Damian Kaczmarek, Maciej Kamiński

RexIO Terminal Control Library 1.0

Comprehensive source code listing

for revision number 271

April 2, 2009
www.rexio.org

Contents

Note:

If possible, please refer to the newest version of this source code, availble from your software distributor. If your distribution doesn't include it, please consider changing it.

This source code is also availble from our website, and public SVN repository located at svn://67.207.133.55/rexio.

1 Build essential files

1.1 CMakeLists.txt

```
 1 if(COMMAND cmake_policy)
 2     cmake_policy(SET CMP0003 OLD)
 3 endif(COMMAND cmake_policy)
 4 project (TP2007)
 5
 6 include_directories (${TP2007_SOURCE_DIR}/include)
 7 set (SCREEN_SOURCE_DIR ${TP2007_SOURCE_DIR}/lib/screen)
 8 set (TOOLKIT_SOURCE_DIR ${TP2007_SOURCE_DIR}/lib/toolkit)
 9
10
11 include (FindLATEX)
12 add_custom_target (install make -f installer.mak install COMMENT
       Installing)
13 add_custom_target (uninstall make -f installer.mak uninstall COMMENT Un-
       Installing)
14 add_custom_target (doc make -C doc/ COMMENT Building documentation)
15 add_custom_target (cleandoc make -C doc/ clean COMMENT Cleaning
       documentation)
16
17 add_subdirectory (lib)
18 add_subdirectory (test)
19 add_subdirectory (extra/games/MUD)
```

1.2 installer.mak

```
 1 install:
 2     install lib/screen/librexio.so /usr/lib
 3     install lib/toolkit/librexiotk.so /usr/lib
 4     rm -fr /usr/include/rexio
 5     mkdir /usr/include/rexio
 6     install include/rexio/*.h++ /usr/include/rexio
 7     mkdir /usr/include/rexio/tk
 8     install include/rexio/tk/*.h++ /usr/include/rexio/tk
 9     ldconfig
10
11 uninstall:
12     rm -f /usr/lib/librexio.so
13     rm -f /usr/lib/librexiotk.so
14     rm -rf /usr/include/rexio
```

1.3 lib/CMakeLists.txt

```
1 add_subdirectory (screen)
2 add_subdirectory (toolkit)
3 #add_subdirectory (rcurses)
4 add_subdirectory (net)
```

1.4 lib/net/CMakeLists.txt

```
1 project (rexionet)
2
3
4 set (CMAKE_CXX_FLAGS "-ansi -Wall -pedantic -pedantic-errors -fPIC -std=c
      ++98 -Wno-long-long -rdynamic")
5 add_definitions(-DDO_VALIDATE_UTF_8_OUTPUT)
6
7 #link_libraries (-lboost_thread-mt  -lpthread -ldl)
8 add_library (rexionet SHARED netconn.c++)
```

1.5 lib/rcurses/CMakeLists.txt

```
1 project (rcurses)
2
3 set (CMAKE_CXX_FLAGS "-ansi -Wall -pedantic -pedantic-errors -fPIC -std=c
      ++98 -Wno-long-long")
4
5 include_directories (include)
6 include_directories (../../include)
7 include (src/CMakeLists.txt)
8
9 add_library (rcurses SHARED ${rcurses_sources})
```

1.6 lib/rcurses/src/CMakeLists.txt

```
1 set (rcurses_sources
2 ${rcurses_SOURCE_DIR}/src/rcurses.c++
3 )
```

1.7 lib/screen/CMakeLists.txt

```
1 project (screen)
2
```

```
 3
 4 set (CMAKE_CXX_FLAGS "-ansi -Wall -pedantic -pedantic-errors -fPIC -std=c
       ++98 -Wno-long-long -rdynamic")
 5 add_definitions(-DDO_VALIDATE_UTF_8_OUTPUT)
 6
 7 include_directories (include)
 8 include_directories (../../include/rexio)
 9 include (src/terminfo/CMakeLists.txt)
10 include (src/core/CMakeLists.txt)
11 include (src/real/CMakeLists.txt)
12 include (src/subscreen/CMakeLists.txt)
13 link_libraries (-lboost_thread-mt  -lpthread -ldl)
14 add_library (rexio SHARED ${terminfo_sources} ${core_sources} ${
       real_sources} ${subscreen_sources})
```

1.8 lib/screen/src/core/CMakeLists.txt

```
 1 set (core_sources
 2 ${screen_SOURCE_DIR}/src/core/bufferedinput.c++
 3 ${screen_SOURCE_DIR}/src/core/keyboard.c++
 4 ${screen_SOURCE_DIR}/src/core/commons.c++
 5 ${screen_SOURCE_DIR}/src/core/connection.c++
 6 ${screen_SOURCE_DIR}/src/core/displaystyle.c++
 7 ${screen_SOURCE_DIR}/src/core/screen.c++
 8 ${screen_SOURCE_DIR}/src/core/screenbase.c++
 9 ${screen_SOURCE_DIR}/src/core/screenbuffer.c++
10 ${screen_SOURCE_DIR}/src/core/utf8.c++
11 ${screen_SOURCE_DIR}/src/core/glyphwidth.c++
12 ${screen_SOURCE_DIR}/src/core/exception.c++
13 ${screen_SOURCE_DIR}/src/core/stacktrace.cpp
14 )
```

1.9 lib/screen/src/real/CMakeLists.txt

```
 1 set (real_sources
 2 ${screen_SOURCE_DIR}/src/real/terminal.c++
 3 ${screen_SOURCE_DIR}/src/real/screenconnection.c++
 4 ${screen_SOURCE_DIR}/src/real/terminfoenabled.c++
 5 ${screen_SOURCE_DIR}/src/real/genericscreen.c++
 6 ${screen_SOURCE_DIR}/src/real/localscreen.c++
 7 ${screen_SOURCE_DIR}/src/real/remotescreen.c++
 8 ${screen_SOURCE_DIR}/src/real/vt100compatible.c++
 9 )
```

1.10 lib/screen/src/subscreen/CMakeLists.txt

```
 1 set (subscreen_sources
```

```
2 ${screen_SOURCE_DIR}/src/subscreen/subscreen.c++
3 )
```

1.11 lib/screen/src/terminfo/CMakeLists.txt

```
1 set (terminfo_sources
2 ${screen_SOURCE_DIR}/src/terminfo/terminfokeymap.c++
3 ${screen_SOURCE_DIR}/src/terminfo/terminfocore.c++
4 ${screen_SOURCE_DIR}/src/terminfo/terminfoentry.c++
5 ${screen_SOURCE_DIR}/src/terminfo/terminfodatabase.c++
6 )
```

1.12 lib/toolkit/CMakeLists.txt

```
1 project (toolkit)
2
3 set (CMAKE_CXX_FLAGS "-ansi -Wall -pedantic -pedantic-errors -fPIC -std=c
      ++98 -Wno-long-long")
4
5 include_directories (include)
6 include_directories (../../include/)
7 include_directories (../../include/rexio)
8 include_directories (../../include/rexio/tk)
9 include (src/CMakeLists.txt)
10
11 add_library (rexiotk SHARED ${toolkit_sources})
```

1.13 lib/toolkit/src/CMakeLists.txt

```
1 set (toolkit_sources
2 ${toolkit_SOURCE_DIR}/src/widget.c++
3 ${toolkit_SOURCE_DIR}/src/rootwindow.c++
4 ${toolkit_SOURCE_DIR}/src/window.c++
5 ${toolkit_SOURCE_DIR}/src/widgetgroup.c++
6 ${toolkit_SOURCE_DIR}/src/boxgroup.c++
7 ${toolkit_SOURCE_DIR}/src/verticalgroup.c++
8 ${toolkit_SOURCE_DIR}/src/horizontalgroup.c++
9 ${toolkit_SOURCE_DIR}/src/label.c++
10 ${toolkit_SOURCE_DIR}/src/activewidget.c++
11 ${toolkit_SOURCE_DIR}/src/checkbox.c++
12 ${toolkit_SOURCE_DIR}/src/button.c++
13 ${toolkit_SOURCE_DIR}/src/inputbox.c++
14 ${toolkit_SOURCE_DIR}/src/framedwindow.c++
15 ${toolkit_SOURCE_DIR}/src/scrollbar.c++
16 ${toolkit_SOURCE_DIR}/src/selectbox.c++
17 ${toolkit_SOURCE_DIR}/src/stylesheet.c++
18 )
```

2 Header files of public interface

2.1 include/rexio/commons.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __COMMONS_H__
29 #define __COMMONS_H__
30
31 #include <tr1/memory>
32
33 #include <string>
34 #include <sstream>
35 #include <iostream>
36 #include <iomanip>
37 #include <rexio/trace.h++>
38
39 // boost implements new std. library functions
40 // - it may be useful if tr1/memory is not availble on specific platform
41 //
42 // #include<boost/shared_ptr.hpp>
43 //
44 // namespace std
45 // {
```

```
46 //  namespace tr1=boost;// alias to implementations of C++0x Technical
       Report 1
47 //                     // defined interfaces
48 // }
49
50 namespace Scr {
51     //! Machine specific unsigned integer. Type of at least 32 bits.
52     typedef unsigned long Uint;
53     //! Maximal value of Uint type
54     const Uint UintMax = -1;
55     //! Machine specific signed integer. Type of at least 32 bits.
56     typedef long Sint;
57     //! Maximal value of Sint type
58     const Uint SintMax = UintMax/2;
59     //! Minimal value of Sint type
60     const Uint SintMin = -SintMax-1;
61
62     typedef int               Sint32;
63     typedef unsigned int      Uint32;
64
65 /* ISO C++ does not define long long, however it is practical*/
66 # if defined __x86_64__
67     typedef unsigned long int Uint64;
68     typedef long int          Sint64;
69 # else
70     typedef
71     unsigned long long int    Uint64;
72     typedef long long int     Sint64;
73 # endif
74
75     //! vector container
76     struct Vector
77     {
78         /*!
79           \param _rows rows offset
80           \param _cols cols offset
81
82           Simple constructor for convenient initialization and
83           creation.
84         */
85         Vector(Sint _rows, Sint _cols);
86
87         //! offset in rows
88         Sint rows;
89         //! offset in columns
90         Sint cols;
91     };
92
93     //! size container
94     struct Size
95     {
96         /*!
97           \param _height height
98           \param _width width
99
100          Simple constructor for convenient initialization and
101          creation.
102        */
103        Size(Uint _height, Uint _width);
104
105        /*!
106          height property
```

