VALENTINA 1.2.1

written by Ruslan Zasukhin,
Paradigma (www.paradigmasoft.com)

Technical notes

Acknowledgments:

Brian Blood	**-** for help in development of this product.
Andy Bachorski	**-** consultant, Apple Computer, Inc.
Hideaki Iimori	- for help in internationalization

Introduction

Valentina is a Relational Database Management System (RDBMS) which has an extremely fast and powerful database engine.

Valentina has its own file system, so many logical files of the database can be stored in one disk file. Valentina keeps a database in 4 separate disk files:

“dbName.vdb”	- description of the database
“dbName.dat”	- major data of the database (Tables, Relations)
“dbName.blb”	- contains BLOB-data (described below)
“dbName.ind”	- temporary data of the database (indexes, and any other temporary files)

■ BLOB data is of variable size and can contain QuickTime movies, sound files, PICT files, etc. Basically it can hold anything that can be stored in a file on the Mac or PC.

■ The file “dbName.ind” can be deleted and the database will not be corrupted since Valentina will rebuild indexes when they are needed. You can use this if there is any crash during debugging.

Characteristics**:**

Max length of the disk file:	$2^{(32-1)}$ (2 GB)
Max number of Tables:	2^{32} (4’294’967’295)
Max number of Fields in the Table:	2^{16} (65’535)
Max number of records in the Table:	2^{32} (4’294’967’295)
Max size of a BLOB record:	$2^{(32-1)}$ (2 GB)

Valentina operates with the following basic concepts:
- Table,
- Field of a table,
- Record of a table,
- RecID - record identifier
- Selection - set of records.

Table:
- A database of Valentina can have one or more tables.
- The table represents a file which has a header and keeps records of fixed size.
- The table has columns (Fields) and records (Rows).
- The address of the record can be calculated.
- Deleted records are marked as such and their space will be reused.

Record:
- Is a composition of one or more fields.
- Each record has a unique RecID (valid range is 1...).
- 0 is used as the RecID of an undefined record.

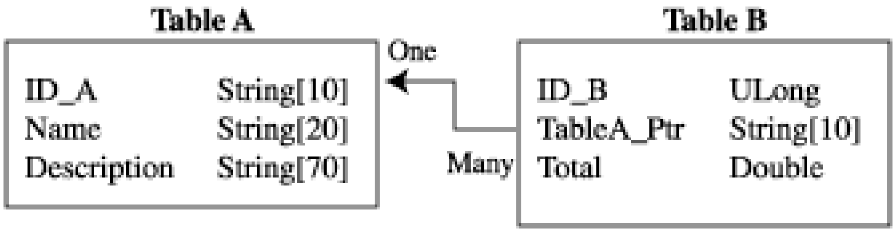
Relating Tables:

To establish a relation between 2 tables, Valentina offers a special mechanism - field of type “ObjectPtr”. To understand how it works, let’s quickly consider traditional techniques of the relational and the network data models.

Relational data model:

In order to establish a relation between 2 tables, a relational database uses a pointer by Value:

Table A:	**ID_A**	**String[10]**
	Name	**String[20]**
	Description	**String[70]**
Table B:	**ID_B**	**ULong**
	TableA_Ptr	**String[10]**
	Total	**Double**



i.e. Table A must have the field “ID” (indexed, unique, required),
and Table B must have a field with the same attributes - pointer to the record in Table A. The related records have the same values in the fields “Table A.ID” and “Table B.Table_A_Ptr”.
As a result, we have a relation [Table A : Table B] of one to many [1 : M], i.e. one record of Table A can be related to many records of Table B.

Drawbacks:
1) A database developer must provide values for the unique identifier of Table A. There are 2 traditional ways:
a) As an identifier which is used in some real world value (i.e. social security number).
b) A special artificially generated number is used.

2) A field of any type can be used as the identifier of the record. So if a string[20] is used, then Table B must store an additional 20 bytes in the record and an additional 20 bytes must be stored in the index of the field “Table_A_Ptr”.

