrypt ( "key12345" )
```

Example:

```
db.open()
db.decrypt ( "key12345", #kStructureOnly )
```

```
changeEncryptionKey(
        string inOldKey
        string inNewKey
        symbol inDataKind = #kRecordsOnly )
```

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

Allows you to change the encryption key for the database.

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

Example:

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

Example:

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

requiresEncryptionKey() as boolean

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

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

Example:

```
res =  db.requiresEncryptionKey()
```

useEncryptionKey(
    string inKey
    symbol inDataKind = #kRecordsOnly)

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

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

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

Example:

```
db.useEncryptionKey( "key12345" )
db.open()
```

Example:

```
db.useEncryptionKey( "key12345", #kStructureOnly )
db.open()
```

Dump Methods

```
dump(
        string inDumpFile,
        symbol inDumpType,
        symbol inDumpData = #kStructureAndRecords,
        int    inFormatDump = false,
        string inEncoding = "UTF-16" )
```

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

Returns: VOID

Dumps the database to the file in XML or SQL format.
Tip: You can use this file to recreate a database into a different location.

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

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

Example:

```
db.dump( fiXML, #kXML )
```

```
loadDump(
        string inDumpFile,
        string inNewDb,
        symbol inDumpType,
        string inEncoding = "UTF-16" )
```

Parameter	Description
inDumpFile	The location of dump file.
inDumpType	Type of a dump.
inEncoding	Encoding of dump file.

Returns: VOID

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

Example:

```
db.loadDump( fiXML,fiNewDb, #kXML )
```

VTable Xtra

Properties

VDatabase	database (r/o)
integer	fieldCount (r/o)
integer	ID (r/o)
string	ioEncoding
string	name
boolean	isEncrypted (r/o)
integer	linkCount (r/o)
string	locale
integer	physicalRecordCount (r/o)
integer	recID
integer	recordCount (r/o)
string	storageEncoding

Field Methods

field(any inNameOrIndex)

Link Methods

link(any inNameOrIndex)

Record Methods

setBlank((symbol inAccess = #forUpdate)

addRecord([list])
updateRecord([list])
updateAllRecords(VSet inSet)
deleteRecord()
deleteAllRecords(VSet inSet)

getRecord(integer inRecIndex = 0)

Cach Methods

flush()

Navigation Methods

```
firstRecord()
lastRecord()
nextRecord()
prevRecord()
recordExists( integer inRecID )
```

Structure Methods

```
createBooleanField    ( string inName, list inFlags = [], string inMethod = "" )
createByteField       ( string inName, list inFlags = [], string inMethod = "" )
createShortField      ( string inName, list inFlags = [], string inMethod = "" )
createUShortField     ( string inName, list inFlags = [], string inMethod = "" )
createMediumField     ( string inName, list inFlags = [], string inMethod = "" )
createUMediumField    ( string inName, list inFlags = [], string inMethod = "" )
createLongField       ( string inName, list inFlags = [], string inMethod = "" )
createULongField      ( string inName, list inFlags = [], string inMethod = "" )
createLLongField      ( string inName, list inFlags = [], string inMethod = "" )
createULLongField     ( string inName, list inFlags = [], string inMethod = "" )
createFloatField      ( string inName, list inFlags = [], string inMethod = "" )
createDoubleField     ( string inName, list inFlags = [], string inMethod = "" )

createDateField       ( string inName, list inFlags = [], string inMethod = "" )
createTimeField       ( string inName, list inFlags = [], string inMethod = "" )
createDateTimeField   ( string inName, list inFlags = [], string inMethod = "" )

createStringField     ( string inName, list inFlags = [], string inMethod = "" )
createVarCharField    ( string inName, list inFlags = [], string inMethod = "" )

createFixedBinaryField( string inName, integer inMaxLength )
createVarBinaryField  ( string inName, integer inMaxLength )

createBLOBField       ( string inName, integer inSegmentSize )

createTextField       ( string inName, integer inSegmentSize, list inFlags = [], string inMethod = "" )

createPictureField    ( string inName, integer inSegmentSize )

createObjectPtrField  ( string inName,
                        object inTarget,
                        symbol inOnDeletion = #kOnDelete_Cascade,
                        list inFlags = [] )

dropField( object inFld )

changeType(
        object inFld,
        symbol inNewType,
        integer inParam1 )
```

Collation Methods

getCollationAttribute(symbol inColAttribute)
setCollationAttribute(symbol inColAttribute, symbol inValue)

Encryption Methods

encrypt(string inKey)
decrypt(string inKey)
changeEncryption(string inOldKey, string inNewKey)

requiresEncryptionKey()
useEncryptionKey(string inKey, symbol)

Dump Methods

```
dump(
        string inDumpFile,
        symbol inDumpType,
        symbol inDumpData,
        int inFormatDump )

loadDump(
        string inDumpFile,
        symbol inDumpType )
```

Selection Methods

selectAllRecords()
selectNoneRecords()

```
sort(
        object inSet,
        object inField,
        integer inAcending )
```

sortMultiple(object inSortList)

Class description

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

Properties

VDatabase database (r/o)

Returns the parent database of this table.

Example:

```
db = table.database
```

integer fieldCount (r/o)

Returns the number of custom fields in the table.

Example:

```
fldCount = table.fieldCount
```

nteger ID (r/o)

Returns the unique identifer of the table.

Example:

```
id = table.ID
```

string name

Returns the name of the table as a string.

Example:

```
Table.name = "person"
```

boolean isEncrypted (r/o)

Returns TRUE if the database is encrypted.

Example:

```
encrypted = table.isEncrypted
```

integer linkCount (r/o)

Returns the number of links in the table.

Example:

```
linkCount = table.linkCount
```

string locale

Specifies for this table the locale name.

Example:

```
locale = table.locale

table.locale = "en_US"
table.locale = "jp_JP"
```

integer physicalRecordCount (r/o)

Returns the number of physical records in the table.

Example:

```
physRecCount = table.physicalRecordCount
```

integer recID

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

Also you can use this property to change the current record of the Table. In case you try move to a non- existant record the current record will not be changed.

Example:

```
recID = Table.recID

Table.recID = RecID                 -- move to specific record
Table.recID = 54                    -- move to specific record

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

integer recordCount (r/o)

Returns the record count for in the table.

Example:

```
rcdCount = table.recordCount
```

string storageEncoding

Specifies for this table the string encoding stored on disk.

Example:

```
Encoding = table.storageEncoding

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

Field Methods

field(any inNameOrIndex)

Parameter	Description
inNameOrIndex	The name or the index of the field.

Returns: VField

Returns the reference of table's field.

Example:

```
fld = tbl.field( "name" )
```

Example:

```
count = tbl.fieldCount
repeat with i > 1 to count
f = tbl.field( i )
-- do come Work
end repeat
```

Links Methods

link(any inNameOrIndex)

Parameter	Description
inNameOrIndex	The index or name of a link.

Returns: VLink

Returns a link of this table, specifield by name or index.

Example:

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

Record Methods

setBlank(symbol inAccess = #forUpdate)

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

Returns: VOID

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

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

Example:

```
Table.setBlank()
```

addRecord([list])

Returns: integer

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

This method can accept optionally a list of values you want to use in the new record. You can use both simple list and property list. For a simple list, it is important that the order of values in the list corresponds to the order of fields in the cursor. Usage of list can simplify syntax. Note, that no need in this case call SetBlank() explicitly.

Returns the RecID of the new record.

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

Example:

```
thePerson.setBlank()
        thePerson.firstName.Value = “John”
        thePerson.lastName.Value = “Roberts”
newRecID = thePerson.addRecord()
```

Example:

```
newRecID = thePerson.addRecord( [#firstName:John, #lastName:Roberts] )
```

deleteRecord()

Returns: VOID

Deletes the current record of a Table.

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

Example:

```
Table.deleteRecord()
```

deleteAllRecords(VSet inSet)

Parameter	Description
inSet	The selection of records.

Returns: VOID

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

Example:

```
Table.deleteAllRecords()
```

updateRecord([list])

Returns: VOID

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

This method can accept optionally a list of values you want to use in the new record. You can use both simple list and property list. For a simple list, it is important that the order of values in the list corresponds to the order of fields in the cursor. Usage of list can simplify syntax.

Example:

```
thePerson.recID = SomeRecID
thePerson.field("FirstName").Value = “Brian”
thePerson.field("LastName").Value = “Blood”
thePerson.updateRecord()
```

Example:

```
thePerson.updateRecord( [#FirstName:Brian, #LastName:Blood] )
```

updateAllRecords(VSet inSet)

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

Returns: VOID

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

Example:

```
Table.updateAllRecords()
```

getRecord(integer inRecIndex = 0)

Parameter	**Description**
inRecIndex	The index of the record to be returned. Default - current record.

Returns: list.

Returns a list of ALL table fields for a specified record.

Using of this function will work a little faster than calling many times GetField.

Example:

```
Fields =Table.getRecord( )    -- returns a list of values of the current record.
Fields =Table.getRecord( i )  -- returns a list of values of the i-th record.
```

Cache Methods

flush()

Returns: VOID

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

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

Example:

```
Table.flush()
```

Navigation Methods

firstRecord()

Returns: boolean

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

Example:

```
Table.firstRecord()
```

lastRecord()

Returns: boolean

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

Example:

```
Table.lastRecord()
```

nextRecord()

Returns: boolean

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

Example:

```
res = Table.nextRecord()
```

prevRecord()

Returns: boolean

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

Example:

```
Table.prevRecord()
```

recordExists(integer inRecID)

Parameter	Description
inRecID	The recID of record.

Returns: boolean

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

Example:

```
res = Table.recordExists( recID )
```

Structure Methods

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

At its simplest, this lets you change the size of records in a Table. If a Table already has records, then your disk files must be translated. Valentina performs these operations with the help of temporary files. If the host computer crashes for any reason, the database itself has protection from corruption.

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

Methods to create numeric fields.