```
107         */
108         Uint height;
109         /*!
110           width property
111         */
112         Uint width;
113     };
114
115     //! position container.
116     struct Position
117     {
118         /*!
119           \param _row row position
120           \param _col col position
121
122           Simple constructor for convenient initialization and
123           creation.
124         */
125         Position(Uint _row,Uint _col);
126         /*!
127           \param pos addition target
128
129           Simple addition.
130         */
131         Position operator+(const Position& pos);
132         /*!
133           \param size size to increment by
134
135          Result of incrementing position by a size of some object.
136         */
137         Position operator+(const Size& size);
138         /*!
139           \param vec vector to add
140
141           Result of modificating position by a vector.
142         */
143         Position operator+(const Vector& vec);
144         /*!
145           \param pos addition target
146
147           Simple assignment by addition.
148         */
149         Position& operator+=(const Position& pos);
150         /*!
151           \param size size to increment by
152
153           Incrementation of position by a size of some object.
154         */
155         Position& operator+=(const Size& size);
156         /*!
157           \param vec vector to add
158
159           Modification of position by a vector.
160         */
161         Position& operator+=(const Vector& vec);
162         /*!
163           \param pos substraction target
164
165           Simple substraction.
166         */
167         Position operator-(const Position& pos);
168         /*!
```

```
169           \param size size to decrement by
170
171          Result of decrementing position by a size of some object.
172         */
173         Position operator-(const Size& size);
174         /*!
175           \param vec vector to substract
176
177           Result of modificating position by a vector.
178         */
179         Position operator-(const Vector& vec);
180         /*!
181           \param pos substraction target
182
183           Simple assignment by substraction.
184         */
185         Position& operator-=(const Position& pos);
186         /*!
187           \param size size to decrement by
188
189           Decrementation of position by a size of some object.
190         */
191         Position& operator-=(const Size& size);
192         /*!
193           \param vec vector to substract
194
195           Modification of position by a vector.
196         */
197         Position& operator-=(const Vector& vec);
198
199
200         /*!
201           row number
202         */
203         Uint row;
204         /*!
205           column number
206         */
207         Uint col;
208     };
209
210     //! Standard comparison operator
211     inline bool operator!=(const Scr::Position & p1,const Scr::Position &
          p2)
212     {
213         return p1.col!=p2.col or p1.row!=p2.row;
214     }
215
216     //! Standard comparison operator
217     inline bool operator==(const Scr::Position & p1,const Scr::Position &
          p2)
218     {
219         return p1.col==p2.col and p1.row==p2.row;
220     }
221
222
223
224     //! \brief base class for exceptions thrown by library
225     //! objects.
226     /*!
227       exception holds message about conditions etc, where it was thrown
228      */
```

```
229     class Exception:  public std::exception
230     {
231     private:
232         /*!
233           message passed as reference counting pointer to prevent
234           resource waste during throw-catch sequence.
235         */
236         std::tr1::shared_ptr<std::string> message;
237     public:
238         /*!
239           \param _m message associated w/ exception. i.e. brief
240           description of situation. Will be displayed after program
241           failure.
242
243           Only argument is exception reason.
244         */
245         Exception(std::string _m)throw() ;
246         /*!
247           \param _base exception to copy (copy constructor is used
248           widely during throw-catch sequence.
249
250         */
251         Exception(const Exception& _base)throw();//copy ctor.
252
253         /*!
254           what() derrivated from std::exception: informs on reason of
                exception
255         */
256         virtual const char* what() const throw();
257         /*!
258           destructor conditionally frees resources (thanks to smart
259           pointer used).
260         */
261         virtual ~Exception()throw() ;
262     };
263
264
265 /*!
266   Macro to simplify defining of exception hierarchy. To define insitu
267   new exception just add __DE(MoreSpecificException,MoreGeneralOne)
268  */
269 #define __DE(CHILD,PARENT) class CHILD: public PARENT   \
270     {public:                                            \
271         CHILD(const CHILD & _b)throw():PARENT(_b){;}    \
272         CHILD(std::string _m)throw():PARENT(_m){;}      \
273     }
274
275     // always outside of throw specification, always causes abort,
276     // and therefore may be useful while you want to explicitly stop
277     // application on certain condition
278     __DE(FatalException,Exception);
279
280     // also outside of throw spec., Used in assertion: exception may be
281     // thrown only if programmer is mistaken.
282     __DE(LogicError,Exception);
283 }
284
285 #include <rexio/glyphwidth.h++>
286
287 #endif // __COMMONS_H__
```

2.2 include/rexio/glyphwidth.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __GLYPHWIDTH_H__
29 #define __GLYPHWIDTH_H__
30
31 #include <wchar.h>
32 #include <rexio/commons.h++>
33 #include <bitset>
34
35 namespace Scr {
36
37
38     /*! Singleton class. */
39     class GlyphWidth {
40 /*!
41   Bitset for caching the width results. 2 bits per glyph.
42   Note: the bitset gets reasonably fast only in the Release build
43 */
44         static std::bitset<(1<<17)*2> glyphWidth;
45         GlyphWidth();
46     public:
47         __DE(NotPrintable, Exception);
48
49         /*!
50           \arg ch
51
52           \return width of unicode character (0 or 1 or 2), that means
                number of
53           cells in console, it needs to fit.
54         */
55         static Uint Get(wchar_t ch) {
56             static GlyphWidth g;
57
58             if((Uint)ch >= glyphWidth.size()/2) {
59                 // covers the > 17 bits 2 column with.
```

```
60                 if((ch >= 0x20000 && ch <= 0x2fffd) ||
61                    (ch >= 0x30000 && ch <= 0x3fffd))
62                     return 2;
63                 /* the following condition is a condition for 0 width
64                    characters, taken from the wcwidth.c and containing
                          only ranges
65                    above 17 bits. */
66                 else if( ( ch >= 0xE0001 && ch <= 0xE0001 ) || ( ch >= 0
                       xE0020 && ch <= 0xE007F ) ||
67                            ( ch >= 0xE0100 && ch <= 0xE01EF ))
68                     return 0;
69                 else
70                     return 1;
71             }
72
73             // one lookup for 1 width characters, 2 for the others
74             return glyphWidth[(ch<<1)] ? 1:2*glyphWidth[(ch<<1) + 1];
75         }
76     };
77
78     /*!
79      * Computes width of unicode character (0 or 1 or 2), that means
            number of
80      * cells in console, it needs to fit. Furthermore, it returns -1 if a
            character
81      * is a non-printable one.
82      */
83     inline unsigned long width(wchar_t c)
84     {
85         return GlyphWidth::Get(c);
86     }
87     //__attribute__ ((deprecated));
88 }
89
90 #endif // __GLYPHWIDTH_H__
```

2.3 include/rexio/keyboard.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
```

```
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __KEYBOARD_H__
29 #define __KEYBOARD_H__
30 #include "commons.h++"
31 namespace Scr
32 {
33     /*!
34       \brief Class represents key (or key combination) pressed on client
35       terminal.
36      */
37     class Key
38     {
39     private:
40         //Alt+letter or Esc,than letter
41         static const Uint     altMask = 0x00400000;
42
43         //Control+letter
44         static const Uint controlMask = 0x00200000;
45
46         //!special key pressed
47         static const Uint specialMask = 0x56000000;
48
49         //basic key mask
50         static const Uint basicKeyMask= 0x0000007f;
51         Uint key;
52     public:
53
54         /*!
55          * Special ascii keys as teletype mnemonics. Please note, that
                this
56          * enum is temporary, and will be deprecated in 1.1
57          *
58          */
59         enum ASCII
60         {
61             LF   = 0xa,
62             CR   = 0xd,
63             EoF  = 0x4
64         };
65
66         /*!
67          * Special key names. May be used for comparizons against key
                object
68          * (please refer to handbook for use example)
69          *
70          */
71         enum Special
72         {
73             Escape  = 0x1b,
74             Tab     = 0x561f0000, // on most systems: shift+tab
75             BackTab,
76             F1,
77             F2,
78             F3,
79             F4,
```

```
 80
 81             F5,
 82             F6,
 83             F7,
 84             F8,
 85
 86             F9,
 87             F10,
 88             F11,
 89             F12,
 90 //shifted
 91             F13,
 92             F14,
 93             F15,
 94             F16,
 95
 96             F17,
 97             F18,
 98             F19,
 99             F20,
100
101             F21,
102             F22,
103             F23,
104             F24,
105 //??
106             F25,
107             F26,
108             F27,
109             F28,
110
111             F29,
112             F30,
113             F31,
114             F32,
115
116             F33,
117             F34,
118             F35,
119             F36,
120
121             Up,
122             Down,
123             Right,
124             Left,
125
126             CtrlUp,
127             CtrlDown,
128             CtrlRight,
129             CtrlLeft,
130
131             Enter,
132             Delete,
133             Backspace,
134
135             Insert,
136             Home,
137             PageUp,
138             PageDown,
139             End
140         };
141
```

```
142         __DE(NotABasicKey,Exception);
143         __DE(NotASpecialKey,Exception);
144 
145         Key():key(0){;}
146 
147         /*!
148          * \param key unsigned integer form
149          *
150          * This constructor allows implicit conversion between two forms
                of key
151          */
152         Key(Uint key)throw()
153             {
154                 this->key = key;
155             }
156 
157         /*!
158          * implicit or
159          * static cast operator
160          */
161         operator Uint()throw()
162             {
163                 return key;
164             }
165 
166         /*!
167          * If represents plain ascii character, function returns true.
168          * false is returned otherwise
169          */
170         bool IsABasicKey()throw()
171             {
172                 return
173                     ( key >= ' ' )
174                     && ( (key&0xff) <=127)
175                     && ((key & specialMask) != specialMask);
176             }
177 
178         /*
179          * Function returns true if key is a special key (that means
                return,
180          * one of arrows, function key, delete etc.
181          */
182         bool IsASpecialKey()throw()
183             {
184                 return  (key < ' ') or
185                         ((key & specialMask) == specialMask);
186             }
187 
188         /*!
189          * as if it was a letter A-Z
190          */
191         char GetBasicKey()throw(NotABasicKey);
192 
193         //!
194         Special GetSpecialKey()throw(NotASpecialKey);
195 
196         const char * GetKeyName()throw();
197     };
198 }
199 
200 #endif
```

2.4 include/rexio/net.h++