3) Access speed from the record of Table B to the related record of Table A is low, because the database must perform 4 steps:
a) Get the ID value from the field “Table_A_Ptr”.
b) Search the index of the field “Table A.ID”, using this value to get some internal ID of the record.
c) Search the primary index of Table A, using the internal ID to get the address of the record.
d) Load the record of Table A.

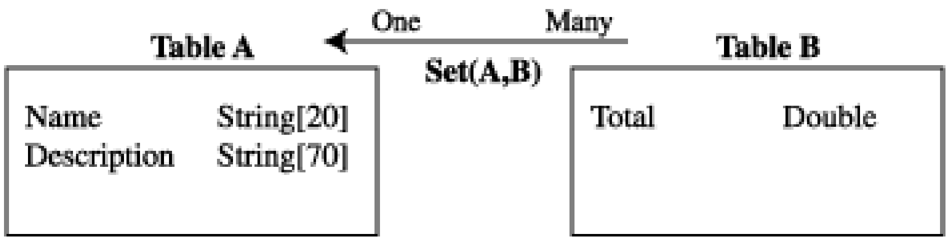
4) The RDBMS have no information about relation of tables. In other words for RDBMS Tables A and B are independent. Relation of Tables must be specified each time, for example via SQL-query.

Network data model:

The other, much less commonly used database technology is the pointer-based navigational network model. The same database structure for the network model looks like:

Table A:	**Name**	**String[20]**
	Description	**String[70]**
Table B:	**Total**	**Double**

Set (Table A, Table B);



As we can see, this model does not need ID fields in the tables, so there is less work for the developer. Instead there is an additional concept - “Set”. The system automatically links related records in the list in the folowing way:
- A record of Table A which is a parent record, has pointers to the first and the last related records of Table B (4 + 4 = 8 bytes);
- Each related record of Table B has pointers to the previous and the next related records and a pointer to the parent record (4 + 4 + 4 = 12 bytes).

We want to emphasize that the direct pointer is a direct address of the record. As a result, the system can find the parent record in the shortest time.

Drawbacks:
1) The internal representation of the record depends on the database structure. Indeed, if we add a new set, then the system must add an additional space where the pointers will be stored to each record.

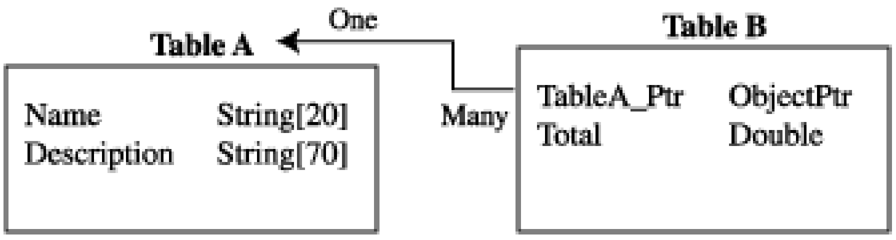
2) Sets do not allow for fast random or sorted access to the records, because there are no indexes.

3) Although on the first view network model is more simple then relational this is not true.
The work with sets is difficult and require from developer significant efforts.

Valentina’s data model:

Valentina suggests a hybrid of both previous technologies:
It stores direct pointers to the related records as normal fields of the table.

Table A:	**Name**	**String[20]**
	Description	**String[70]**
Table B:	**Table_A_Ptr**	**ObjectPtr;**
	Field2	**Double**



As we can see:
1) There are no ID fields in the tables. Forget about this headache forever!
2) There is a special type of field - “ObjectPtr”. This field is intended for storing RecIDs of the related One records (ulong, 4 bytes). Notice, pointer ObjectPtr points on the Table but not on field of the Table as in the relational data model.

So, we get traditional relational tables, but:
- The database structure is more simple and flexible.
- It needs less disk space, because:
  - There are no IDs in the tables and there are no their indexes.
  - Size of the pointer is always 4 bytes.
- A parent record can be found in a single logical disk access.