```
createShortField(
        string inName,
        list inFlags = [],
        string inMethod = "" )
```

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

Returns: VShort

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

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

Example

```
fldAge = tblPerson.createShortField(
                "age",   [ #fNullable, #fIndexed] )
```

Methods to create string/varchar fields.

```
createStringField(
        string inName,
        list inFlags = [],
        string inMethod = "" )

createVarCharField(
        string inName,
        list inFlags = [],
        string inMethod = "" )
```

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

Returns: VString,VVarChar

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

Example

```
fldAge = tblPerson.createStringField( "name", 40, [ #fNullable, #fIndexed] )
```

Methods to create fixed/var binary fields.

```
createFixedBinaryField(
        string inName,
        integer inMaxLen )

createVarBinaryField(
        string inName,
        integer inMaxLen )
```

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

Returns: VFixedBinary, VVarBinary

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

Example

```
fldAge = tblPerson.createFixedBinaryField(
                "nameStile", 40,  [ #fNullable, #fIndexed] )
```

Method to create BLOB fields.

```
createBLOBField(
        string inName,
        integer inSegmentSize )
```

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

Returns: VBLOB

Create a BLOB (Binary Large OBject) field.

•You need to specify the segment size in bytes.

Example

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

Method to create TEXT fields.

```
createTextField(
        string inName,
        integer inSegmentSize,
        list inFlags = [],
        string inMethod = "" )
```

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

Returns: VText

Create a Text field.

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

Example

```
        fldAge = tblPerson.createTextField(
                "notes", 256,  [ #fNullable, #fIndexed] )
```

Method to create Picture fields.

```
CreatePictureField(
        string inName,
        integer inSegmentSize )
```

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

Returns: VPicture

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

Example

```
fldAge = tblPerson.createPictureField(
                "foto", 256, [ #fNullable, #fIndexed] )
```

Method to create ObjectPtr fields.

```
CreateObjectPtrField(
        string inName,
        object inTarget,
        symbol inOnDeletion = #kOnDelete_Cascade,
        list inFlags = [] )
```

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

Returns: VObjectPtr

Create an ObjectPtr field.

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

Example

```
fldAge = tblPerson.createObjectPtrField(
                "ParentPtr",  [ #fNullable, #fIndexed] )
```

dropField(object inFld)

Parameter	Description
inFld	The Field, which must be deleted.

Returns: VOID

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

Example:

```
Table.dropField( fld )
```

changeType(
object inFld,
symbol inNewType,
integer inParam1)

Parameter	Description
inFld	The field whose type should be changed.
inNewType	New type for a field. See FieldType types.
inParam1	The additional parameter (see below).

Returns: VField

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

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

Example:

```
fld = Table.changeType( fld, #kTypeString,40 )
```

Collation Methods

getCollationAttribute(symbol inColAttribute)

Parameter:	Description:
inColAtribute	A collation attribute (see ColAttribute types)

Returns: symbol (see ColAttributeValue types)

Returns the value of the specified collation attribute of this table.

Example:

```
v = tbl.getCollationAttribute( #kStrength )
```

setCollationAttribute(symbol inColAttribute, symbol inValue)

Parameter:	Description:
inColAtribute	A collation attribute to be changed (see ColAttribute types)
inValue	A new value for the table (see ColAttributeValue types)

Set new value for the specified collation attribute of this table.

Example:

```
tbl.setCollationAttribute( #kStrength, #kPrimary )
```

VTable Encryption Methods

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

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

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

---

encrypt(string inKey)

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

Allows you to encrypt the Table.

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

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

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

Example:

```
tbl.encrypt( "key12345" )
```

---

decrypt(string inKey)

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

Allows to decrypt the Table.

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

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

Example:

```
tbl.decrypt( "key12345" )
```

```
changeEncryptionKey(
        string inOldKey,
        string inNewKey )
```

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

Allows you to change the encryption key fot the Table.

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

Example:

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

requiresEncryptionKey() as Boolean

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

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

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

Example:

```
res = tbl.requiresEncryptionKey()
```

useEncryptionKey(string inKey)

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

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

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

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

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

Example:

```
db.useEncryptionKey( "key12345" )
db.open()

tbl.useEncryptionKey( "key12345" )
```

Dump Methods

dump(
string inDumpFile,
symbol inDumpType,
symbol inDumpData,
int inFormatDump)

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

Returns: VOID

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

Example:

```
tbl.dump( pathToFile, #kXML )
```

loadDump(
string inDumpFile,
symbol inDumpType)

Parameter	Description
inDumpFile	The location of the dump file.
inDumpType	The type of dump. See DumpType types.

Returns: VOID

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

Example:

```
tbl.loadDump(pathToFile, #kXML )
```

Selection Methods

selectAllRecords()

Returns: VBitSet

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

Example:

```
allRecs = Table.selectAllRecords()
```

selectNoneRecords()

Returns: VBitSet

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

Example:

```
NoneRecs = Table.selectNoneRecords()
```

sort (
object inSet,
object inField,
integer inAscending)

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

Returns: VArraySet

Executes sorting of a table selection inSet by the field Field. The parameter inAscending specifies the order of sorting.

Returns a new sorted selection as an VArraySet.

Example:

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

sortMultiple(object inSortList)

P**arameter**	**Description**
inSortList	List of sort conditions.

Returns: VArraySet

Sorts table using the given list of sort criterias.

Example:

```
SortedSet = table.sortMultiple
```

VField Xtra

Propeties

integer ID (r/o)
VIndexStyle indexStyle
string ioEncoding

boolean isIndexed
boolean isUnique
boolean isNullable
boolean isMethod (r/o)
boolean isEncrypted (r/o)

string locale
string methodText
string name

string storageEncoding

symbol type (r/o)
string typeString (r/o)

VTable table (r/o)

// value properties
variant defaultValue
any isNull
any value

// String Varchar fields
boolean indexByWords
integer maxLength

// BLOB properties
integer dataSize
boolean isCompressed
integer segmentSize (r/o)

// Picture properties
symbol pictureType (r/o)

Value Methods

setBlank()

getString()
setString(string inValue)

Search Methods

valueExists(any inValue, integer inSelection)
findValue(any inValue, object inSelection = VOID)
findValueAsArraySet(any inValue, object inSelection = VOID)
findRange(integer inLeftInclude, any inLeftValue, integer inRightInclude, any inRightValue, object inSelection = VOID)
findRangeAsArraySet(integer inLeftInclude, any inLeftValue, integer inRightInclude, any inRightValue, object inSelection = VOID)
findSingle(any inValue, object inSelection = VOID)
findNulls(object inSelection = VOID)
findNotNulls(object inSelection = VOID)

findStartsWith(string inCriteria, object inSelection = VOID)
findContains(string inCriteria, object inSelection = VOID)
findEndsWith(string inCriteria, object inSelection = VOID)
findRegEx (string inCriteria, object inSelection = VOID)
findLike(string inCriteria, object inSelection = VOID)

Collation Methods

getCollationAttribute(symbol inColAttribute)
setCollationAttribute(symbol inColAttribute, symbol inValue)

Encryption Methods

encrypt(string inKey)
decrypt(string inKey)
changeEncryption(string inOldKey, string inNewKey)

requiresEncryptionKey()
useEncryptionKey(string inKey, symbol)

BLOB Interface

```
deleteData()

getMedia( any inCastMember )
setMedia( any inCastMember )

fromFile( string inFilePath )
toFile ( string inFilePath )
```

Picture Interface

```
getPicture(
        any inCastMember = VOID )

setPicture(
        any inCastMemberOrPicture,
        symbol inPicFormat = #kJPG,
        integer inCompression = 50 )

getImage()

setImage(
        image  inImage,
        symbol inPicFormat = #kJPG,
        integer inCompression = 50 )
```

Properties

integer ID (r/o)

Return the unique identifier of the table.

Example:

```
id = table.ID
```

VIndexStyle indexStyle

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

Example:

```
fld.indexStyle = style1

currStyle = fld.indexStyle
```

boolean isIndexed

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

Example:

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

boolean isUnique

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

Example:

```
fld.isUnique = TRUE
if( fld.isUnique )
```

boolean isNullable

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

Example:

```
fld.isNullable = TRUE
if( fld.isNullable )
```

boolean isMethod (r/o)

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

Example:

```
if( fld.isMethod )
```

boolean isEncrypted (r/o)

Returns TRUE if the database is encrypted.

Example:

```
encrypted = table.isEncrypted
```

string locale

Specifies for this table the locale name.

Example:

```
locale = table.locale

table.locale = "en_US"
table.locale = "jp_JP"
```

string methodText

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

Example:

```
method = fld.methodText

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

string name

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

Example:

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

string storageEncoding

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

Example:

```
Encoding = fld.storageEncoding

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

symbol type (r/o)

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

Each field has several flags, which define its behavior:

Example:

```
case fld.type
```

See also: VTable.ChangeType

string typeString (r/o)

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

Example:

```
strType = fld.typeString
```

VTable table (r/o)

Returns the Table of this field.

Example:

```
t = fld.table
```

variant defaultValue

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

Example:

```
v = fld.defaultValue
```

any isNull

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

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

Example:

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

any value

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

Example:

```
fld.value = 5
iv = fld.value
```

boolean indexByWords

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

Example:

```
fldString.indexByWords = TRUE
```

integer maxLength

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

Note: If you change the maximum length of the field, you change a size of the table records. This means that Valentina must rebuild the table, so this operation require some time.

Example:

```
len = fldString.maxLength
fldString.maxLength = 120
```

integer dataSize

Returns the size in bytes of BLOB value of the current record for this BLOB field. You need this size to preapre memory buffer for this BLOB value.

Example:

```
size = fldBLOB.dataSize()
```

boolean isCompressed

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

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

Example:

```
fldBlob.isCompressed =TRUE
```

integer segmentSize (r/o)

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

Example:

```
segment = fldBlob.segmentSize
```

symbol pictureType (r/o)

Returns the type of picture stored in the Picture field.

See PictType types: #DIB, #Pict, #JPG, #TIFF

Example:

```
pictType = fldPicture.pictureType
```

Value Methods

setBlank()

Returns: VOID

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

Example:

```
fld.setBlank()
```

getString()

Returns: string

Returns the value of the field as a string.

Example:

```
str = fld.getString()
```

setString(string inValue)

Parameter:	Description:
inValue	New value for the field.

Returns: VOID

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

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

Example:

```
str = "aaaaa"
...

fld.setString( str )
```

Search Methods

valueExists(
any inValue,
object inSelection)

Parameter:	Description:
inValue	The value to search.
inSelection	The selection of record for search.

Returns: boolean

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

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

Example:

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

findValue(
any inValue,
object inSelection = VOID)

Parameter:	**Description:**
inValue	The value to search.
inSelection	Selection of records.

Returns: VBitSet

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

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

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

Example:

```
s1 = fld1.findValue(5)
s2 = fld2.findValue( 7, s1 )
```

findValueAsArraySet(
any inValue,
object inSelection = VOID)

Parameter:	**Description:**
inValue	The value to search.
inSelection	Selection of records.

Returns: VArraySet

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

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

Example:

```
s1 = fld1.findValueAsArraySet(5)
s2 = fld2.findValueAsArraySet( 7, s1 )
```

```
findRange(
        integer inLeftInclude,
        any inLeftValue,
        integer inRightInclude,
        any inRightValue,
        object inSelection = VOID )
```

Parameter:	**Description:**
inLeftInclude	TRUE if the left value of the range must be included.
inLeftValue	The left value of the range.
inRightInclude	The right value of the range.
inRightValue	TRUE if the right value of the range must be included.
inSelection	Selection of records.

Returns: VBitSet

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

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

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

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

Example:

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

```
findRangeAsArraySet(
        boolean inLeftInclude,
        any inLeftValue,
        boolean inRightInclude,
        any inRightValue,
        object inSelection = VOID )
```

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

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

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

Example:

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

```
findSingle(
        any inValue,
        object inSelection = VOID )
```

Parameter:	**Description:**
inValue	The value to search.
inSelection	Selection of records.

Returns: integer

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

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

Example:

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

findNulls(object inSelection = VOID)

Parameter:	**Description:**
inSelection	Selection of records.

Returns: VBitSet

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

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

Example:

```
bset = fld.findNulls()
```

findNotNulls(object inSelection = VOID)

Parameter:	**Description:**
inSelection	Selection of records.

Returns: VBitSet

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

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

Example:

```
bset = fld.findNotNulls()
```

String Search Methods

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

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

---

findStartsWith(
    string inCriteria,
    object inSelection = VOID)

Parameter:	**Description:**
inCriteria	The search string.
inSelection	Selection of records.

Returns: VBitSet

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

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

Example:

```
bset = fld.findStartsWith( "Jo" )
```

---

findContains(
    string inCriteria,
    object inSelection = VOID)

Parameter:	**Description:**
inCriteria	The search string.
inSelection	Selection of records.

Returns: VBitSet

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

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

Example:

```
bset = fld.findContains( "Jo" )
```

findEndsWith(
string inCriteria,
object inSelection = VOID)

Parameter:	**Description:**
inCriteria	The search string.
inSelection	Selection of records.

Returns: VBitSet

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

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

Example:

```
bset = fld.findEndsWith( "hn" )
```

findRegEx (
string inCriteria,
object inSelection = VOID)

Parameter:	**Description:**
inCriteria	The search string.
inSelection	Selection of records.

Returns: VBitSet

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

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

Example:

```
bset = fld.findRegEx( "Pe?" )
```

```
findLike(
        string inCriteria,
        object inSelection = VOID )
```

Parameter:	**Description:**
inCriteria	The search string.
inSelection	Selection of records.

Returns: VBitSet

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

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

Example:

```
        bset = fld.findLike( "%eter" )
```

Collation Methods

getCollationAttribute(symbol inColAttribute)

Parameter:	Description:
inColAtribute	A collation attribute (see ColAttribute types)

Returns: symbol (see ColAttributeValue types)

Returns the value of the specified collation attribute of this field.

Example:

```
v = fld.getCollationAttribute( #kStrength )
```

setCollationAttribute(symbol inColAttribute, symbol inValue)

Parameter:	Description:
inColAtribute	A collation attribute to be changed (see ColAttribute types)
inValue	A new value for the field (see ColAttributeValue types)

Set new value for the specified collation attribute of this field.

Example:

```
fld.setCollationAttribute( #kStrength, #kPrimary )
```

VField Encryption Methods

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

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

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

---

encrypt(string inKey)

Parameter:	Description:
inKey	The encryption key.

Allows you to encrypt the separate Field in the table.

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

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

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

Example:

```
fld.encrypt( "key12345" )
```

---

decrypt(string inKey)

Parameter:	Description:
inKey	The encryption key.

Allows to decrypt the Field in the table.

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

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

Example:

```
fld.decrypt( "key12345" )
```

```
changeEncryptionKey(
        string inOldKey,
        string inNewKey )
```

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

Allows you to change the encryption key for the Field.

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

Example:

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

requiresEncryptionKey() as Boolean

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

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

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

Example:

```
res =fld.requiresEncryptionKey()
```

useEncryptionKey(string inKey)

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

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

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

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

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

Example:

```
db.useEncryptionKey( "key12345" )
db.open()

fld.useEncryptionKey( "key12345" )
```

BLOB Interface

deleteData()

Returns: VOID

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

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

Example:

```
fldBLOB.deleteData()
curs.updateRecord()
```

getMedia(any inCastMember)

Parameter:	**Description:**
inCastMember	A cast member in which to put media.

Returns: VOID

Copies the content of a BLOB field to the specified Director cast member.

Example:

```
fldBLOB.getMedia( theCursor, "Photo", member "Photo"  )
```

setMedia(any inCastMember)

Parameter:	**Description:**
inCastMember	A cast member in which to put media.

Returns: VOID

Copies the contents of the specified Director cast member to a BLOB field.
This handler can store in the BLOB field any media type except a Movie.

Director has different formats of Media in versions 6 and 7+, so if you use SetMedia then your database will not be compatiable between Director 6 and 7. If you need a compatible database, then use the SetPicture handler.

Example:

```
fldBLOB.setMedia ( member "Photo" of CastLib "MyDB" )
```

fromFile(string inFilePath)

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

Returns: VOID

Read the whole file into the BLOB field.

Example:

```
fldBLOB.fromFile( location )
tbl.addRecord()
```

toFile (string inFilePath)

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

Returns: VOID

Upload the value of BLOB field into the specified disk location, i.e. it creates a new disk file.

Example:

```
fldBLOB.toFile( location )
```

Picture Interface

---

getPicture(any inCastMember = VOID)

Parameter:	**Description:**
inCastMember	The cast member into which put the picture.

Returns: VOID OR a picture

Reads a picture from the database and put it into the cast member if it is specified. Otherwise returns the picture as Lingo variable (for Director ver 7.0 and newer).

Example:

```
fldPicture.getPicture( member "Photo"  )      -- returns into the cast member
```

Example:

```
thePict = fldPicture.getPicture()             -- returns into the Lingo variable
```

---

setPicture(
any inCastMemberOrPicture,
symbol inPicFormat = #kJPG,
integer inCompression = 50)

Parameter:	**Description:**
inCastMemberOrPicture	The cast member from which we get picture or Lingo picture variable.
inPicFormat	The picture format. kJPG on default. Can be kTIFF also.
inCompression	For JPG specifies the compression quality [1-100]. For TIFF specifies method of compression.

Returns: VOID

Stores Picture into the database VPicture field using the specified format.

You can specify the cast member that contains a picture or pass the Lingo picture variable (for Director ver 7.0 and better).

The parameter inCompression can be in the range 0..100 and specify quality of a jpeg compression. The bigger value the better quality.

For TIFF parameter inCompression specifies method of compression. Valid values are:

TIFFTAG_COMPRESSION	259
COMPRESSION_NONE	1
COMPRESSION_CCITTRLE	2
COMPRESSION_CCITTFAX3	3
COMPRESSION_CCITT_T4	3
COMPRESSION_CCITTFAX4	4
COMPRESSION_CCITT_T6	4
COMPRESSION_LZW	5
COMPRESSION_OJPEG	6
COMPRESSION_JPEG	7
COMPRESSION_NEXT	32766
COMPRESSION_CCITTRLEW	32771
COMPRESSION_PACKBITS	32773
COMPRESSION_THUNDERSCAN	32809
COMPRESSION_IT8CTPAD	32895
COMPRESSION_IT8LW	32896
COMPRESSION_IT8MP	32897
COMPRESSION_IT8BL	32898
COMPRESSION_PIXARFILM	32908
COMPRESSION_PIXARLOG	32909
COMPRESSION_DEFLATE	32946
COMPRESSION_ADOBE_DEFLATE	8
COMPRESSION_DCS	32947
COMPRESSION_JBIG	34661
COMPRESSION_SGILOG	34676
COMPRESSION_SGILOG24	34677
COMPRESSION_JP2000	34712

Example:

```
-- using cast member:
fldPicture.setPicture( member "Photo", #kJPG, 40 )

-- using Lingo variable:
fldPicture.setPicture( thePict, #kJPG, 40 )

-- using cast member:
kKZW = 5
fldPicture.setPicture( member "Photo", #kTIFF, kLZW )
```

getImage()

Returns: image

Reads a picture from the database and returns it as Image into Lingo variable.

Example:

```
image = fldPicture.getImage()
```

setImage(
image inImage,
symbol inPicFormat = #kJPG,
integer inCompression = 50)

Parameter:	**Description:**
inPicFormat	The picture format. kJPG on default. Can be kTIFF also.
inCompression	For JPG specifies the compression quality [1-100]. For TIFF specifies method of compression.

Returns: VOID

Stores Picture into the database VPicture field using the specified format.

You specify a picture as Lingo Image variable.

The parameter inCompression can be in the range 0..100 and specify quality of a jpeg compression. The bigger value the better quality.

For TIFF parameter inCompression specifies method of compression. See above for valid values.

Example:

```
fldPicture.setImage( image, #kJPG, 40 )
```

VCursor Xtra

Properties

boolean BOF
boolean EOF

object database
integer fieldCount
boolean isReadOnly
integer position
integer recordCount

Field Methods

Field(any inNameOrIndex)

bindField(any inNameOrIndex)

Navigation Methods

nextRecord()
firstRecord()
prevRecord()
lastRecord()

Record Methods

setBlank()

addRecord([list])
updateRecord([list])
updateAllRecords([list])
deleteRecord()
deleteAllRecords()
dropRecord()

Batch Record Methods

```
getRecord( integer inRecIndex = 0 )
getRecords(
        integer inFrom = 1,
        integer inRecCount = -1 )

getRecordAsPropList( integer inRecIndex )
getRecordsAsPropList(
        integer inFrom = 1,
        integer inRecCount = -1 )

getColumn(
        any inFldNameorIndex,
        integer inFrom,
        integer inRecCount )

getRecordAsString(
        integer inRecIndex = 0,
        string inFldDelimiter = TAB,
        string inFldPrefix = "",
        string inFldSuffix = "",
        string inRecPrefix = "",
        string inRecSuffix = "")

getRecordsAsString(
        integer inFromRec = 0,
        integer inMaxRecords = 1,
        string inFldDelimiter = TAB,
        string inRecDelimiter = RETURN)
        string inFldPrefix = "",
        string inFldSuffix = "",
        string inRecPrefix = "",
        string inRecSuffix = "")
```

Import/export Methods

```
importText(
        string inFile,
        string inFieldDelimter = chr(09),
        string inLineDelimter = LE,
        string inEncoding = "UTF-16",
        integer inHasColumnHeader = FALSE )

exportText(
        string inFile,
        string inFieldDelimter = chr(09),
        string inLineDelimter = LE,
        string inEncoding = "UTF-16",
        integer inHasColumnHeader = FALSE )
```

Conversion Methods

```
toArraySet()
```

Description

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

A detailed description of SQL supported by Valentina can be found in the "Valentina SQL" section of Valentina WIKI

Since Valentina uses SQL, you can:
- SELECT fields which you need from one or more tables and define their order.
- SEARCH in one or more related tables by one or more MATCH-conditions joined by AND/OR operators.
- SORT resulting records by one or more fields.

As a result of a SQL query you get a Cursor – a sorted selection of records matching to the specified conditions.

A Cursor can be considered as a new temporary table with chosen fields and records. So, as you will see, VCursor Xtra also has groups of handles to work with fields and records.

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

When you finish working with cursor, you must destroy it to free memory.

Properties

boolean BOF

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

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

Example:

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

boolean EOF

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

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

Example:

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

object database

Returns the reference of the database of this cursor.

Example:

```
db = curs.database
```

integer fieldCount

Returns the number of fields of this cursor.

Example:

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

boolean isReadOnly

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

Example:

```
if( curs.readOnly )
```

integer position

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

The valid range of values is from 1 to the RecordCount.

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

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

Example:

```
for i = 1 to curs.recordCount
    curs.position = i
next
```

integer recordCount

Returns the number of records of cursor as an integer.

Example:

```
recCount = curs.recordCount // get local shortcut to aVOID of calling in loop
for i = 1 to fldcount
```

Field Methods

field(any inNameOrIndex)

Parameter	Description
inNameOrindex	The name or the index of the field.v

Returns: 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 the query.

Example:

```
cur = gDataBase.sqlSelect(
                “select * from person where name like ‘john’ ”)

recCount = curs.recordCount
for i = 1 to recCount
        curs.position = i
        lastName = curs.field( “last_name” ).getString()
next
```

bindField(any inNameOrIndex)

Parameter	Description
inNameOrIndex	The name or the index of a field.

Returns: VOID

Establishes a live link between a field and a cast member. Once bound, the member automatically displays and updates the content of its associated database field, for the current record of the cursor.

Fields of all types can be bound to Director cast members, including fields of type Media. If you bind a BLOB field then methods GetMedia/SetMedia will be used in background. If you bind Picture fields then handlers GetPicture/SetPicture will be used in background.

If you need to use handlers BLOB_ReadData/BLOB_WriteData then you must not use the binding for such field and call them explicitly.

Example:

```
SQLstring = text of member "SQLquery"
gCursor = gMyDataBase.sqlSelect( SQLstring )

gCursor.bindField(  "First Name", member "fFirstName" )
gCursor.bindField(  "Image", member "fImage" )
```

Navigation Methods

nextRecord()

Returns: boolean

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

Example:

```
if(  myCursor.firstRecord( ) )
        Do
                // work here
        Loop    Until myCursor.nextRecord( ) = FALSE
end if
```

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

Example:

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

prevRecord()

Returns: boolean

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

Example:

```
res = curs.prevRecord( )
```

firstRecord()

Returns: boolean

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

Example:

```
firstRecord ( theCursor )
```

lastRecord()

Returns: boolean

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

Example:

```
lastRecord ( theCursor )
```

Record Methods

setBlank()

Returns: VOID

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

Example:

```
curs.setBlank( )
        curs.Field( 1).Value = i
        curs.Field( 2).Value = i
res = curs.addRecord()
```

addRecord([list])

Returns: integer

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

This method can accept optionally a list of values you want to use in the new record. You can use both simple list and property list. For a simple list, it is important that the order of values in the list corresponds to the order of fields in the cursor. Usage of list can simplify syntax. Note, that no need in this case call SetBlank() explicitly.

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

Example:

```
curs.setBlank()
        curs.Field(1).Value = 1
        curs.Field(2).Value = 2
res = curs.addRecord()
```

Example:

```
-- simple list
res = curs.addRecord( [1, 2] )
```

Example:

```
-- property list
-- both lines give the same effect.
res = curs.addRecord( [#fld1:1, #fld2:2] )
res = curs.addRecord( [#fld2:2, #fld1:1] )
```

updateRecord([list])

Returns: VOID

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

This method can accept optionally a list of values you want to use in the new record. You can use both simple list and property list. For a simple list, it is important that the order of values in the list corresponds to the order of fields in the cursor. Usage of list can simplify syntax.

Returns FALSE if the record cannot be updated, e.g. cursor is readOnly.

Example:

```
curs.currentRecord = i
        curs.Field(1).Value = 101
        curs.Field(2).Value = 102
curs.updateRecord()
```

Example:

```
curs.updateRecord( [101, 102] )
```

Example:

```
curs.updateRecord( [#fld1:101, #fld2:102] )
```

updateAllRecords([list])

Returns: boolean

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

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

NOTE: This function is equivalent of SQL command:
"UPDATE TABLE T(f1,f2) VALUES(v1, v2) WHERE SomeCondition".
i.e. it assigns the same values to all records selected by the WHERE part.

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

Example:

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

res = curs.updateAllRecords()
```

deleteRecord()

Returns: VOID

Deletes the current recordof a cursor.

After deletion, the record which follows the record being deleted becomes the new current record. If there is no record following the deleted record, the preceding record becomes the new current record. If there is no record preceding the deleted record, the selection is then empty and the current record is not defined.

Example:

```
theCursor.deleteRecord()
```

deleteAllRecords()

Returns: VOID

Deletes all records of the cursor.
The Cursor must be constructed to accept updates of records. Otherwise this command will be ignored.

Example:

```
theCursor.deleteAllRecords()
```

dropRecord()

Returns: VOID

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

Example:

```
theCursor.dropRecord()
```

Batch Record Methods

getRecord(integer inRecIndex = 0)

Parameter	Description
inRecIndex	The index of the record to be returned. Default - current record.

Returns: list.

Returns the list of ALL cursor fields for a specified record.

Using of this function will work a little faster than calling many times GetField.

Example:

```
Fields =cursor.getRecord( )   -- returns list of values of the current record.
Fields =cursor.getRecord( i ) -- returns list of values of the i-th record.
```

getRecords(
integer inFrom = 1,
integer inRecCount = -1)

Parameter	Description
inFrom	The index of the start record.Default -- 1.
inRecCount	Maximal number of the records to return. Default -- all records.

Returns: list

Returns the list of lists of field values for several records of a cursor. You specify a start record to be returned and the maximal number of records.

Using of this function will work a little faster than calling many times GetField.

Example:

```
records = cursor.getRecords()          -- returns list of ALL records.
records = cursor.getRecords( 1, 10 )   -- returns list of 1-10 records.
```

getRecordAsPropList(integer inRecIndex)

Parameter	Description
inRecIndex	The index of the record to be returned. Default -- current.

Returns: a property list

Returns values of ALL cursor fields of the specified record as a property list.

Example:

```
Fields = cursor.getRecordAsPropList()      -- returns values of current record.
Fields = cursor.getRecordAsPropList( i )   -- returns values of the i-th record.
```

getRecordsAsPropList(
integer inFrom = 1,
integer inRecCount = -1)

Parameter	Description
inFrom	The index of the start record.Default -- 1.
inRecCount	Maximal number of the records to return. Default -- all records.

Returns: a list of property lists.

Returns the list of property lists of field values for several records of cursor. You specify a start record to be returned and the maximum number of records.

Example:

```
records = cursor.getRecordsAsString()          -- returns list of ALL records.
records = cursor.getRecordsAsString( 1, 10 )   -- returns list of 1-10 records.
```

getColumn(
any inFldNameorIndex,
integer inFrom,
integer inRecCount)

Parameter	Description
inFldNameOrIndex	The name or the index of the field to be used.
inFrom	The index of the start record.Default -- 1.
inRecCount	Maximal number of the records to return. Default -- all records.

Returns: list

Returns ALL records of a cursor for ONE cursor field as a list.

Example:

```
ListOfNames = cursor.getColumn( "Name" )
```

```
getRecordAsString(
        integer inRecIndex = 0,
        string inFldDelimiter = TAB,
        string inFldPrefix = "",
        string inFldSuffix = "",
        string inRecPrefix = "",
        string inRecSuffix = "")
```

Parameter	Description
inRecIndex	The index of the record to be returned. Default -- current.
inFldDelimiter	The char to be used as the delimiter of the fields. Default - TAB.
inFldPrefix	The prefix for each field.
inFldSuffix	The suffix for each field.
inRecPrefix	The prefix for record.
inRecSuffix	The suffix for record.

Returns: string

Returns the current record of cursor as string that contains field values delimited by the specified field delimiter.

Also you can specify prefix and suffix strings to be used with each field. And you can specify prefix and suffix for the whole record. These parameters can be useful for example, to produce HTML code.

Example:

```
str = cursor.getRecordAsString()      -- returns values of current record.
str = cursor.getRecordAsString( n )   -- returns values of the n-th record.
```

Example:

```
str = cursor.getRecordAsString( 0, TAB, "<TD>", "</TD>", "<TR>", "</TR>" )
```

```
getRecordsAsString(
        integer inFromRec = 0,
        integer inMaxRecords = -1,
        string inFldDelimiter = TAB,
        string inRecDelimiter = RETURN)
        string inFldPrefix = "",
        string inFldSuffix = "",
        string inRecPrefix = "",
        string inRecSuffix = "")
```

Parameter	Description
inFromRec	The index of the start record.Default is current.
inMaxRecords	The maximal number of the records to return. Default – all.
inFldDelimiter	The char to be used as the delimiter of the fields. Default – TAB.
inRecDelimiter	The char to be used as the delimiter of the records. Default – '\r'.
inFldPrefix	The prefix for each field.
inFldSuffix	The suffix for each field.
inRecPrefix	The prefix for record.
inRecSuffix	The suffix for record.

Returns: string

Returns a string that contains the concatenated field values of several records of a cursor. You can specify a range of records to be used.

Also you can specify prefix and suffix strings to be used with each field. And you can specify prefix and suffix for the whole record. These parameters can be useful for example, to produce HTML code.

Example:

```
str = cursor.getRecordsAsString()            -- returns list of ALL records.
str = cursor.getRecordsAsString( 1, 10 )     -- returns list of 1-10 records.
```

Example:

```
str = cursor.getRecordAsString(
        0, -1, TAB, RETURN, "<TD>", "</TD>", "<TR>", "</TR>" )
```

Import/Export Methods

```
ImportText(
        string inFile,
        string inFieldDelimiter = chr(09),
        string inLineDelimiter = LE,
        string inEncoding = "UTF-16",
        integer inHasColumnHeader = FALSE )
```

Parameter	Description
inFile	File to be imported.
inFieldDelimiter	Character to be used as a field delimeter.
inLineDelimiter	Character to be used as a record delimeter.
inEncoding	The encoding of the file to import.
inHasColumnHeader	TRUE if the import file has a column header line.

Returns: VOID

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

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

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

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

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

Example:

```
// we need create cursor with no records
curs = db.SqlSelect( "SELECT * FROM myTable WHERE FALSE" )
curs.importText( fileToImport, TAB, CR )
```

```
exportText (
        string inFile,
        string inFieldDelimiter = chr(09),
        string inLineDelimiter = LE,
        string inEncoding = "UTF-16",
        integer inHasColumnHeader = FALSE )
```

Parameter	Description
inFile	File to be imported.
inFieldDelimiter	Character to be used as a field delimiter.
inLineDelimiter	Character to be used as a record delimiter.
inEncoding	The encoding of the file to import.
inHasColumnHeader	TRUE if import file has column header line.

Returns: VOID

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

Example:

```
curs.exportText( fileToExport, TAB, CR )
```

toArraySet ()

This method establish a brige between cursors and sets. You can use this method to obtain an ArraySet that contains RecID values selected by cursor and in the correct order.

Important to note, that this method will work only with cursor built on the single table. You cannot use it for JOIN or GROUP BY results, for example.

TIP. If your target is to build cursor and convert it into set, then it is good idea to SELECT RecID only.

Example:

```
curs = db.sqlSelect( "SELECT RecID FROM T WHERE ..." )

arraySet = curs.toArraySet()

curs = VOID    // we do not need cursor any more.
```

VSet Xtra

Properties

integer count (r/o)
boolesn isSorted (r/o)
boolean isEmpty (r/o)

Construction Methods

clone()

Element Methods

append(integer inValue)
remove(integer inValue)
include(integer inValue)
makeNewIterator()

sort()

Set Operations

union(object inRightSet)
intersection(object inRightSet)
difference(object inRightSet)
symmetricDifference(object inRightSet)

Properties

Integer count (r/o)

The number of items in the Set.