```
1 #ifndef __NETCONN_H__
2 #define __NETCONN_H__
3 #include <iostream>
4 #include <rexio/tk/rootwindow.h++>
5 namespace RexIO { namespace Networking {
6     /*!
7      * Virtual base for server implementation has almost all code needed
           to
8      * operate as RexIO server. This class facilitates thread management,
           window
9      * creation and so on.
10     * \note this class is not guaranteed to be thread safe. it uses some
           global
11     * data structures, and was not designed with many RexIO servers
           operated
12     * within one process, so please avoid id
13     */
14 class ServerImpl {
15     // some of internal classes
16     friend class __Connection;
17 private:
18     // used by stop
19     bool active;
20     static void starter(Scr::Tk::RootWindow * w);
21 protected:
22     virtual Scr::Tk::RootWindow *
23     GenWindow(std::istream & in, std::ostream & out) = 0;
24 public:
25     //! default constructor
26     ServerImpl();
27     //! start listening on specified port number
28     //! \param portnum port number
29     void Start(int portnum);
30     //! end listening, send "terminate" messages to all clients. Then end.
31     //! \note this function is not guaranteed to succeed: if any thread is
32     //! enters infinite loop, this function will wait until kill -9.
33     void Stop();
34 };
35
36 //! templatized version of ServerImpl (WIN parameter is class derivated
       from
37 //! RootWindow
38 template < typename WIN >
39 class Server : public ServerImpl {
40 protected:
41
42     Scr::Tk::RootWindow * GenWindow(std::istream & in, std::ostream & out)
           {
43         return new WIN(in, out);
44     }
45 };
46 }}
47 #endif
```

2.5 include/rexio/screen.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __SCREEN_H__
29 #define __SCREEN_H__
30
31 #include <rexio/commons.h++>
32 #include <rexio/keyboard.h++>
33
34 //! Namespace of lower half of the library
35 /*!
36   This namespace contains classes and other utilities connected with
37   general purpose text screen manipulation and input processing. It
38   implements platform independent Screen class, and Connection class
39   representing basic framework for console application development.
40   \note Scr::Tk is upper part of the library, and is recommended for
41   all higher level UI manipulation, but Scr::Connection can be used
42   alone
43
44   \latexonly
45
46   Following figure is
47   simplified class relationship diagram for Scr::Screen and
48   Scr::Connection connected items, focusing on internal layout of
49   implementation of screen.
50
51   \begin{figure}[H]
52   \begin{center}
53   \leavevmode\includegraphics[width=400pt]{../Basic_class_structure_uml}
54   \end{center}
55   \end{figure}
56
57   \endlatexonly
58 */
59 namespace Scr
60 {
61     //! Foreground colors and styles.
62     namespace Fg
```

```
63      {
64          //! Color itself. 8 basic colours + 2 special (Fg::System, Fg::
                Transparent)
65          enum Color
66          {
67              //!  special colour represents default colour of system
68              //!  (for some terminals and terminal emulator this may
69              //!  differ from 8 basic colors)
70              System      = 0,
71              //!  special colour represents colour of
72              //! just-replaced character
73              Transparent = 1,
74              Black       = 30,//!< color 1
75              Red         = 31,//!< color 2
76              Green       = 32,//!< color 3
77              Yellow      = 33,//!<color 4
78              Blue        = 34,//!<color 5
79              Magenta     = 35,//!<color 6
80              Cyan        = 36,//!<color 7
81              White       = 37 //!<color  8
82          };
83          //!  foreground styles
84          enum Style
85          {
86              Dark   = 0,
87              Bright = 1
88          };// in future maybe also UnderlineOn and UnderlineOff... etc.
89      }
90      //! Background colors. WITHOUT style.
91      namespace Bg
92      {
93          //! background colours enumeration
94          enum Color
95          {
96              //!  special colour represents default colour of system
97              //!  (for some terminals and terminal emulator this may
98              //!  differ from 8 basic colors)
99              System      = 0,
100             //!  Set colour of just-replaced text
101             Transparent = 1,
102             Black       = 40,//!< color 1
103             Red         = 41,//!< color 2
104             Green       = 42,//!< color 3
105             Yellow      = 43,//!< color 4
106             Blue        = 44,//!< color 5
107             Magenta     = 45,//!< color 6
108             Cyan        = 46,//!< color 7
109             White       = 47 //!< color 8
110         };
111     }
112
113     //! \brief complete set of display properties for
114     //! single character
115     class DisplayStyle
116     {
117     private:
118
119         /*!
120          * style described as an union
121          */
122         union
123         {
```

```
124             /*!
125              * As single unsigned integer - for easy copying
126              */
127             Uint32 style;
128 
129             /*!
130              * And as a set of three separate variables, for easy
                     manipulation
131              */
132             struct
133             {
134                 unsigned char fgColor; //!< foreground color
135                 unsigned char fgStyle; //!< foreground style
136                 unsigned char bgColor; //!< background color
137             }properties;
138         };
139     public:
140         /*!
141           Set up specified style (parametrized constructor)
142           \param _fgColor
143           \param _fgStyle
144           \param _bgColor
145         */
146         DisplayStyle(Fg::Color _fgColor,
147                      Fg::Style _fgStyle,
148                      Bg::Color _bgColor)throw();
149 
150         /*!
151           \param s - source to copy
152 
153           basic copy constructor - binary 1:1 copy.
154          */
155         DisplayStyle(const DisplayStyle & s)throw();
156         /*!
157           Nonparameter constructor sets colours default. default
158           values are implementation-specific (currently white on
159           green, but this may vary - maybe once upon the time we will
160           implement some special "undefined" values for all three
161           members of this class);
162         */
163         DisplayStyle()throw();
164 
165 
166         /*!
167          * \return foreground color
168          */
169         Fg::Color GetFgColor()const throw()
170             {
171                 return static_cast<Fg::Color>(properties.fgColor);
172             }
173 
174         /*!
175          * \return foreground style
176          */
177         Fg::Style GetFgStyle()const throw()
178             {
179                 return static_cast<Fg::Style>(properties.fgStyle);
180             }
181 
182         /*!
183          * \return nackground color
184          */
```

```
185         Bg::Color GetBgColor()const throw()
186             {
187                 return static_cast<Bg::Color>(properties.bgColor);
188             }
189 
190         /*!
191          * Set foreground color
192          *
193          * \param col new color
194          */
195         void SetFgColor( const Fg::Color col )throw()
196             {
197                 properties.fgColor=static_cast<unsigned char>(col);
198             }
199 
200         /*!
201          * Set foreground style
202          *
203          * \param s new style
204          */
205         void SetFgStyle( const Fg::Style  s  )throw()
206             {
207                 properties.fgStyle=static_cast<unsigned char>(s);
208             }
209 
210         /*!
211          * Set background color
212          *
213          * \param col new color
214          */
215         void SetBgColor( const Bg::Color col )throw()
216             {
217                 properties.bgColor=static_cast<unsigned char>(col);
218             }
219 
220 
221 
222 
223         /*!
224           \param other
225           (fgColor==other.fgColor) && (fgStyle==other.fgStyle) && (bgColor
                ==other.bgColor)
226         */
227         bool operator==(const DisplayStyle & other)
228             {
229                 return style==other.style;
230             }
231 
232         /*!
233           \param other
234           (fgColor!=other.fgColor) || (fgStyle!=other.fgStyle) || (bgColor
                !=other.bgColor)
235         */
236         bool operator!=(const DisplayStyle & other)
237             {
238                 return style!=other.style;
239             }
240 
241         /*!
242          * \param other style, whose content will be assigned to this
243          *
244          * Copy-assignment operator
```

```
245          */
246         Scr::DisplayStyle& operator=(const DisplayStyle & other)
247             {
248                 style=other.style;
249                 return *this;
250             }
251 
252     };
253 
254 
255 
256     //! \brief Class representing basic output operation is defined as ABC
257     //! (abstract base)
258     /*!
259       Operations are performed using subroutines apropriate to output
260       type. Note, that some implementations of Screen (i.e. remote
261       ones) use spcific forms of compression to limit data transfer,
262       other rather optimize CPU usage.
263     */
264     class Screen
265     {
266     private:
267     protected:
268         Screen()throw();
269     public://exceptions may be thrown
270 
271         __DE(ConnectionError,Exception); // when failed to update
272         // screen due to communication interrupt or missing client
                terminal
273         // capabilities
274 
275         __DE(TerminalTypeUnknown,ConnectionError);// client terminal is
                too primitive
276         // to support request.
277         __DE(CursorVisibilityNotSupported,Exception);
278 
279         __DE(IllegalOperation,Exception);// each illegal operation
280 
281         __DE(RangeError,IllegalOperation);// range checking
282         // failed
283 
284         __DE(IllegalCharacter,IllegalOperation);
285 
286         __DE(InvalidUTF8,IllegalCharacter);//any encoding, that
287         //does not pass UTF-8
288         //validation.
289 
290         __DE(InvalidFirstByte,InvalidUTF8);//doesn't match any of standard
291         //patterns (that is it doesn't match any of utf-8 sequences)
292 
293         __DE(InvalidTrailingByte,InvalidUTF8);
294 
295         __DE(OverlongUTF8Encoding,InvalidUTF8);// too many bytes
296         // used to encode
297         // specific unicode
298         // character.
299 
300         __DE(CharacterExceedsUTF8Range,IllegalCharacter);// char
301         //code equal to or greater than 1<<21
302 
303         __DE(GotoOutOfRange,RangeError);//gotoYX illegal location
304 
```

```
305         __DE(SubscreenOutOfRange,RangeError);// too big subscreen
306
307         __DE(PrintOutOfRange,RangeError);//printing where
308         //illegal (out of array
309         //boundaries)
310
311         __DE(PrintOutOfHorizontalRange,PrintOutOfRange);
312         __DE(PrintOutOfVerticalRange,PrintOutOfRange);
313
314         __DE(SubscreenResize,IllegalOperation);//subscreen can not
315         //be resized
316         //__DE()
317
318         //(does not move active point, fills screen
319         //withactive bg color)
320         /*!
321           Fill whole screen with current background colour.
322
323           \note function does not operate on physical screen. Use
324           Refresh to see effect.
325         */
326         virtual void Clear()throw()                = 0;
327
328         // following 3 functions sets active properties (used while
329         // inserting new characters or clearing screen)
330
331         /*!
332           \param col new background colour to be set
333           \return  nothing upon successful execution
334
335           Function sets background colour. Background colour is of
336           type Bg::Color. Typical use example: <code>
337           myscreen.SetBgColor(Bg::Black) </code>.
338
339           Function is exception safe as it does not throw any exceptions.
340
341           \note thanks to overloaded operator <<,
342           something like <code> myscreen << Bg::Black </code>will also be
                valid and
343           will do exactly the same as above.
344         */
345         virtual void SetBgColor(Bg::Color col)
346             throw()                                = 0;
347
348         /*!
349           \param col new foreground colour to be set
350           \return  nothing upon successful execution
351
352           Function sets foreground colour. Background colour is of
353           type Bg::Color. Typical use example: <code>myscreen.SetFgColor(
                Fg::Red)</code>.
354
355           Function is exception safe as it does not throw any
356           exceptions.
357
358           \note thanks to overloaded operator <<,
359           something like <code>myscreen << Fg::Red</code> will also be
                valid and
360           will do exactly the same as above.
361
362           \note colour is not only foreground property: Fg
363           style sets bright or dark variant of each colour, and it
```

```
364           doubles total number of availble colours.
365         */
366         virtual void SetFgColor(Fg::Color col)
367             throw()                                 = 0;
368
369         /*!
370           \param s new foreground text style to be set
371           \return  nothing upon successful execution
372
373
374           Set foreground style (i.e. bright (bold) or dim
375           (regular)). Maybe once upon the time more styles will be
376           suppotred to utilise capabilities of more advanced terminal
377           types (such as blink and underline for DEC VT220), but for
378           now we don't specify this, as portability is one of primary
                  goals
379           for our library
380
381           Function is exception safe as it does not throw any
382           exceptions.
383         */
384         virtual void SetFgStyle(Fg::Style s)throw()= 0;
385
386         // move active point
387         /*!
388           \param y
389           \param x  new coordinates of active point (please
390           remember the order of theese attributes)
391
392           Change active point position (that is point, where writing
393           will start after invocation of AddText or AddCharacter.
394
395           Function throws exception
396           Scr::Screen::GotoOutOfRange when coordinates
397           exceed size of screen. After exception throw active
398           position is undefined.
399
400           \sa SetFgColor # SetBgColor
401           \return  nothing upon successful execution
402         */
403         virtual void GotoYX(Uint y, Uint x)
404             throw(GotoOutOfRange)      = 0;
405
406         /*!
407           \param c character to be printed
408           \return  nothing upon successful execution
409
410           Print single low ascii character (for characters out of
411           basic 7-bit ascii set please use integer version of this
412           function and proper UNICODE codes of characters)
413
414           \exception Scr::Screen::PrintOutOfRange  as for
415           AddText
416
417           \exception Scr::Screen::IllegalCharacter
418           negative signed (or over-127-unsigned) c supplied.
419         */
420         virtual void AddCharacter(char c)// add single LOW ascii character
421             throw(PrintOutOfRange,// (128-255 ascii codes
422                   IllegalCharacter)     = 0;//are forbidden)
423
424         /*!
```

```
425           \param c character to be printed
426           \return  nothing upon successful execution
427
428           Print single unicode character.
429
430           \note what programmes supply as parameter is direct number
431           of character, not UTF-8 encoded version of it. UTF-8 may be
432           supplied using  AddText
433
434           \exception Scr::Screen::PrintOutOfRange  as for
435           AddText
436
437
438           \exception Scr::Screen::IllegalCharacter too
439           large value of c.
440         */
441         virtual void AddCharacter(wchar_t c) // add single unicode
442             throw(PrintOutOfRange,     //character
443                   IllegalCharacter)     = 0;
444
445         //print something (need Refresh to see it), starting from
446         //active point. move active point AFTER newly added text.
447         /*!
448           \param text traditional null-terminated string in UTF-8 encoding
                .
449           \return  nothing upon successful execution
450
451           Adds specified text in position starting from active point
452           (see GotoYX). Moves active point just after the newly added
453           text irrespectively if this position is valid (so next text
454           will start just after it, always in the same line). Function
455           does not support line breaks.
456
457           As function supports UTF-8, it also requires string
458           to be valid UTF-8, so each character must be low ascii
459           (1-127) or multibyte.
460
461           \note function will not emit text to physical screen, unless
462           Refresh called afterwards
463
464           \exception Scr::Screen::PrintOutOfRange is thrown if
465           initial position of active point is invalid, or if text is
466           too long (as function
467           does not support line breaks).<BR>
468           If the text ends exactly in last column of screen, active
469           point is set after it, in the same line, so is invalid, and
470           next trial of usage of this function (or any other
471           character-adding one) will fail with
472           Scr::Screen::PrintOutOfRange.
473
474
475           \exception  Scr::Screen::IllegalCharacter will be
476           thrown if text supplied is not a valid UTF-8 string (even
477           "overlong sequences" will be considered illegal (according
478           to an apropriate RFC
479
480           \sa AddCharacter, Refresh, RFC 3629
481           \latexonly \index{RFC, reference to!3629}\endlatexonly
482         */