■ obviously, if you wish, you can define a relation between tables using the standard technique “pointer by value”.
■ in subsequent versions of Valentina there will be an additional, even more powerful and flexible mechanism of relations, it will be especialy effective for Many to Many relations.

AppleScript Dictionary of Valentina:

Valentina’s AppleScript dictionary includes the following database classes:
- database
- base object
- field
- record
- set

Using commands with these classes, you can do any database operation:
- Create database.
- Open/close database.
- Create a new table in the database.
- Remove a table from the database.
- Add/remove fields to a table.
- Change parameters of the fields (name, type, indexing...).
- Add/update/remove records to/from a table.
- Perform a compound search by several fields of a table.
- Sort selected records on one or more fields.

Class: database:

To begin work with a database you must at first create a new empty one or open existing. Valentina can manage multiple open database.

Properties:

name	[r/o] string

Elements:

base object	by index, by name.

Example Commands:

```
set theSpec to new file with prompt «New database» default name «Customers»
set DB to make new database with data theSpec
— Creates on the disk a new database at the specified location theSpec.

set DB to open database fileRef
— opens the database at the specified location fileRef.

flush database «Customers»
— Saves all changes in the memory buffer to disk for the ‘Customers’ database.

close database «Customers»
- close database, flushes not saved information on disk.
```

In order to work with base objects (tables), you can use the following commands:

```
make new base object with properties {name:»Component»} at end
— Creates a new base object in the database.
—  Must be called only once when you really want to create a new table.

delete base object «Component»
— Deletes the table «Component» and all its indexes from the database.

count of base objects in database «Invoices»
— Returns the number of base objects in the database

base object 1 of database «Customers»
— Returns a pointer to base object 1 of the database «Customers»

base object «Component» of Database «Customers»
— Finds the base object by name
```

Class: base object:

The class ‘base object’ implements a table in a database. It is called a ‘base object’ because in subsequent versions of Valentina there will be a significant difference between ‘base object’ and ‘Table’. Valentina will have database inheritance, so one base object will be able manage several tables.

In this version of Valentina (1.2 or less) you can consider a ‘base object’ as a standard database table.

Properties:

name	string
current record	record

Elements:

field	by index, by name.
record	by ID.

Example Commands**:**

Working with fields:

```
tell base object «Person»
make new field with properties
     {name:»Description», type:tString, len:50 } at end
end tell

count of fields in base object «Person»
— Returns the number of fields in the «Person» table

field «Name» of base object «Person»
—  Returns a pointer to the field «Name» in the «Person» table
```

Working with records:

```
make new record
—  Adds a new record with current values of the fields to a table.

delete current record of base object «Person»

delete record i of SomeSelection

set theRecord to record i of SomeSelection
delete theRecord
— Deletes the specified record from a table.

delete records
— Deletes all records in a table.
- The ‘delete records’ command is much faster then deleting each record in a loop.
```

If the deleted record has related Many records then behavior of deletion depends on flag ‘deletion control’ of the tObjectPtr field. See description of tObjectPtr on details.

```
update
— Updates the current record with the new values of the fields.

count of records
— Returns the number of records in a table.

set current record to record i of SomeSelection
— Makes the specified record current, result is true if record exists.
```

The following example allows you to loop through records in a table using any sort order:

```
set Set1 to select all records of Customer
set Set2 to sort Set1 by {field «Last Name»}
set theCount to count of records in Set2

repeat with i from 1 to theCount
  set current record of Customer to record i of Set2
  set ValuesList to fields of Customer
  — Add code to do what you want with the records
end repeat
```

You will have fast access to the values of the fields of the current record using the following example:

```
set ValuesList to fields of Customers
  -> {«John», «Some address», «photo here :-)»}
```

As a result, using only one AppleEvent you can get list of current values of all fields of the table. If your table has 10 fields then this way will work up to 10 times faster.
You can get values from the list in the following manner:

```
  set theFirstName to item 1 of ValuesList
```

In the similar way, you can set values of a new record or update values of the existing record:

```
 set fields of Customers to
     { «John», «Some address», thePhoto  }
```

There are 2 important assumptions here:
1) Values in the list must have the same order as the order of the fields in the table.
2) In the list, values for all fields must be specified or values of the last fields can be skipped. So, in the previous example, you can skip the field “thePhoto”: { «John», «Some address» }, but you can not skip the field “address”: { «John», thePhoto }.