Example:

```
count = set1.count
```

boolean isSorted (r/o)

Returns TRUE if the Set is sorted.

Example:

```
sorted = set1.isSorted
```

boolean isEmpty (r/o)

Returns TRUE if the Set is empty.

Example:

```
empty = set1.isEmpty
```

Construction Methods

clone()

Returns: VSet

Clones this Set, i.e. create and return a new set which is of the same type, has the same size and contains the same items.

Example:

```
s2 = s1.clone()
```

Element Methods

append(integer inValue)

Parameter: **Description:**
inValue A value.

Returns: VOID

Appends a new value to the Set.

Example:

```
set.append( recID )
```

remove(integer inValue)

Parameter: **Description:**
inValue A value.

Returns: VOID

Removes the specified value from the Set.

Example:

```
set.remove( recID )
```

include(integer inValue)

Parameter: **Description:**
inValue A value.

Returns: boolean

Returns TRUE if the Set contains the specified value.

Example:

```
found = set.include( recID )
```

makeNewIterator()

Returns: VSetIterator

Creates and returns a new Iterator for this Set.

Example:

```
iter = s1.makeNewIterator()
```

Sort()

Returns: VOID

Sorts the Set.

Example:

```
s1.sort()
```

Set Operations

union(object inRightSet)

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

Returns: VSet

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

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

Example:

```
s1.union( s2 )
```

intersection(object inRightSet)

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

Returns: VSet

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

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

Example:

```
s1.intersection( s2 )
```

difference(object inRightSet)

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

Returns: VSet

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

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

Example:

```
s1.difference( s2 )
```

symmetricDifference(object inRightSet)

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

Returns: VSet

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

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

```
s1.symmetricDifference( s2 )
```

VSetIterator Xtra

Properties

integer value (r/o)

Methods

firstItem()
lastItem()
nextItem()
prevItem()

Properties

integer value (r/o)

Returns the current value of the iterator.

Example:

```
v = iter.value
```

VSetIterator Methods

firstItem()

Returns: integer

Moves the iterator to the first item of the Set.
Returns the value of the item if it is found, else returns 0.

Example:

```
v = iter.firstItem
```

lastItem()

Returns: integer

Moves the iterator to the first item of the Set.
Returns the value of the item if it is found, else returns 0.

Example:

```
v = iter.lastItem
```

nextItem()

Returns: integer

Moves the iterator to the first item of the Set.
Returns the value of the item if it is found, else returns 0.

Example:

```
v = iter.nextItem
```

prevItem()

Returns: integer

Moves iterator to the first item of the Set.
Returns the value of item if it is found, else returns 0.

Example:

```
v = iter.prevItem
```

VLink Xtra

Properties

Integer	branchCount	(r/o)
Integer	ID	(r/o)
boolean	isTemporary	(r/o)
symbol	leftType	(r/o)
symbol	rightType	(r/o)
string	name	
Integer	onDelete	
Integer	onUpdate	

Table Methods

getIsBetween(integer inTableA, integer inTableB)

Table(integer inIndex)

Search Methods

```
findLinked(
        integer inRecID,
        object inTableA,
        object inTableB,
        symbol inRecursionDirection = #kFromParentToChild )

findExclusivelyLinked(
        integer inRecID,
        object inTableA,
        object inTableB,
        symbol inRecursionDirection = #kFromParentToChild )

findAllLinked(
        object inTableA,
        object inTableB,
        symbol inRecursionDirection = #kFromParentToChild )
```

Linking Methods

```
countLinked (
        integer inRecID,
        object inTableA,
        object inTableB,
        symbol inRecursionDirection = #kFromParentToChild )

linkRecords(
        integer inRecIDA,
        integer inRecIDB,
        * )

unlinkRecords(
        integer inRecIDA,
        integer inRecIDB,
        * )

deleteLinkedRecords(
        integer inRecID,
        VTable inTableA,
        symbol inRecursionDirection = #kFromParentToChild )

deleteAllLinkedRecords( VTable inTableA )

getIsLinked(
        integer inRecIDA,
        integer inRecIDB,
        * )
```

Properties

Integer branchCount (r/o)

Returns the number of branches for this link.

Example:

```
brc = Link.branchCount
```

Integer ID (r/o)

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

Example:

```
link_id = Link.ID
```

Boolean isTemporary (r/o)

Returns TRUE if this link is temporary and will not be saved into the database schema.

Example:

```
tmp = Link.isTemporary
```

symbol leftType (r/o)

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

Example:

```
lt = Link.leftType
```

symbol rightType (r/o)

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

Example:

```
rt = Link.rightType
```

string name

Returns the name of the link.

Example:

```
s = L ink.name
```

Integer onDelete

The behavior on deletion of the record-owner.

Example:

```
v = link.onDelete
```

Integer onUpdate

The behavior on update of the record-owner.

Example:

```
v = Link.onUpdate
```

Table Methods

getIsBetween(
integer inTableA,
integer inTableB)

Parameter:	Description:
inTableA	Left table of link.
inTableB	Right table of link.

Returns: boolean

Returns TRUE if this Link links both specified Tables.

Example:

```
res = Link.getIsBetween( TablA, TablB )
```

Table(integer inIndex)

Parameter:	Description:
inIndex	The index of table.

Returns: VTable

Returns a table of the link by index.

Example:

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

Search Methods

```
findLinked(
        integer inRecID,
        object inTableA,
        object inTableB,
        symbol inRecursionDirection = #kFromParentToChild )
```

Parameter:	Description:
inRecID	The recID of a record of the left table.
inTableA	Left table of link.
inTableA	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns: VArraySet

Returns the records from table2 linked to recID from inTable1. If zero records are found then returns VOID.

For a recursive link you should specify the parameter inRecursionDirection. If you specify kFromParentToChild then the function will use child records of the inRecID record. Otherwise it will use parent record(s) of the inRecID record.

Example:

```
res = Link.findLinked( rec, Tbl1, Tbl2 )
```

findExclusivelyLinked(
integer inRecID,
object inTableA,
object inTableB,
symbol inRecursionDirection = #kFromParentToChild)

Parameter:	**Description:**
inRecID	The recID of a record of the left table.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns: VArraySet

Returns the records from inTableB linked to the record inRecID of inTableA and only to it. If zero records are found then returns VOID.

For a recursive link you should specify the parameter inRecursionDirection. If you specify kFromParentToChild then the function will use child records of the inRecID record. Otherwise it will use parent record(s) of the inRecID record.

Note: This function returns result different from FindLinked() function only for M : M link.
Example:

```
res = Link.findExclusivelyLinked( rec, TblA, TblB )
```

---

findAllLinked(
object inTableA,
object inTableB,
symbol inRecursionDirection = #kFromParentToChild)

Parameter:	**Description:**
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns: VArraySet

Returns all records of table2 linked to any record of table1. If zero records are found then returns VOID.

Example:

```
tbl = Link.findAllLinked( TblA, TblB )
```

Linking Methods

```
countLinked (
        integer inRecID,
        object inTableA,
        object inTableB,
        symbol inRecursionDirection = #kFromParentToChild )
```

Parameter:	Description:
inRecID	The recID of a record of the left table.
inTableA	Left table of link.
inTableB	Right table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns: VArraySet

Returns the number of records of table inTableB linked to the record inRecID of table inTableA.

For a recursive link you should specify the parameter inRecursionDirection. If you specify kFromParentToChild then the function will use child records of the inRecID record. Otherwise it will use parent record(s) of the inRecID record.

Example:

```
tbl = Link.countLinked( rec, TblA, TblB )
```

```
linkRecords(
        integer inRecIDA,
        integer inRecIDB,
        * )
```

Parameter:	Description:
inRecIDA	RecID of record of the first Table of Link.
inRecIDB	RecID of record of the second Table of Link.
*	More inRecID parameters

Returns: VArraySet

Establishes a link between records of linked Tables, specified as an array of RecID values (Valentina 2.0 supports 2-branch links only, so 2 records must be specified).

The array must contains the correct number of values, in the order of branches of this link. The order of branches corresponds to the order of Tables on link creation.

Example:

```
// Link record 1 of the left table to the record 3 of the right table of the Link.
Link.linkRecords( 1, 3 )
```

```
unlinkRecords(
        integer  inRecIDA,
        integer  inRecIDB,
        * )
```

Parameter:	**Description:**
inRecIDA	RecID of record of the first Table of Link.
inRecIDB	RecID of record of the second Table of Link.
*	More inRecID parameters

Returns: VArraySet

Breaks the link between records of the linked Table specified as an array of RecID values.

The array must contain the correct number of values, in the order of branches of this link. The order of branches corresponds to the order of Tables on link creation.