483         virtual void AddText(const char * text)
484             throw(PrintOutOfRange,
485                   IllegalCharacter)     = 0;
```

```
486
487
488
489         /*!
490          * \param text as above but as std::string, not C-style string
491          *
492          * exceptions: as above.
493          */
494         virtual void AddText(const std::string & text)
495             throw(PrintOutOfRange,
496                   IllegalCharacter)     = 0;
497
498         /*!
499          * \param text wide UNICODE string  to be printed
500          *
501          * \exception PrintOutOfRange is thrown if
502          * initial position of active point is invalid, or if text is
503          * too long (as function does not support line breaks).
504          *
505          * \exception IllegalCharacter will be
506          * thrown if text supplied is not a valid UTF-8 string (even
507          * "overlong sequences" will be considered illegal (according
508          * to an apropriate RFC
509          *
510          *  \copydoc AddText(const std::wstring & text)
511          */
512         virtual void AddText(const wchar_t * text)
513             throw(PrintOutOfRange,
514                   IllegalCharacter)     = 0;
515
516         /*!
517          *  \param text text to be printed
518          *
519          *  \sa Screen::AddText(const char * text) for extensive
                 description
520          */
521         virtual void AddText(const std::wstring & text)
522             throw(PrintOutOfRange,
523                   IllegalCharacter)     = 0;
524
525         /*!
526           \param text wide string
527           \param limitcols max width in columns
528
529           Function prints AT MOST limitcols wide string. Width means
530           number of columns, which is not the same thing as number of
531           characters, as most CJK glyphs are multicolumn.
532
533           \exception PrintOutOfRange is thrown if
534           initial position of active point is invalid, or if text is
535           too long (as function does not support line breaks).
536
537           \exception IllegalCharacter will be
538           thrown if text supplied is not a valid UTF-8 string (even
539           "overlong sequences" will be considered illegal (according
540           to an apropriate RFC
541
542           \sa Screen::AddText(const char * text) for extensive description
543          */
544         virtual Uint AddTextCols(const wchar_t * text, Uint limitcols)
545             throw(PrintOutOfRange,
546                   IllegalCharacter)     = 0;
```

```
547
548         /*!
549           \param text wide string
550           \param limitcols max width in columns
551
552           Function prints AT MOST limitcols wide string. Width means
553           number of columns, which is not the same thing as number of
554           characters, as most CJK glyphs are multicolumn.
555          */
556         virtual Uint AddTextCols(const std::wstring & text, Uint limitcols
                )
557             throw(PrintOutOfRange,
558                   IllegalCharacter)     = 0;
559
560         /*!
561           \param c ASCII character
562           \param n number of repetitions (length of line)
563
564           Function adds horizontal line of n characters c.
565          */
566         virtual void HorizontalLine(char c, Uint n)
567             throw(PrintOutOfRange, IllegalCharacter) = 0;
568
569         /*!
570           \param c UNICODE character
571           \param n number of repetitions (length of line)
572
573           Function adds horizontal line of n characters c.
574          */
575         virtual void HorizontalLine(wchar_t c, Uint n)
576             throw(PrintOutOfRange,
577                   IllegalCharacter)     = 0;
578
579         /*!
580           \param c ASCII character
581           \param n number of repetitions (length of line)
582
583           Function adds verticel line of n characters c.
584          */
585         virtual void VerticalLine(char c, Uint n)
586             throw(PrintOutOfRange,
587                   IllegalCharacter)     = 0;
588
589         /*!
590           \param c UNICODE character
591           \param n number of repetitions (length of line)
592
593           Function adds vertical line of n characters c.
594          */
595         virtual void VerticalLine(wchar_t c, Uint n)
596             throw(PrintOutOfRange,
597                   IllegalCharacter)     = 0;
598
599         /*!
600          * \param c character used to create rectangle
601          * \param s dimensions of rectangle
602          *
603          * Function creates rectangle of characters. s specifies
604          * number of rows and number of repetitions of character
605          * c in each row.
606          */
607         virtual void Rectangle(char c, const Size & s)
```

```
608             throw(PrintOutOfRange,
609                   IllegalCharacter)    = 0;
610
611         /*!
612          * \param c character used to create rectangle
613          * \param s dimensions of rectangle
614          *
615          * Function creates rectangle of characters. s specifies
616          * number of rows and number of repetitions of character
617          * c in each row.
618          */
619         virtual void Rectangle(wchar_t c, const Size & s)
620             throw(PrintOutOfRange,
621                   IllegalCharacter)    = 0;
622
623         //change screen size
624         /*!
625           \param rows new number of rows (new height) of screen
626           \param cols new number of columns of screen
627           \return  nothing upon successful execution
628
629           Change the output size.
630
631           \note this function does not change size of physical screen,
632           it may only be used to resize this object to fit physical
633           screen size. If screen grows, new characters are filled with
634           current background colour.
635           \note Function does not refresh the physical screen after
636           it's resizing, so it's content is undefined after call of
637           this function (however left-top part of it will be restored
638           after Refresh call).
639
640           \exception Scr::Screen::Exception::IllegalOperation if
641           particular screen may nor be resized for some
642           implementation- or platform- specific reasons. In particular
643           case primitive subscreens may not be resized
644           (SubscreenResize specialization of exception is thrown then).
645
646
647         */
648         virtual void Resize(Uint rows, Uint cols)
649             throw(IllegalOperation)   = 0;
650
651         /*!
652           \param p new cursor position
653
654           Force cursor position after finishing next refresh. If *this
655           is a subscreen, position (relative to *this) will be mapped
656           to the physical screen.
657
658           \note effective position after refresh will be position
659           set by last successful call to ForceCursorPosition
660          */
661         virtual void ForceCursorPosition(Position p)throw(RangeError) = 0;
662
663
664         /*!
665           make cursor invisible
666          */
667         virtual void HideCursor()throw(CursorVisibilityNotSupported) = 0;
668
669         /*!
```

```
670           make it visible again
671          */
672         virtual void ShowCursor()throw(CursorVisibilityNotSupported) = 0;
673
674         //update real screen
675         /*!
676           \return  nothing upon successful execution
677
678           Rewrite internal buffers to physical screen. When writing
679           complex, multi-layer items, it is
680           recommended to call this function after finishing writing
681           everything. When small changes need to be displayed, it may
682           be called every single AddCharacter, as it can't be a very
683           expansive operation in terms of CPU or transfer usage
684           (remote implementations will be optimized for transfer,
685           while local will be writen to achieve best performance for
686           specific terminal).
687         */
688         virtual void Refresh()
689             throw(ConnectionError)    = 0;
690
691         /*!
692           \param _y_offset vertical offset from top edge of this
693           screen to top edge of new SubScreen.
694           \param _x_offset horizontal offser
695           \param _h height
696           \param _w with
697           \return  pointer to new SubScreen (programmer will have to free
                  it's
698           resources to prevent memory leak and other errors).
699
700           \exception Scr::Screen::SubscreenOutOfRange
701           is thrown when too big subscreen requested or inapropriate
702           position specified
703         */
704         virtual Screen *
705         CreateSubScreen(Uint _y_offset,
706                         Uint _x_offset, Uint _h,
707                         Uint _w)throw(SubscreenOutOfRange) = 0;
708
709
710
711         //!
712         //!\return current type of terminal
713         //!
714         virtual const char * GetType() const throw(TerminalTypeUnknown) =
              0;
715
716
717         //retrieve settings
718         /*!
719           \return vertical offset of active point
720         */
721         virtual Uint GetY() const throw()= 0;
722         /*!
723           \return horizontal offset of active point
724         */
725         virtual Uint GetX() const throw()= 0;
726         /*!
727           \return Current height of screen
728         */
729         virtual Uint GetHeight() const throw()= 0;
```

```
730         /*!
731           \return Current width of screen
732         */
733         virtual Uint GetWidth() const throw()= 0;
734 
735         /*!
736            \return true if cursor is visible, false if it ishidden
737            \sa ShowCursor HideCursor
738          */
739         virtual bool GetCursorVisibility() const throw()= 0;
740 
741         virtual ~Screen()throw();
742 
743     };
744 
745     //! Class representing basic input and output operations
746     /*!
747       Class is implemented and designed as base class for any specific
748       application. It controls directly screen size, and specifies
749       event interface for reacting keyboard and screen connected
750       events. It is designed to be platform-transparent, so programmer
751       does not have to bother OS specific method of checking window
752       size, key value etc.
753 
754       OnEvent actions are defined as virtual member functions
755     */
756     class Connection
757     {
758     private:
759     protected:
760         std::auto_ptr<Screen> screen;
761         Connection(std::istream & _input, std::ostream & _output)throw();
762     public:
763 
764         __DE(StartFailed,Exception);
765         __DE(StopFailed,Exception);
766         typedef StopFailed ExitFailed; // type name alias (the
767                                        // same exception)
768         __DE(AlreadyStopped,StopFailed);
769         __DE(Broken,StartFailed);
770 
771         __DE(IllegalClientAction,StartFailed);
772 
773         __DE(IllegalTelnetAction,IllegalClientAction);
774 
775         __DE(IncorrectSubnegotiation,IllegalTelnetAction);
776         __DE(IncorrectWindowSizeSubnegotiation,IncorrectSubnegotiation);
777         __DE(IncorrectTerminalTypeSubnegotiation,IncorrectSubnegotiation);
778 
779         __DE(UnsupportedClientFeature,IllegalClientAction);
780         __DE(UnsupportedKey,UnsupportedClientFeature);
781         __DE(UnsupportedTelnetFeature,UnsupportedClientFeature);
782 
783         __DE(AlreadyRunning,StartFailed);
784         __DE(NotYetStarted,StopFailed);
785         /*!
786           \return result of whole connection. If broken, the result is
787           1. Else the result is argument passed to Exit(int)
788 
789 
790           Start connection (with no arguments - they must be set with
791           application specific methods defined by
```

```
792           programmer). Function blocks execution of thread up to
793           finish of connection.
794
795           \exception Scr::Connection::AlreadyRunning when connection has
796           already been started (one execution thread per class
797           instance allowed) and hasn't yet been stopped.
798           \exception Scr::Connection::Broken is thrown
799           when connection is broken (i.e. input/output error occured)
800
801           \exception Scr::Connection::FailedToStart when
802           connecction can not be estabilished for some reason.
803           \note as Start() is defined in way, that allows it to throw
804           only one exception class and all OnEvent functions do not
805           allow any exceptions, all of them must be handled within
806           exception handling function. Unexpected exception handler
807           will be used otherwise.
808         */
809         virtual int Start()
810             throw(StartFailed,Screen::IllegalCharacter); // nonvirtual;
811
812         /*!
813           \param argc number of arguments
814           \param argv C-style array of arguments
815
816           Start connection. \a argv can be parsed in inheritting classes.
817
818           \sa \a Start() for detailed info
819         */
820         virtual int Start(int argc, char **argv)
821             throw(StartFailed,Screen::IllegalCharacter); // nonvirtual;
822
823         /*!
824           \return nothing
825           \param code this will be the result of ongoing Start()
826
827           If connection is currently running (that means, Start()
828           member function of specific object is running) Exit tells it
829           to break as soon as possible, call OnExit() and return code
830           given.
831
832           \exception Scr::Connection::AlreadyStopped
833           exception is thrown when  Exit was already called, but
834           connection wasn't stopped yet.
835           \exception Scr::Connection::NotYetStarted is
836           thrown when connection was already stopped or hasn't yet
837           been started.
838         */
839         void Exit(int code)throw(StopFailed); // finishes connection
840         // Start() returns code passed to Exit() or 1 if other cause
841         // of finishing connection
842
843
844
845         //event "callback" functions
846         virtual void OnStart()throw();
847         virtual void OnResize(Uint rows, Uint cols)
848             throw();//              new height           new width
849         virtual void OnKeyDown(Key key) // ascii code or
850             throw();                                // special code.
851         //!
852         //! \param code exit code. Will be returned by Start just
853         //! after finish of app.
```

```
854         //!
855         virtual void OnExit(int code)throw(); // last actions of program
856         //(called by destructor; while OnExit called, screen may not
857         //be used (it is already destroyed)
858         virtual ~Connection()throw();
859
860         friend class Screen;
861         //friend class LocalScreen;
862     };//class Connection
863
864     //! namespace containing iomanipulator-like items
865     namespace Control
866     {
867
868         /*!
869           This is ,,private'' class of system. It is only designed as
870           a return type of Scr::Control::GotoYX(Uint , Uint) -
871           simmilar idea to std::_Setw (as return type of
872           std::setw(int)).
873         */
874         class _PositionYX:public Position
875         {
876         public:
877             _PositionYX(Uint _row,Uint _col):
878                 Position(_row,_col){;}
879         };
880
881         /*!
882           \param _y row on screen
883           \param _x column on screen
884
885           Controlling screen active point position (the point, where
886           text starts).
887           FooScreen << Scr::Control::GotoYX(3,4) is an
888           direct equivalent of FooScreen.GotoYX(3,4).
889         */
890         _PositionYX GotoYX(Uint _y, Uint _x);
891
892
893         /*!
894           Special one-element type introduced only for Refresh manipulator
895         */
896         enum _Refresh {
897             /*!
898               This manipulator forces refreshing of screen.
899               FooScreen << Scr::Control::Refresh is an
900               direct equivalent of FooScreen.Refresh().
901             */
902             Refresh
903         };
904
905         /*!
906           Special one-element type introduced only for Clear manipulator
907         */
908         enum _Clear {
909             /*!
910               This manipulator clears whole screen.
911               FooScreen << Scr::Control::Clear is an
912               direct equivalent of FooScreen.Clear().
913             */
914             Clear
915         };
```

```
916
917     } // namespace Control
918
919
920
921 //  Screen& operator<<(Screen & screen,const EString & whatto);
922     Screen& operator<<(Screen & screen,const std::wstring & whatto);
923     Screen& operator<<(Screen & screen,wchar_t const * const & whatto);
924     Screen& operator<<(Screen & screen,wchar_t * const & whatto);
925
926     Screen& operator<<(Screen & screen,const std::string & whatto);
927     Screen& operator<<(Screen & screen,char const * const & whatto);
928     Screen& operator<<(Screen & screen,char * const & whatto);
929
930     Screen& operator<<(Screen & screen,const Fg::Color & whatto);
931     Screen& operator<<(Screen & screen,const Fg::Style & whatto);
932     Screen& operator<<(Screen & screen,const Bg::Color & whatto);
933     Screen& operator<<(Screen & screen,const Control::_PositionYX & whatto
            );
934     Screen& operator<<(Screen & screen,const Control::_Refresh & whatto);
935     Screen& operator<<(Screen & screen,const Control::_Clear & whatto);
936     Screen& operator<<(Screen & screen,const DisplayStyle & whatto);
937
938     Screen& operator<<(Screen & screen,unsigned int  whatto);
939
940
941     Screen& operator<<(Screen & screen,int  whatto);
942     Screen& operator<<(Screen & screen,std::_Setw  whatto);
943
944     Screen& operator<<(Screen & screen,unsigned long  whatto);
945     Screen& operator<<(Screen & screen,long  whatto);
946     Screen& operator<<(Screen & screen, char  whatto);
947     Screen& operator<<(Screen & screen, wchar_t  whatto);
948
949 }
950
951 #endif
```