Commands to build a selection of records:

```
set Selection1 to select records
— Selects all records in the base object

set Selection2 to select records where
  {field «Cost», «>100 and <400»,
   field «Date», «>1/01/99 and <1/03/99»}
— Selects records which match the specified conditions.
```

You can perform a compound search in several fields of a table, just like in FileMaker Pro. For a search, you must specify the following data: { Field, its condition string }

Say we want to find invoices where:

Field «Total»:	«>=100 and <=500» and
Field «Date»:	«>=03/01/1999 and <=03/31/1999»

In the script, it looks like:

```
tell Invoice
   set Result to select records where {
   field «Total», «>=100 and <500»,
   field «Date»,  «>=03/01/1999 and <=03/31/1999»  }
end tell
```

The following script searches for records which have a value of the field «Total» equal to 450:

```
set Result to select records where {field «Total», «450»}
```

Note, you must always send the condition as a string, but not a numeric value. So, if you have a numeric script variable then you can write:

```
set Result to select records
where {field «Total», theValue as string}
```

Only the tObjectPtr field works in a different way — you must specify the parent record:

```
set InvoicesSet to select records where
  {field «CustomerPTR», current record of Customer}
```

In the condition string you can use the following comparison operators: "=", ">", ">=", "<", "<="
Operator "=" is optional, for non string fields, i.e. search conditions "=450" and "450" are the same.

But for tString field operator "=" means: find records which match exactly to specified string. If the operator "=" is not specified then Valentina find all records which begins with the specified string.
Example:

"=aaa",	returns records with value "aaa".
"aaa",	returns all records which start with "aaa" like "aaagg", "aaa", "aaa256", ...

You can build a compound conditional string using the logical operators "and" and "or".

If the field is nullable then you can use search conditions like:
"Null", "not null", "<50 and not NULL":

```
set InvoicesSet to select records where
  {field «Total», "NULL"}
```

Notice, the keywords "and", "or", "not", "null" are not case sensetive.

Examples working with selections:

```
sort Selection2 by {field «Description»}
  — Sorts the selection on the specified fields.

set field «Cost» to 458
— This only changes the value in memory.
replace value of field «Cost» in Selection1
— This command replaces values of the field «Cost» in all selected records on disk
with the current value of the field in memory.
```

Other commands:

'Import'	— Adds data from an ASCII file into a table
from ascii file	— file spec
to	— list
[field delimiter]	— string (optional)

'Export'	— Takes data from a table and saves it into an ASCII file
from	— list
to ascii file	— file spec
[using selection**]**	— the selection of records to export, on default export all records
[field delimiter]	— string (optional)

Default field delimiter is tab. Valid field delimiters include comma(","), space (" ") and tab

Examples:

```
import from ascii file theFile to {field «Name», field «Address»}
— Imports records from the tab delimited ASCII text file theFile using the list of
fields as a map:
  First field     -> field 'Name'
  Second field    -> field 'Address'

import from ascii file theFile to
    {field «Name», field «Address»} with field delimiter «,»
— Imports data from the comma separated text file 'theFile'

export from {field «Name», field «Address»} to ascii file theSpec
— Exports the specified fields of selected records to the ASCII text file 'theSpec'.
```

On import the list of fields can contain "0" as a reference to the "nil" field to skip a field from the imported file. For example: {field "Name", 0, field "Address"}.