Example:

```
Link.unlinkRecords( 1, 3 )
```

```
deleteLinkedRecords(
        integer inRecID,
        VTable inTableA,
        symbol inRecursionDirection = #kFromParentToChild )
```

Parameter:	**Description:**
inRecID	The recID of a record of the left table.
inTableA	Left table of link.
inRecursionDirection	The direction of movement for a recursive link.

Returns: VArraySet

Removes all records that are linked by this Link to the record inRecID of table inTableA.

The action of this function depends on the DeletionControl parameter of the link, which can be { refuse, delete some records, update some records }.

ERRORS: errRestrict.

Example:

```
Link.deleteLinkedRecords( rec, TblA )
```

deleteAllLinkedRecords(VTable inTableA)

Parameter:	**Description:**
inTableA	Left table of link.

Returns: VArraySet

Removes all records linked by this Link to the any record of table tableA.

The action of this function depends on the DeletionControl parameter of the link, which can be { refuse, delete some records, update some records }.

ERRORS: errRestrict.

Example:

```
Link.deleteAllLinkedRecords( TblA )
```

getIsLinked(
integer inRecIDA,
integer inRecIDB,
*)

Parameter:	**Description:**
inRecIDA	RecID of record of the first Table of Link.
inRecIDB	RecID of record of the second Table of Link.
*	More inRecID parameters

Returns: Boolean

Returns TRUE, if the two specified records are linked.

Example:

```
res = Link.getIsLinked( 3, 2 )
```

VServer Xtra

Properties

integer available (r/o)
integer connectionCount (r/o)
integer databaseCount (r/o)
string hostName (r/o)
integer port (r/o)
integer userCount (r/o)
string userName (r/o)
string version (r/o)

Method

VServer(
VConnection inConnection)

Connection Methods

cancelConnection (integer inConnectionID)
restart ()
refresh ()
shutdown ()

INI-File Methods

getVariable(string inName)
setVariable(string inName, string inValue)

Master databases Methods

registerDatabase(string inDbName, [string inServerPath])
unRegisterDatabase (string inDbName)

registerProject(string inProjectName)
unRegisterProject (string inProjectName)

User Methods

addUser (string inUserName, string inPassword, integer isAdmin = FALSE)
removeUser (string inUserName)
changeUserPassword (string inUserName, string inNewPassword)

getUserName(integer inUserIndex)
getUserIsAdmin(integer inUserIndex)

DatabaseInfo methods

DatabaseInfo(any dbNameOrIndex)

Description

You will only need to use this class in developing the server portion of a Server application. This class allows you to develop your own front end for VServer. It allows to managing parameters of the Server for a user which has administration rights, locally or remotely.

Properties

integer available (r/o)

Tests server for availability.

Example:

```
res = server.available
```

integer connectionCount (r/o)

Returns the number of active connections.

Example:

```
connCount = server.connectionCount
```

integer databaseCount (r/o)

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

Example:

```
dbCount = server.databaseCount
```

string hostName (r/o)

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

Example:

```
hostName = server.hostName
```

integer port (r/o)

Returns the port number of the server host.

Example:

```
port = server.port
```

integer userCount (r/o)

Returns the number of registered users.

Example:

```
count = server.userCount
```

string userName (r/o)

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

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

Example:

```
userName = server.userName
```

string version (r/o)

Returns a string that contains the VServer version number.

Example:

```
version = server.version
```

VServer Method

VServer(
VConnection inConnection)

Parameter	Description
inConnection	A connection object constructed before.

This method constructs VServer Xtra object. You must provide a VConnection object, which already have open() connection to VServer.

Note: Only Administrator User(s) can use VServer class and its methods.

Example:

```
connection = new( Xtra "VConnection",  "localhost", "sa", "sa" )
connection.open()
...
vsrv = new( Xtra "VServer", connection )
...
```

Connection Methods

cancelConnection(integer inConnectionID)

Parameter	Description
inConnectionID	The connection ID.

Returns: VOID

Cancels an existing connection by its ID.

Example:

```
server.cancelConnection( connID )
```

restart()

Returns: VOID

Forces a restart of the VServer.

Example:

```
server.restart()
```

refresh()

Returns: VOID

This method allows you to refresh the list of DatabaseInfo objects. This method sends a request to the Valentina Server.

Example:

```
server.refresh()
```

shutdown()

Returns: VOID

Shuts down the VServer.

Note: After this operation there is no way to restart VServer from the application. If you want to restart the VServer, use Restart( ).

Example:

```
server.shutdown()
```

INI-File Methods

getVariable(string inName)

Parameter:	Description:
inName	The name of server variable.

Returns: string

This method allows you to read a value of the specified Server Variable. The name of the variable is case insensitive. With names of variables you can use constants of the INI-file of VServer. For more information, refer to the Valentina Server documentation.

Example:

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

setVariable(string inName, string inValue)

Parameter:	Description:
inName	The name of the server variable.
inValue	New value for this variable.

Returns: VOID

This method allows you to change a value of the specified Server Variable. The name of variable is case insensitive. With names of variables you can use constants of the INI-file of VServer. For more information, refer to the Valentina Server documentation.

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

Example:

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

Master databases Methods

registerDatabase(
string inDbName,
string inServerPath = "")

Parameter	Description
inDbName	The name of the database.
inServerPath	The full path of the database located on the server computer.

Returns: VOID

You can use this method to register in Vserver some existed database. This command adds a new record to the Master Database.

Usually you need just to drop a database into the folder pointed by .ini varable "System-Catalog", and call this method specifying only the name of database. Also it is possible to specify the full path of database on the server computer.

Note: For a MacOS X version of Valentina Server, use a UNIX path.

Errors:

The Database Name already exists.

Example:

```
server.registerDatabase( "DbName"  )
```

This assumes that a database with name "DbName" or "DbName.vdb" exists in the "databases" folder of VServer.

Example:

```
server.registerDatabase( "Accounting", "C:\SomeCompany\account2002.vdb" )
```

unRegisterDatabase(string inDbName)

Parameter	Description
inDbName	The name of a database.

Returns: VOID

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

Errors:

Database Name not found.

Example:

```
vsrv.unRegisterDatabase( "Accounting")
```

registerProject(
string inProjName,
string inServerPath = "")

Parameter	Description
inProjName	The name of the Project.
inServerPath	The full path of the Project located on the server computer.

Returns: VOID

You can use this method to register in Vserver some existed Project. This command adds a new record to the Master Project.

Usually you need just to drop a Project into the folder pointed by .ini varable "SystemCatalog", and call this method specifying only the name of Project. Also it is possible to specify the full path of Project on the server computer.

Note: For a MacOS X version of Valentina Server, use a UNIX path.

Errors:
The Project Name already exists.

Example:

```
server.registerProject( "ProjName"  )
```

This assumes that a Project with name "ProjName" or "ProjName.vdb" exists in the "Projects" folder of VServer.

Example:

```
server.registerProject( "Accounting", "C:\SomeCompany\account2002.vsp" )
```

unRegisterProject(string inProjName)

Parameter	Description
inProjName	The name of a Project.

Returns: VOID

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

Errors:

Project Name not found.

Example:

```
vsrv.unRegisterProject( "Accounting")
```

User Methods

addUser(
string inUserName,
string inPassword,
integer isAdmin = FALSE)

Parameter	Description
inUserName	The user name.
inPassword	The password for this user.
IsAdmin	TRUE if this user has administrator permissions.

Returns: VOID

An Administrator can add new users to the Master Database.

Errors:
The user name already exists.

Example:

```
server.addUser(  "Peter", "a1234fteg4" )
```

removeUser(string inUserName)

Parameter	Description
inUserName	The user name.

Returns: VOID

An administrator can remove users from the Master Database.

Errors:
The user name is not found.

Example:

```
server.removeUser(  "Peter" )
```

changeUserPassword(
string inUserName,
string inNewPassword)

Parameter	Description
inUserName	The user name.
inNewPassword	New password for this user.

Returns: VOID

An administrator can change the password of a user.

Errors:
The user name is not found.

Example:

```
server.changeUserPassword(  "Peter", "rvsa3341" )
```

---

getUserName(integer inUserIndex)

Parameter	Description
inUserIndex	The user index.

Returns: string

Returns the name of the user based on the index.

Example:

```
server.getUserName( i )
```

---

getUserIsAdmin(integer inUserIndex)

Parameter	Description
inUserIndex	The user index.

Returns: boolean

Returns TRUE if the specified user is an administrator.

Example:

```
res = server.getUserIsAdmin( i )
```

DatabaseInfo Methods

DatabaseInfo(any dbNameOrIndex)

Parameter:	Description:
dbNameOrIndex	1-based index or name

Returns: VDatabaseInfo

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

A Vserver obtains a list of the DatabaseInfo upon OpenSession. You can periodically refresh this list using the Refresh() method.