2.6 include/rexio/throw.h++

```
 1 ////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
```

```
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 /*! \file throw.h++
29   \brief Useful macros for exception handling.
30
31   __WHERE_AM_I__ by Curtis Krauskopf; see whole article:
32   http://www.decompile.com/cpp/faq/file_and_line_error_string.htm
33 */
34 #define STRINGIFY(x) #x
35 #define TOSTRING(x) STRINGIFY(x)
36
37 //! file name and line number as plain string
38 #define __WHERE_AM_I__ "in " __FILE__ ":" TOSTRING(__LINE__)
39 //! throw exception x with __WHERE_AM_I__ as constructor argument
40 #define THROW(x) throw x(__WHERE_AM_I__)
41
42 //! throw exception when assertion evaluates false
43 #define EASSERT(assertion,exception)            \
44     if (!(assertion))THROW(exception)
45
46 //! throw exception, that has specific parameters
47 #define THROWP(x,p) throw x(std::string(__WHERE_AM_I__)+'\n'+(p))
48
49 //! if assertion false, THROWP
50 #define EASSERTP(a,e,p) if (!(a))THROWP(e,p)
```

2.7 include/rexio/tk/activewidget.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
```

```
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __ACTIVEWIDGET_H__
29 #define __ACTIVEWIDGET_H__
30
31 #include <rexio/tk/widget.h++>
32 #include <rexio/tk/window.h++>
33
34 namespace Scr
35 {
36     namespace Tk
37     {
38         const DisplayStyle ACTIVEWIDGET_DEFAULT_STYLE(WIDGET_DEFAULT_STYLE
                );
39         const DisplayStyle ACTIVEWIDGET_DEFAULT_ACTIVESTYLE(
40             DisplayStyle(Fg::Black, Fg::Dark, Bg::White));
41
42         //! Focus capable widget
43         /*!
44           Focusable widget, useful for input fields and other form
                elements.
45         */
46         class ActiveWidget:public Widget
47         {
48         protected:
49             ActiveWidget(Uint _height,
50                          Uint _width,
51                          const DisplayStyle& _style
52                          = ACTIVEWIDGET_DEFAULT_STYLE,
53                          const DisplayStyle& _activeStyle
54                          = ACTIVEWIDGET_DEFAULT_ACTIVESTYLE)
55                 throw();
56             ActiveWidget(const DisplayStyle& _style
57                          = ACTIVEWIDGET_DEFAULT_STYLE,
58                          const DisplayStyle& _activeStyle
59                          = ACTIVEWIDGET_DEFAULT_ACTIVESTYLE)
60                 throw();
61
62             virtual void SetStylesheet(Stylesheet* _styleSheet)throw() {
63                 Widget::SetStylesheet(_styleSheet);
64                 __FetchProperty(style, "style");
65                 __FetchProperty(activeStyle, "activeStyle");
66             }
67
68             ~ActiveWidget()throw();
69
70             DisplayStyle activeStyle;
71             bool active;
72         public:
73             void OnFocus(FocusPolicy focustype)throw();
74             void OnUnFocus(FocusPolicy focustype)throw();
75             void OnKeyDown(Key key)throw();
76             virtual void OnAction()throw();
77
78             void SetActive(bool _active)throw();
79             bool GetActive()throw();
80
81             DisplayStyle& GetActiveStyle()throw();
```

```
82             void SetActiveStyle(const DisplayStyle& _activeStyle
83                                 = DisplayStyle(Fg::White, Fg::Dark,
84                                                Bg::Black))throw();
85             RTTI_OBJ(ActiveWidget, Widget);
86         };
87     }
88 }
89
90 #endif // __ACTIVEWIDGET_H__
```

2.8 include/rexio/tk/autolist.h++