'Flush'

You may need this command if you have to add to a table many new records in a loop. After the loop, you should call flush to save all changes to disk.

```
flush base object «Customer» of database «Customers»
— Saves information in memory from the 'Customer' table to the hard disk.
```

Class: field:

Properties:

name	string	
type	enum{ tBool, tByte, tShort, tUshort,tLong, tUlong, tString, tDate, tTime, tBLOB, tObjectPtr }	
indexed	bool	
unique	bool	
nullable	bool	
length	integer	for fields with type tString only
language	****	for fields with type tString only
pointed object	base object	for fields with type tObjectPtr only
deletion control	leave/delete_many/cant	
block size	integer	size of the block (in bytes) for a BLOB field.
NULL	bool [r/o]	

Each field must have name (up to 32 bytes). Name of the field must be unique in the scope of its base object.

Valentina has all standard database field types plus the special type "ObjectPtr" to establish a relationship between 2 tables.

Type	**size in table**	**range**
boolean	1 bit	0,1
byte	1 byte	0..255
short	2 bytes	-32768..32767
ushort	2 bytes	0..65535
long	4 bytes	-2147483647..2147483647
ulong	4 bytes	0..4294967295
float	4 bytes	
doable	8 bytes	
string	1..255 bytes	
date	4 bytes	a date beetwen -2^{22}..2^{22} years
time	4 bytes	
BLOB	up to 4 GB	any binary data
ObjectPtr	4 bytes	

When Valentina needs to search or sort on some field it automatically builds the index for that field if it doesn't exist and set the flag 'indexed' to the true. If you set this flag to the false then you force Valentina to throw out the index of that field.

If the flag 'unique' is true then Valentina will not accept a new record with duplicate value for this field.

The flag 'nullable' define does the field accepts the NULL values.
On default flag 'nulable' is false. If it is true then field can store NULL values, this feature require storing of additional information on the disk – 1 bit per record of the nullable field.
Null is not the same as empty string "" or zero.
Null means – "value is not defined" or "value is not known", so unique field can have many records with Null value.
On sorting the records with Null value are placed first.

NULL -- this flag differs from others because it is property not field only but and of current record. It is true if the field's value of the current record is NULL. If the field is not nullable then it is false always.

Example of commands:

```
make new field with properties {name:»aaa»,type: tLong}
make new field with properties {name:»bbb»,type: tString, length:40}
make new field with properties
                   {name:»ccc»,type: tObjectPtr, pointed object: thePerson }
— Creates a new field in the table.

delete field «aaa»
— Removes the specified field from a table, the index file of the field is deleted
from the disk.

get data size of field «aaa»
— Returns the size of the current field’s value in bytes.

set field «Total» to 5254
— Set value of the field.

set theTotal to field «Total»
— Get value of the field and assign it to the script variable

set name of field «aaa» to «bbb»
— Changes the name of the field

set length of field «aaa» to 50
— Changes the length of the string field.

set type of field «bbb» to tUlong
— Changes the type of the database field.

set indexed of field «bbb» to false
— Deletes the index of the field.
```

tDate and tTime fields

To set value of this fields you must use the AppleScript format of date:
set field “BornDate” to date 3/3/1999

For date and time you must use format which is specified in the control panel “Date and Time” on your computer. This format must be used everywhere.

So if the script with USA format of date must be running on a computer with Japanese format of date then at first all dates in the text of script must be changed.

tString field:

The string field can store up to 255 bytes. If you use a single byte language it can store up to 255 characters. If you use 2-byte code language, then it can accept up to 127 characters. Property ‘length’ of the string field determines the size of the field in bytes (not in characters).

The string field has a property “language” which can accept short integer values – id of the corresponded itl2 resource of the system file or string – name of the corresponded itl2 resource.

The default ‘language’ value is –1 and then Valentina uses the byte-to-byte method of string comparison which sorts based on the ASCII value of the characters. This is the fastest method, but it can be used only for some single-byte languages (for example English).
If for your language this method is not acceptable then you must specify the language code for it.