Example:

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

DatabaseInfo Xtra

Properties

integer	clientCount
integer	cursorCount
string	name
string	path

Method

getClientInfo(integer inIndex)

refresh()

Properties

integer clientCount

Returns the clients count of the database.

Example:

```
count = dbi.clientCount
```

integer cursorCount

Returns the cursor count for the database.

Example:

```
count = dbi.cursorCount
```

string name

Returns the name of the database.

Example:

```
name = dbi.name
```

string path

Returns the path(on the server) of the database.

Example:

```
path = dbi.path
```

Methods

---

getClientInfo(Integer inIndex)

Parameter:	**Description:**
inIndex	The name of server variable.

Returns:VClientInfo

Returns a reference to the client info object by its index.

Example:

```
for i = 1 to dbi.clientCount
        cli = dbi.getClientInfo(i)
        ....
next
```

---

refresh()

Returns: VOID

Refreshes the database info.

Example:

```
dbi.refresh()
```

VClientInfo Xtra

Properties

string	address
integer	connectionID
integer	cursorCount
string	login
integer	port

Method

refresh ()

Properties

string address

Returns the address(name or IP-address) of the remote client.

Example:

```
adress = clientInfo.adress
```

integer connectionID

Returns the connection id of the remote client.

Example:

```
ID = clientInfo.connectionID
```

integer cursorCount

Returns count of cursors created by the remote client.

Example:

```
count = clientInfo.cursorCount
```

string login

Returns the login of the remote client.

Example:

```
login = clientInfo.login
```

integer port

Returns the port number of the remote client.

Example:

```
port = clientInfo.port
```

VClientInfo Methods

refresh()

Returns: VOID

Refreshes the client info .

Example:

```
cli.refresh()
```

VProject Xtra

Properties

reportCount	integer (r/o)
reportName(index)	integer (r/o)

Construction Methods

```
VProject(
        string          inProjectLocation )

VProject(
        VConnection     inConnection,
        string          inProjectName )
```

Report Methods

```
MakeNewReport(
        integer         inIndex,
        VDatabase       inDatabase,
        string          inQuery,
        symbol          inCursorLocation  = #kClientSide,
        symbol          inLockType        = #kReadOnly,
        symbol          inCursorDirection = #kForwardOnly

MakeNewReport(
        string          inName,
        VDatabase       inDatabase,
        string          inQuery,
        symbol          inCursorLocation = #kClientSide,
        symbol          inLockType       = #kReadOnly,
        symbol          inCursorDirection = #kForwardOnly )
```

Properties

integer reportCount (r/o)

Returns the count of reports inside of this container.

Example:

```
reports_count = my_project.reportCount
```

string reportName (r/o)

Returns the name of Nth reports. This name can be used, for example, to show the list of all reports in the project.

Example:

```
for i = 1 to reports_count
   report_name = my_project.ReportName( i )
end
```

Construction Methods

VProject(
string inProjectLocation)

Returns: VOID

inProjectLocation The location of a Valentina project file “*.vsp”.

Description:

Constructs a new instance of VProject class. You need provide a disk location of ”.vsp” file that contains description of one or more Reports

Example:

```
my_project = new ( Xtra "VProject", "MyProject.vsp" )

-- Now you can use methods of VProject class to:
-- * investigate how many reports are inside of this container.
-- * get their names to display in e.g. menu
-- * extract single reports creating VReport class instance.
```

VProject(
VConnection inConnection,
string inProjectName)

Returns: VOID

inConnection A connection to a Valentina Server
inProjectName The name of a VProject hosted by that VServer.

Description:

Constructs a new instance of VProject class to handle project hosted on a Valentina Server. You need provide a connection object and the project name known to the Valentina Server.

Example:

```
my_project = new (  Xtra "VProject", connectionToMyServer, "MyProject.vsp" )

-- Now you can use methods of VProject class to:
-- * investigate how many reports are inside of this container.
-- * get their names to display in e.g. menu
-- * extract single reports creating VReport class instance.
```

Report Factory Methods

```
MakeNewReport(
   integer     inIndex,
   VDatabase   inDatabase,
   String      inQuery,
   symbol      inCursorLocation = #kClientSide,
   symbol      inLockType       = #kReadOnly,
   symbol      inCursorDirection = #kForwardOnly )
```

Parameter	Description
inIndex	The index of a report in range 1 .. ReportCount.
inDatabase	The database that will be used as a data source.
inQuery	The SQL string of a query.
inCursorLocation	The location of cursor. See CursorLocation types.
inLockType	The lock type for records of a cursor. See LockType types.
inCursorDirection	The direction of a cursor. See CursorDirection types.

Returns: VReport

This method plays role of a VReport class factory. Returns instance of the VReport class for Nth report of this project. Returns NULL if the specified report is not found.

To create a report instance, the VREPORT DLL must know:

- A database that will be used to get data.
- Query that should be executed to get data
- Some optional parameters for this query.

IMPORTANT: If this project is local then inDatabase can be both local and remote located on any Valentina Server. But if project is hosted on some Valentina Server, then inDatabase must be hosted on the same server.

* Parameters inCursorLocation, inLockType, inCursorDirection are the same as for VDatabase.SqlSelect() method. Please read details about them there.

* inQuery parameter must contain any SQL that returns VCursor. Usually this is SELECT statement, although can be SHOW statement or CALL procedure that returns cursor.

IMPORTANT: When you have design report in the Valentina Studio Pro, you have assign assigned some SQL SELECT query to it. And you have use fields of that cursor to build layout of the report. But that was during DESIGN mode.

Now, in the RUNTIME mode, you can provide totally different database and use other query. The only requirement is that query produces a cursor with the same field names that was used in the report layout. Otherwise report will produce nothing for that field.

Usually you will use SELECT with the same SELECT and FROM parts, and only will modify WHERE part to select different records.

Example:

```
theReport = my_project.MakeNewReport( reportIndex, mDB, Query )
```

```
MakeNewReport(
   string      inName,
   VDatabase   inDatabase,
   String      inQuery,
   symbol      inCursorLocation = #kClientSide,
   symbol      inLockType       = #kReadOnly,
   symbol      inCursorDirection = #kForwardOnly )
```

Parameter	**Description**
inName	The name of a report.
inDatabase	The database that will be used as a data source.
inQuery	The SQL string of a query.
inCursorLocation	The location of cursor. See CursorLocation types.
inLockType	The lock type for records of a cursor. See LockType types.
inCursorDirection	The direction of a cursor. See CursorDirection types.

Returns: VReport

This method do the same as the above method, except that report is specified by its name instead of index. Please see detailed description above

Example:

```
theReport = my_project.MakeNewReport( "report_1", mDB, Query )
```

VReport Xtra

Properties

integer pageCount (r/o)

Preview Properties

integer previewZoom
integer previewWidth
integer previewHeight

Preview Methods

previewPage(integer inPageIndex)

Printing Methods

```
printToDisk(
        string      inLocation,
        symbol      inPrintType,
        integer     inStartPageIndex = 0,
        integer     inEndPageIndex = 0 )
```

Properties

integer pageCount (r/o)

Returns the count of pages that will be produced fro this report using the specified Cursor and the current Page format settings.

Example:

```
pages = report.PageCount
```

integer previewZoom

Specifies the preview zoom in percents 1-100. Affects only following calls of VReport. PreviewPage() method.

Example:

```
report.PageZoom = 50
preview = report.PreviewPage()
```

integer previewWidth

Specifies the width of preview in pixels. Affects only the following calls of VReport.Preview-Page() method.

Example:

```
report.PreviewWidth = 600
preview = report.PreviewPage()
```

integer previewHeight

Specifies the height of preview in pixels. Affects only the following calls of VReport.Pre-viewPage() method.

Example:

```
report.PreviewHeight = 600
preview = report.PreviewPage()
```

Preview Methods

previewPage(integer inPageIndex)

Parameter:	Description:
inPageIndex	index of report page to preview. Starts from 1.

Returns: Picture

Description:

This method generates preview of Nth page of the report.

Usually you need this to build some kind of user interface to show preview of first report pages before printing.

Example:

```
page_preview_pict = report.PreviewPage( 1 )
```

Printing Methods

```
PrintToDisk(
        string      inLocation,
        symbol      inPrintType,
        integer     inStartPageIndex = 0,
        integer     inEndPageIndex = 0   )
```

Parameter:	Description:
inLocation	The location for generated file.
inPrintType	Specifies the format of generated report.
inStartPageIndex	The index of the first page to be printed (1..N). Zero to print all records of the report
inEndPageIndex	The index of the last page to be printed (1..N).

Returns: VOID

Description:

Prints all pages or the specified range of pages of the report to the disk file at the given location.

You can specify format of produced file with help of the inPrintType parameter. . Usually you will use such values as kToPDF, kToHTML, kToPicture_JPG.

Example:

```
report.PrintToDisk( #kToHTML, "report_1.html" )
report.PrintToDisk( #kToPDF, "report_1.pdf" )
```