```
 1 ////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __AUTOLIST_H__
29 #define __AUTOLIST_H__
30
31 #include <list>
32 #include <tr1/unordered_map>
33 // #ifdef __GNUC__
34 // #define HASH_MAP_NAMESPACE __gnu_cxx
35 // #include <ext/hash_map>
36 // #else
37 // #include <hash_map>
38 // #define HASH_MAP_NAMESPACE std
39 // #endif
40
41 //namespace HASH_MAP_NAMESPACE {
42 //}
43
44 namespace Scr
45 {
```

```
46 /*!
47   \brief container combining advantages of list and hash map,
48   allowing
49
50   It is implemented using standard STL list and almost_standard hash_map.
51  */
52 template <class T>
53 class AutoList
54 {
55 public:
56     typedef typename std::list<T>::const_reverse_iterator
          const_reverse_iterator;
57     typedef typename std::list<T>::reverse_iterator reverse_iterator;
58     typedef typename std::list<T>::const_iterator const_iterator;
59     typedef typename std::list<T>::iterator iterator;
60     typedef typename std::list<T>::size_type size_type;
61 private:
62     template <class _T>
63     struct _hash{
64         size_t operator() (const _T x) const {
65             return reinterpret_cast<size_t>(x);
66         }
67     };
68     std::list<T> list;
69
70 //  HASH_MAP_NAMESPACE::
71     typedef
72     std::tr1::unordered_map<const T, iterator,_hash<T> > hashmap_t;
73     hashmap_t hashmap;
74     typedef typename hashmap_t
75     ::iterator hashiterator;
76
77     size_type _size;
78 public:
79
80     AutoList<T>() : list(), hashmap(), _size(0) { };
81
82     /*!
83       \param elem element to find
84       \return list iterator to specific element
85      */
86     iterator operator[](T &elem) {
87         hashiterator temp = hashmap.find(elem);
88         if(temp == hashmap.end())
89             return list.end();
90         return temp->second;
91     }
92     iterator operator[](const T &elem) {
93         hashiterator temp = hashmap.find(elem);
94         if(temp == hashmap.end())
95             return list.end();
96         return temp->second;
97     }
98
99     /*!
100      \return list iterator to lase element
101     */
102    reverse_iterator rbegin() {
103        return list.rbegin();
104    }
105    /*!
106      \return list iterator rend() of list
```

```
107      */
108     reverse_iterator rend() {
109         return list.rend();
110     }
111     /*!
112       \return list iterator to first element
113      */
114     iterator begin() {
115         return list.begin();
116     }
117     /*!
118       \return list iterator end() of list
119      */
120     iterator end() {
121         return list.end();
122     }
123     /*!
124       \return list iterator to first element
125      */
126     const_iterator begin()const {
127         return list.begin();
128     }
129     /*!
130       \return list iterator end() of list
131      */
132     const_iterator end()const {
133         return list.end();
134     }
135     /*!
136       \return last element in the list
137      */
138     const T& back() {
139         return (list.back());
140     }
141     /*!
142       \return number of elements
143      */
144     size_type size() {
145         return _size;
146     }
147     /*!
148       \return true if _size is 0
149      */
150     bool empty() {
151         return _size?false:true;
152     }
153
154     /*!
155       \param i list iterator to  specific element to be erased
156      */
157     void erase(iterator i) {
158         _size--;
159         hashmap.key_erase(*i);
160         list.erase(i);
161     }
162
163     /*!
164       \param elem  specific element to be erased
165      */
166     void remove(T elem) {
167         _size--;
168         hashiterator i = hashmap.find(elem);
```

```
169         list.erase(i->second);
170         hashmap.erase(i);
171     }
172
173     /*!
174       \param before where to insert
175       \param newelem what to insert
176      */
177     iterator insert(const T& before, const T& newelem) {
178         _size++;
179         return (hashmap[newelem] = list.insert(hashmap[before], newelem));
180     }
181
182     /*!
183       \param elem what to insert
184      */
185     void push_front(const T& elem) {
186         _size++;
187         list.push_front(elem);
188         hashmap[elem] = list.front();
189     }
190     /*!
191       \param elem what to insert
192      */
193     void push_back(const T& elem) {
194         _size++;
195         list.push_back(elem);
196         hashmap[elem] = --list.end();
197     }
198
199     /*!
200       \param elem1 element to be swapped w/ elem2
201       \param elem2 element to be swapped w/ elem1
202
203       swaps two elements in the Autolist
204      */
205     void swap(const T& elem1, const T& elem2) {
206         hashiterator hashiter1 = hashmap.find(elem1),
207             hashiter2 = hashmap.find(elem2);
208         iterator iter1 = hashiter1->second;
209         iterator iter2 = hashiter2->second;
210         hashiter1->second = iter2;
211         hashiter2->second = iter1;
212
213         std::iter_swap(iter1, iter2);
214     }
215 };
216 }
217 #endif // __AUTOLIST_H__
```

2.9 include/rexio/tk/boxgroup.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
```

```
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __BOXGROUP_H__
29 #define __BOXGROUP_H__
30
31 //#include <ext/hash_map>
32 #include <rexio/tk/widgetgroup.h++>
33 #include <tr1/unordered_map>
34 //namespace HASH_MAP_NAMESPACE {
35 //  template <>
36 //}
37
38 namespace Scr
39 {
40     namespace Tk
41     {
42         //! Provides horizontal and vertical widget grouping capabilities.
43         /*!
44           Intelligently places the containing widgets among allocated
              space.
45           Widgets can be placed vertically or horizontally.
46           \latexonly
47           \begin{figure}
48           \begin{center}
49           \leavevmode
50           \includegraphics[width=200pt]{../boxgroup1} \\
51           BoxGroup in Horizontal Mode. Widget1's stretchFactor == 1 and
              the others' is 2. \\
52           \includegraphics[width=120pt]{../boxgroup2}
53           \includegraphics[width=120pt]{../boxgroup3}
54           \includegraphics[width=120pt]{../boxgroup4} \\
55           All of the widgets here have their maxSize set so that there is
              still free space. \\
56           It shows different types of AlignPolicy. Respectively: Center,
              Start, Distributed.
57           \end{center}
58           \end{figure}
59           \endlatexonly
60
61           \sa \a VerticalGroup and \a HorizontalGroup provided for
              convenience.
62         */
```

```
63          class BoxGroup:public WidgetGroup
64          {
65          public:
66              /*!
67                Widget aligning policy in case of not all space being used.
68              */
69              typedef enum {
70                  /*!
71                    Align everything to the left/top depending on \a
                          groupType.
72                  */
73                  Begin,
74                  /*!
75                    Align everything to the center.
76                  */
77                  Center,
78                  /*!
79                    Align everything to the right/bottom depending on
80                    \a groupType.
81                  */
82                  End,
83                  /*!
84                    Try to evenly distribute free space between widgets,
85                    adding a margin between each of them.
86                  */
87                  Distribute
88              } AlignPolicy;
89          protected:
90              BoxGroup(Uint _height,
91                       Uint _width,
92                       const DisplayStyle & _style
93                       = WINDOW_DEFAULT_STYLE)throw();
94
95              BoxGroup(const WidgetGroup & base)throw();
96
97              virtual void ArrangeContents()throw() = 0;
98
99              //! Additional data used for positioning.
100             /*!
101               Widget layouting information inside BoxGroup.
102             */
103             struct LayoutData {
104                 LayoutData(Uint _stretchFactor) :
105                     stretchFactor(_stretchFactor) {RexIOLog(
                            LogLevelModerate) << "Layout data created" << std
                            ::endl;}
106                 LayoutData() : stretchFactor(1) {RexIOLog(LogLevelModerate
                        ) << "Layout data created" << std::endl;}
107                 /*!
108                   Defines a factor of dividing free space between widgets.
109                   i.e.
110                   space = (this_factor/sum_of_factors) * freespace.
111                 */
112                 Uint stretchFactor;
113             };
114
115             struct _hash {
116                 size_t operator() (const Scr::Tk::Widget *x) const {
117                     return reinterpret_cast<size_t>(x);
118                 }
119             };
120
```

```
121 //          HASH_MAP_NAMESPACE::
122             /*!
123               Associates \a LayoutData to each attached widget.
124             */
125             std::tr1::unordered_map<const Widget*, LayoutData,_hash>
                    elementsLayout;
126
127             /*!
128               Current aligning policy.
129             */
130             AlignPolicy alignPolicy;
131         public:
132             /*!
133               \param widget1 First widget
134               \param widget2 Second widget
135
136               Swap two widgets with together, provided that they are being
137               contained by the WidgetGroup. rearrange contents afterwards
138             */
139             virtual void SwapWidgets(Widget& widget1, Widget &widget2)
                    throw();
140             virtual void AddWidget(Widget& widget)throw();
141             /*!
142               \param widget widget to attach to this window
143               \param stretchFactor part of the added widget's \a
                      LayoutData
144
145               Attach a widget to this window.
146               Specifically, add it to the \a elements.
147             */
148             virtual void AddWidget(Widget& widget, Uint stretchFactor)
                    throw();
149             virtual void DelWidget(Widget& widget)throw();
150
151             virtual void OnStart()throw();
152             virtual void OnResize()throw();
153
154             /*!
155               \param _alignPolicy enumerative type parameter
156               specifying aling policy (refer to documentation for this
157               class for information on it)
158
159               Set new BoxGroupType. Can be invoked anytime and it will
160               initiate a redraw.
161             */
162             virtual void SetAlignPolicy(AlignPolicy _alignPolicy)throw();
163             /*!
164               Get current AlignPolicy.
165             */
166             virtual AlignPolicy GetAlignPolicy()throw();
167             virtual ~BoxGroup()throw();
168             RTTI_OBJ(BoxGroup, WidgetGroup);
169         };
170     }
171 }
172
173 #endif // __BOXGROUP_H__
```

2.10 include/rexio/tk/button.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __BUTTON_H__
29 #define __BUTTON_H__
30
31 #include <rexio/tk/activewidget.h++>
32
33
34 namespace Scr
35 {
36     namespace Tk
37     {
38         const DisplayStyle BUTTON_DEFAULT_STYLE(WIDGET_DEFAULT_STYLE);
39         //! style for button, when it is focused
40         const DisplayStyle BUTTON_DEFAULT_ACTIVESTYLE(
41             DisplayStyle(Scr::Fg::Black, Scr::Fg::Dark, Scr::Bg::White));
42         class Button:public ActiveWidget
43         {
44         private:
45             std::string label;
46         public:
47             Button(Uint _height,
48                    Uint _width,
49                    const std::string& _label,
50                    const DisplayStyle& _style = BUTTON_DEFAULT_STYLE,
51                    const DisplayStyle& _activeStyle
52                    = BUTTON_DEFAULT_ACTIVESTYLE)
53                 throw();
54             Button(const std::string& _label,
55                    const DisplayStyle& _style = BUTTON_DEFAULT_STYLE,
56                    const DisplayStyle& _activeStyle
57                    = BUTTON_DEFAULT_ACTIVESTYLE)
58                 throw();
59             ~Button()throw();
60             void OnRedraw(Screen& screen)throw();
```