Valentina allows for fields with different languages in the same table.

Example:
```
make new field with properties
     {name:»a», type:tString, length:50, language: 16536 }
     — Japanese language.

make new field with properties
     {name:»a», type:tString, length:50, language: “German” }
     — German language
```

If you set for a field, for example, language Japanese then you can mix in this field English and Japanese strings, if the language is German then you can mix English and German strings. In other words you can always store English strings because English letters have fixed values in the first part of the ASCII table.
You can change the language of the field at runtime. In this case Valentina will rebuild the index of the field using a new sort order.

tBLOB field

When you create a BLOB field, you can specify the size of the blocks for this field. This parameter cannot be changed afterwards. By default it is 10 KB. So, if the size of the BLOB-data is 22 KB, then 3 blocks (30 KB) will be used. If the data size is 3 KB, then 1 block (10 KB) will be used. In the latter case, you can optimize storing of the BLOB-data if you set ‘block size’ to 3 + 1 = 4 KB = (4 * 1024) bytes. As a result, you save 6 KB per BLOB-record.

The BLOB field type has one difference from the other field types. If you want to change (update) the contents of the existing BLOB record, then you must use the ‘update’ command but not the ‘set’ command.

Example:
```
  tell base object ‘Person’
    update field ‘Photo’ with theNewContents
    update  — don’t forget to update the record of the table.
  end tell
```

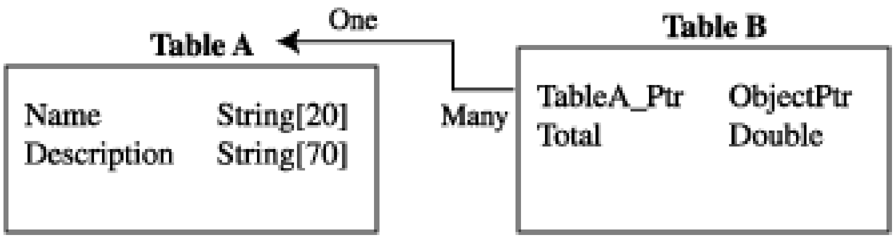
■ Valentina stores the contents of BLOB fields in a separate file. In the table record, only a reference (4 bytes) to the BLOB-data is stored.

■ Obviously that one table record can have several BLOB fields.

■ Between one table record and its BLOB-record there is relation One to One,
for example one man can have only one photo, and the one photo can be related only with one man. If you need to implement other= struture, for example [Many : Many] for “Building” : “Quarter map”, then you need put BLOB field in separate table and use standard relational techics.

tObjectPtr field

The field of type ‘tObjectPtr’ is intended to establish a relation [M:1] between 2 tables (base objects). It stores references to the related parent record (One record). This reference is a unsigned long number (4 bytes, ulong) and it is a physical number of the record in the table.



Parent Table,
One Table

Child Table,
Many Table

The ObjectPtr field must know:
a) The referenced object (parent object);
b) deletion control.

The Deletion control options regulate record deletion in the Many table when a record is deleted in the One table. There are 3 options:
1) Leave related Many records:
i.e. from the database only One record is deleted, the tObjectPtr of the related many records is set to the NULL automatically.
2) Delete related Many records:
from the database is deleted One record and all its Many records.
If a Many record also have related Many records in the third Table(s) then they can be deleted also (cascade deleting).
3) Can not delete if related Many:
The deletion of the One record is not allowed if there is at least one related Many record.

To set the value of this field you must use the ID of the current record of the referenced base object:

```
set field «CustomerPTR» of Invoice to current record of Customer

get field «CustomerPTR» of Invoice
     -> record id N of base object “Customer”
— Notice, the result is some record of the referenced base object.
```

Class: record

Implements a record of the base object. If the base object has no records, then it is empty.

Records have a fixed length, so Valentina can calculate the address of a record. Each record has a unique ID.