```
61              void SetLabel(const std::string& _label)throw();
62              std::string& GetLabel()throw();
63
64              RTTI_OBJ(Button, ActiveWidget);
65          };
66      }
67 }
68 #endif // __BUTTON_H__
```

2.11 include/rexio/tk/checkbox.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __CHECKBOX_H__
29 #define __CHECKBOX_H__
30
31 #include <rexio/tk/activewidget.h++>
32 #include <rexio/tk/label.h++>
33
34 namespace Scr
35 {
36     namespace Tk
37     {
38         const DisplayStyle CHECKBOX_DEFAULT_STYLE(
39             ACTIVEWIDGET_DEFAULT_STYLE);
40         const DisplayStyle CHECKBOX_DEFAULT_ACTIVESTYLE(
41             ACTIVEWIDGET_DEFAULT_ACTIVESTYLE);
42
43         /*!
44           \brief two-state widget
45
46           A widgets that indicates setting of boolean feature canonly
```

```
47          be turned on and off. It has label, that indicates its name
48          and boolean field that indicates its current state
49         */
50        class Checkbox:public ActiveWidget
51        {
52        private:
53            //! label near the checkbox
54            Label label;
55
56            /*!
57              current state (on/off) displayed to user
58            */
59            bool state;
60        protected:
61            Checkbox(Uint _height,
62                     Uint _width,
63                     const Label& _label,
64                     const DisplayStyle& _style
65                     = CHECKBOX_DEFAULT_STYLE,
66                     const DisplayStyle& _activeStyle
67                     = CHECKBOX_DEFAULT_ACTIVESTYLE)throw();
68            Checkbox(const Label& _label,
69                     const DisplayStyle& _style
70                     = CHECKBOX_DEFAULT_STYLE,
71                     const DisplayStyle& _activeStyle
72                     = CHECKBOX_DEFAULT_ACTIVESTYLE)throw();
73            ~Checkbox()throw();
74        public:
75            void OnRedraw(Screen& screen)throw();
76            void OnAction()throw();
77            void SetLabel(const Label& _label)throw();
78            const Label& GetLabel()throw();
79            void SetState(bool _state)throw();
80            bool GetState()throw();
81            RTTI_OBJ(Checkbox, ActiveWidget);
82        };
83    }
84 }
85 #endif // __CHECKBOX_H__
```

2.12 include/rexio/tk/framedwindow.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
```

```
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __FRAMEDWINDOW_H__
29 #define __FRAMEDWINDOW_H__
30
31 #include <rexio/tk/virtualwindow.h++>
32
33 namespace Scr
34 {
35     namespace Tk
36     {
37         const wchar_t _DEFAULT_FRAME_TOP = 0x2500 ;
38         const wchar_t _DEFAULT_FRAME_BOTTOM = 0x2500;
39         const wchar_t _DEFAULT_FRAME_LEFT = 0x2502;
40         const wchar_t _DEFAULT_FRAME_RIGHT  = 0x2502;
41         const wchar_t _DEFAULT_FRAME_TOPLEFT = 0x250C;
42         const wchar_t _DEFAULT_FRAME_TOPRIGHT = 0x2510;
43         const wchar_t _DEFAULT_FRAME_BOTTOMLEFT = 0x2514;
44         const wchar_t _DEFAULT_FRAME_BOTTOMRIGHT = 0x2518;
45         /*!
46           Frame specific style.
47         */
48         struct FrameStyle {
49         public:
50             /*!
51               \param _frameColor frame color
52               \param top
53               \param bottom
54               \param left
55               \param right
56               \param topLeft
57               \param topRight
58               \param bottomLeft
59               \param bottomRight
60             */
61             FrameStyle(const DisplayStyle& _frameColor,
62                        wchar_t top = _DEFAULT_FRAME_TOP,
63                        wchar_t bottom = _DEFAULT_FRAME_BOTTOM,
64                        wchar_t left = _DEFAULT_FRAME_LEFT,
65                        wchar_t right = _DEFAULT_FRAME_RIGHT,
66                        wchar_t topLeft = _DEFAULT_FRAME_TOPLEFT,
67                        wchar_t topRight = _DEFAULT_FRAME_TOPRIGHT,
68                        wchar_t bottomLeft = _DEFAULT_FRAME_BOTTOMLEFT,
69                        wchar_t bottomRight = _DEFAULT_FRAME_BOTTOMRIGHT) :
70                 frameColor(_frameColor) {
71                 properties.top = top;
72                 properties.bottom = bottom;
73                 properties.left = left;
74                 properties.right = right;
75                 properties.topLeft = topLeft;
76                 properties.topRight = topRight;
77                 properties.bottomLeft = bottomLeft;
```

```
78              properties.bottomRight = bottomRight;
79          }
80          //! color of the frame
81          DisplayStyle frameColor;
82          //! holds characters used for frame drawing
83          union {
84              wchar_t frame[8];
85              struct {
86                  wchar_t top;
87                  wchar_t bottom;
88                  wchar_t left;
89                  wchar_t right;
90                  wchar_t topLeft;
91                  wchar_t topRight;
92                  wchar_t bottomLeft;
93                  wchar_t bottomRight;
94              } properties;
95          };
96      }; // FrameStyle
97
98      const DisplayStyle _DEFAULT_FRAME_COLOR(Fg::White, Fg::Bright,
99                                              Bg::Black);
100     const DisplayStyle FRAMEDWINDOW_DEFAULT_STYLE(Fg::System, Fg::Dark
        ,
101                                                   Bg::System);
102     const FrameStyle FRAMEDWINDOW_DEFAULT_FRAMESTYLE(
103         _DEFAULT_FRAME_COLOR);
104
105     /*!
106       \param W class of inside's window.
107       Template for all framed windows.
108       FramedWindowBase is basically a window having a separate
            internal
109       window to which most of the calls(like AddWidget) are routed.
110     */
111     template <class W>
112     class FramedWindowBase : public VirtualWindow<W>
113     {
114     protected:
115         using VirtualWindow<W>::inside;
116     public:
117         using VirtualWindow<W>::GetHeight;
118         using VirtualWindow<W>::GetWidth;
119         using VirtualWindow<W>::SetStyle;
120         using VirtualWindow<W>::GetStyle;
121
122     protected:
123         //! how to draw a frame around \a inside.
124         FrameStyle frameStyle;
125     public:
126         /*!
127           \param _height desired height
128           \param _width desired width
129           \param _style optional style
130           \param _frameStyle optional frame style
131         */
132         FramedWindowBase(Uint _height,  Uint _width,
133                          const DisplayStyle& _style
134                          = FRAMEDWINDOW_DEFAULT_STYLE,
135                          const FrameStyle& _frameStyle
136                          = FRAMEDWINDOW_DEFAULT_FRAMESTYLE) throw()
```

```
137             : VirtualWindow<W>(_height, _width, _frameStyle.frameColor
                    ),
138               frameStyle(_frameStyle) {
139
140             inside.SetSize(Size(_height - 1, _width - 1));
141             inside.SetPosition(1, 1);
142         }
143         virtual void OnResize()throw() {
144             Widget::OnResize();
145             inside.SetSize(Size((GetHeight() >= 2)?GetHeight() - 2:0,
146                 (GetWidth() >= 2)?GetWidth() - 2:0));
147             inside.SetPosition(1, 1);
148         }
149
150         virtual void SetStylesheet(Stylesheet* _styleSheet)throw() {
151             Window::SetStylesheet(_styleSheet);
152             std::cerr << this->TypeName() << " stylesheet."<<std::endl
                    ;
153             __FetchProperty(frameStyle.frameColor, "frameColor");
154             SetStyle(frameStyle.frameColor);
155             DisplayStyle style;
156             __FetchProperty(style, "style");
157             inside.SetStyle(style);
158
159             __FetchProperty(frameStyle.properties.top,
160                             "top");
161             __FetchProperty(frameStyle.properties.bottom,
162                             "bottom");
163             __FetchProperty(frameStyle.properties.left,
164                             "left");
165             __FetchProperty(frameStyle.properties.right,
166                             "right");
167             __FetchProperty(frameStyle.properties.topLeft,
168                             "topLeft");
169             __FetchProperty(frameStyle.properties.topRight,
170                             "topRight");
171             __FetchProperty(frameStyle.properties.bottomLeft,
172                             "bottomLeft");
173             __FetchProperty(frameStyle.properties.bottomRight,
174                             "bottomRight");
175         }
176
177         virtual void OnRedraw(Screen& screen)throw() {
178             screen << GetStyle();
179             VirtualWindow<W>::OnRedraw(screen);
180             screen << frameStyle.frameColor;
181             screen.GotoYX(0, 0);
182             screen.AddCharacter(frameStyle.properties.topLeft);
183             screen.HorizontalLine(frameStyle.properties.top, GetWidth
                    () - 2);
184             screen.AddCharacter(frameStyle.properties.topRight);
185             screen.GotoYX(GetHeight() - 1, GetWidth() - 1);
186
187             screen.GotoYX(1, 0);
188             screen.VerticalLine(frameStyle.properties.left, GetHeight
                    () - 2);
189             screen.GotoYX(1, GetWidth() - 1);
190             screen.VerticalLine(frameStyle.properties.right, GetHeight
                    () - 2);
191
192             screen.GotoYX(GetHeight() - 1, 0);
193             screen.AddCharacter(frameStyle.properties.bottomLeft);
```

```
194                 screen.HorizontalLine(frameStyle.properties.bottom,
                        GetWidth() - 2);
195                 screen.AddCharacter(frameStyle.properties.bottomRight);
196             }
197
198             virtual void SetFrameStyle(const FrameStyle &_frameStyle) {
199                 frameStyle = _frameStyle;
200             }
201
202 //          RTTI_OBJ(FramedWindow, Window);
203         }; // FramedWindowBase
204
205         /*!
206           Basic FramedWindow with basic Window as its internal area.
207         */
208         class FramedWindow : public FramedWindowBase<Window>
209         {
210         public:
211             /*!
212               \param _height desired height
213               \param _width desired width
214               \param _style optional style
215               \param _frameStyle optional frame style
216             */
217             FramedWindow(Uint _height, Uint _width,
218                          const DisplayStyle& _style
219                          = FRAMEDWINDOW_DEFAULT_STYLE,
220                          const FrameStyle& _frameStyle
221                          = FRAMEDWINDOW_DEFAULT_FRAMESTYLE
222                 )throw();
223             RTTI_OBJ(FramedWindow, Window);
224         }; // FramedWindow
225     } // Tk
226 } // Scr
227 #endif // __FRAMEDWINDOW_H__
```

2.13 include/rexio/tk/horizontalgroup.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
```

```
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __HORIZONTALGROUP_H__
29 #define __HORIZONTALGROUP_H__
30
31 #include <rexio/tk/boxgroup.h++>
32
33 namespace Scr
34 {
35     namespace Tk
36     {
37         //! Horizontal widget grouping capabilities.
38         /*!
39           Intelligently places the containing widgets among allocated
                space.
40           Widgets are placed horizontally.
41         */
42         class HorizontalGroup:virtual public BoxGroup
43         {
44         protected:
45             virtual void ArrangeContents()throw();
46         public:
47             HorizontalGroup(const WidgetGroup & base)throw();
48             HorizontalGroup(Uint _height,
49                             Uint _width,
50                             const DisplayStyle & _style
51                             = DisplayStyle(Fg::White,Fg::Dark,Bg::Black))
52                 throw();
53             virtual ~HorizontalGroup()throw();
54             RTTI_OBJ(HorizontalGroup, BoxGroup);
55         };
56     }
57 }
58 #endif // __HORIZONTALGROUP_H__
```