Deleted records are marked and their place and ID can be later reused. So, a table could have 50 physical records (with ID from 1 to 50) and, for example, 32 logical records (18 records are deleted).

Properties:

id	integer.

Commands:

```
make new record
— adds a new record with current values of the fields to the table.

delete (get current record of base object ‘Person’)
delete record i of SomeSelection
delete theRecord
     — Deletes the specified record from the table.

delete records
— Deletes all records in the table.
```

This latter command is much faster then deleting each record in a loop.
If deleted records are referenced by related Many records from other table(s) then behavior of this command will depend on property ‘deletion control’ of the field tObjectPtr (see description of the tObjectPtr).

```
update
— Updates the current record with the new values of the fields.

count of records
— Returns the number of records in the table.

set current record to record i of SomeSelection
  -> true
— Makes the specified record current. Result is true if record exists.


NOTE:
```

Each Table has one memory buffer where it keep values of the fields.
When some record become current then it is loaded to the memory buffer.
When you assign to the fields new values then you change memory buffer. Changes will affect the Table on disk only when you add to the table a new record or update current one.

To prepare memory buffer for a new record usualy you must call before ‘blank’ command:

```
tell Invoice
  blank
  set field “Total” to 246
  set field
end tell
```

Class: selection

The selection is a set of some records in a table. You can get the selection as the result of a search. Then you can sort that selection on some fields of the table, and then you can loop through records in the selection.

If you don't need the selection, then you should delete it to free RAM in Valentina. Valentina supports multiple selections in a base object.

Elements:

record	by index

Commands:

```
count of records in S1
— Returns the number of records of the selection.
— Returns 0 if the selection is empty.

record n of SomeSelection
— Returns n-th record of the selection.
— Note, the first record of the selection can be, for example, the 548-th record in the table.

set S1 to sort S1 by {field 'Total' }
— Sorts the selection on the field 'Total'
```

Important: sorting on fields deletes the original selection from RAM, and returns a new sorted selection. You must assign the new selection to a script variable to be able to use it later.

```
sort S1
— Sorts the selection on record ID, so if you loop through the table by this sorted selection, you will move from the beginning to the end of the table.

delete S1
— Deletes the selection.
```

Appendix

■ To improve efficiency of disk access, Valentina uses a database cache. You can change the size of the cache in the ‘Preferences’ dialog. Note that the size of the cache can not be more than half of the RAM available to Valentina.

■ Valentina’s “sort on fields” algorithm must have enough RAM in order to be successful. In particular it must have 2 * (4 * N) bytes, where N is the number of records in the Selection.

For example:

if N = 10’000 records then RAM for sorting is 80 KB
if N = 100’000 recordsthen RAM for sorting is 800 KB
if N = 1’000’000 records then RAM for sorting is 8 MB

Error numbers specific to Valentina

128	- Invalid file to import;
129	- Field is not specified;
130	- Selection is not specified;
131	- Wrong property for the field;
132	- Invalid date;
133	- Invalid time;
135	- Can not open a new version of database file

Other internal errors

```
kFBL_Error                    = 300,

kFBL_DataFileError            = 301,
kFBL_TableError               = 302,
kFBL_FieldError               = 303,
kFBL_IndexError               = 304,
kFBL_ParserError              = 305,

kFBL_CantOpenNewVersion       = 320,        // DataFile errors.
kFBL_CantReadDomain           = 321,

kFBL_FieldNotIndexed          = 340,        // Field errors.
kFBL_FieldIsConstant          = 341,
kFBL_FieldIsCounted           = 342,
kFBL_FieldIsComposed          = 343,
kFBL_FieldIsUnique            = 344,
kFBL_IndexNotSorted           = 345,

kFBL_TableIsEmpty             = 360,        // Table errors.
kFBL_TableIsCorrupted         = 361,

kFBL_SXCorrupted              = 370,        // Index errors.

kFBL_NeedValue                = 380,        // Query parser errors.
kFBL_NeedLeftBound            = 381,
kFBL_NeedRightBound           = 382
```