2.14 include/rexio/tk/inputbox.h++

```
 1 ////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
```

```
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #include "activewidget.h++"
29
30 /*
31   TODO:
32   -Undo stack.
33   -Text marking.
34   -Possible clipboard interaction.
35 */
36
37 namespace Scr
38 {
39     namespace Tk
40     {
41         struct InputboxStyle {
42             InputboxStyle(const DisplayStyle& _cursorStyle)throw() :
43                 cursorStyle(_cursorStyle) {;}
44             DisplayStyle cursorStyle;
45         };
46
47         const DisplayStyle INPUTBOX_DEFAULT_STYLE(
48             ACTIVEWIDGET_DEFAULT_STYLE);
49         const DisplayStyle INPUTBOX_DEFAULT_ACTIVESTYLE(
50             ACTIVEWIDGET_DEFAULT_ACTIVESTYLE);
51         const InputboxStyle INPUTBOX_DEFAULT_IBOXSTYLE(
52             DisplayStyle(Fg::System, Fg::Bright, Bg::System));
53
54         //! Simple text input field.
55         /* Text input field with wide char support and text winding. */
56         class Inputbox : public ActiveWidget
57         {
58         private:
59             //! Column position of the cursor.
60             //! 0 is considered beginning of Inputbox
61             Uint cursorPos;
62             //! After which character in the current input
63             //! the cursor is located
64             Uint charPos;
65
66             //! Currently shown number of columns
67             Uint curCols;
68             //! Currently shown number of characters
69             Uint curChars;
70         protected:
71             //! Index of first character currently visible in the input.
72             Uint textOffset;
73
74             //! Text content..
75             std::wstring text;
76
```

```
77            //! Inputbox specific style
78            InputboxStyle inputboxStyle;
79
80            //! Maximum length of input
81            Uint maxLength;
82
83        public:
84            __DE(OffsetOutOfRange, Exception); // Wrong offset supplied
85
86            Inputbox(Uint _width,
87                     const std::wstring&  _text,
88                     const DisplayStyle& _style
89                     = INPUTBOX_DEFAULT_STYLE,
90                     const DisplayStyle& _activeStyle
91                     = INPUTBOX_DEFAULT_ACTIVESTYLE,
92                     const InputboxStyle& _inputboxStyle
93                     = INPUTBOX_DEFAULT_IBOXSTYLE)throw();
94            Inputbox(const std::wstring&  _text,
95                     const DisplayStyle& _style
96                     = INPUTBOX_DEFAULT_STYLE,
97                     const DisplayStyle& _activeStyle
98                     = INPUTBOX_DEFAULT_ACTIVESTYLE,
99                     const InputboxStyle& _inputboxStyle
100                    = INPUTBOX_DEFAULT_IBOXSTYLE)throw();
101
102           /*!
103             \param _text Extended string to replace current content
104             of inputbox
105
106             Set the actual content text.
107           */
108           virtual void SetText(const std::wstring& _text)throw();
109
110           /*!
111             \return const reference to the containing text
112
113             Get the content text.
114           */
115           virtual const std::wstring& GetText()throw();
116
117           /*!
118             \param _maxLength new value
119
120             Set max length of possible input
121           */
122           virtual void SetMaxLength(Uint _maxLength)throw();
123
124           /*!
125             \return \a maxLength
126
127             Get max length of possible input
128           */
129           virtual Uint GetMaxLength()throw();
130
131           /*!
132             \param _textOffset new value
133
134             Set new text offset.
135
136             \exception OffsetOutOfRange is thrown had the offset been
137             wrongly provided.
138           */
```

```
139             virtual void SetOffset(Uint _textOffset)throw(OffsetOutOfRange
                    );
140
141             /*!
142               \return \a textOffset
143
144               Return current text offset.
145             */
146             virtual Uint GetOffset()throw();
147
148
149             virtual void SetStylesheet(Stylesheet* _styleSheet)throw() {
150                 ActiveWidget::SetStylesheet(_styleSheet);
151                 __FetchProperty(inputboxStyle.cursorStyle, "cursorStyle");
152             }
153
154             virtual void OnKeyDown(Key key)throw();
155             virtual void OnRedraw(Screen& screen)throw();
156
157             ~Inputbox()throw();
158             RTTI_OBJ(Inputbox, ActiveWidget);
159         };
160     }
161 }
```

2.15 include/rexio/tk/label.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __LABEL_H__
29 #define __LABEL_H__
30
```

```
31 #include <rexio/screen.h++>
32 #include <rexio/tk/window.h++>
33
34 namespace Scr
35 {
36     namespace Tk
37     {
38         const DisplayStyle LABEL_DEFAULT_STYLE(
39             DisplayStyle(Fg::Transparent, Fg::Dark, Bg::Transparent));
40         /*!
41           Simple text data holder.
42         */
43         class Label : public Widget
44         {
45         protected:
46             /*!
47               Actual label holder.
48             */
49             std::string label;
50         public:
51             Label(const DisplayStyle& _style = LABEL_DEFAULT_STYLE)throw()
                  ;
52             explicit Label(Uint _width,
53                            const std::string&  _label,
54                            const DisplayStyle& _style =
                                   LABEL_DEFAULT_STYLE)
55                 throw();
56             explicit Label(const std::string&  _label,
57                            const DisplayStyle& _style =
                                   LABEL_DEFAULT_STYLE)
58                 throw();
59
60             virtual void SetStylesheet(Stylesheet* _styleSheet)throw() {
61                 Widget::SetStylesheet(_styleSheet);
62                 __FetchProperty(style, "style");
63                 __FetchProperty(label, "content");
64             }
65
66             /*!
67               \return containing text
68
69               Return the actual label text.
70             */
71             virtual const std::string& GetText() const throw();
72             /*!
73               \param _label string to replace current content
74               of label
75
76               Set the actual label text.
77             */
78             virtual void SetText(const std::string _label)throw();
79             virtual void OnFocus(FocusPolicy focustype)throw();
80             virtual void OnUnFocus(FocusPolicy focustype)throw();
81             virtual void OnRedraw(Screen& screen)throw();
82
83             virtual ~Label()throw();
84             RTTI_OBJ(Label, Widget);
85         };
86
87     }
88     Screen& operator<<(Screen & screen,const Tk::Label& whatto);
89
```

```
90 }
91
92 #endif // __LABEL_H__
```

2.16 include/rexio/tk/rootwindow.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __ROOTWINDOW_H__
29 #define __ROOTWINDOW_H__
30
31 #include <rexio/screen.h++>
32 #include <rexio/tk/window.h++>
33 #include <fstream>
34
35 namespace Scr
36 {
37     namespace Tk
38     {
39         const DisplayStyle ROOTWINDOW_DEFAULT_STYLE(WINDOW_DEFAULT_STYLE);
40
41         //! Main application window.
42         /*!
43           Lord of the widgets and the main connection point between
44           the Toolkit and lower level library.
45         */
46         class RootWindow:public Connection,public Window
47         {
48         protected:
49             /*!
50               \param _input input stream handler
51               \param _output output stream handler
```

```
52              \param _style default style
53
54              The input handlers make it possible to attach to any
                    character
55              device. Specifically it can be an ordinary terminal
56              or a tcp connection to a remote telnet application.
57            */
58            RootWindow(std::istream& _input, std::ostream& _output,
59                       const DisplayStyle & _style
60                       = ROOTWINDOW_DEFAULT_STYLE)throw();
61            /*!
62              \return Screen handler reference.
63            */
64            virtual Screen& GetScreen()throw();
65            /*!
66              \return 0
67            */
68            virtual Uint GetAbsoluteColumn()throw();
69            /*!
70              \return 0
71            */
72            virtual Uint GetAbsoluteRow()throw();
73          public:
74
75            // Process events recieved as Connection
76            // using procedures typical to Window
77            using Connection::OnStart;
78            using Connection::OnKeyDown;
79            using Window::OnResize;
80
81            /*!
82              \copydoc Connection::Start(int, char **)
83
84              \a RootWindow specific:
85
86              Arguments:
87              -style=FILE     - Use this FILE as a stylesheet.
88            */
89            virtual int Start(int argc, char **argv)throw(StartFailed,
                    Screen::IllegalCharacter);
90            virtual int Start()throw(StartFailed,Screen::IllegalCharacter)
                    ;
91
92            /*!
93              \return referene to self
94            */
95            virtual RootWindow& GetRootWindow()throw();
96
97            virtual void OnStart()throw();
98            virtual void OnKeyDown(Key key)throw();
99            virtual void OnRedraw(Screen& screen)throw();
100           virtual void OnResize(Uint rows, Uint cols)throw();
101
102           __DE(FileNotOpened, Exception);
103           /*!
104             \param filename location of the stylesheet
105
106             Loads stylesheet from the given location.
107
108             \exception FileNotOpened is thrown if the file couldn't
109             be opened.
110
```

```
111               \exception ParsingError is thrown if the input file
                      contained
112               inappropriate input.
113             */
114             void LoadStylesheet(const char* filename)
115                 throw(FileNotOpened, Stylesheet::ParsingError);
116
117             /*!
118               Repaints whole screen (useful after invoking background
119               programs, that modify its content)
120              */
121             void ForceRepaint()throw();
122
123             /*!
124               Trigger OnRedraw event
125              */
126             void ForceOnRedraw()throw()
127                 {OnRedraw(*screen);}
128
129             virtual ~RootWindow()throw();
130
131             RTTI_OBJ(RootWindow, Window);
132         }; // RootWindow
133     } // Tk
134 } // Scr
135 #endif // __ROOT_WINDOW_H__
```

2.17 include/rexio/tk/rtti.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 /*! \file rtti.h++
```

```
29   RTTI - Run Time Type Information
30   This macros can expand a class to have custom RTTI capabilities.
31 */
32
33 #ifndef __RTTI_H__
34 #define __RTTI_H__
35
36 #include <vector>
37 #include <string>
38
39 #define RTTI_BASE(__name) \
40 public: \
41 typedef std::vector<std::string> ClassHierarchy; \
42 protected: \
43 ClassHierarchy classHierarchy; \
44 public: \
45 virtual bool IsTypeOf(std::string _className) const \
46 { \
47     if(#__name == _className) \
48         return true; \
49     else \
50         return false; \
51 }\
52 virtual const char * TypeName() const \
53 { \
54     return #__name; \
55 }\
56 virtual const char * ParentName() const \
57 { \
58     return NULL; \
59 }\
60 const ClassHierarchy& Hierarchy() \
61 { \
62     if(!classHierarchy.size()) \
63         Hierarchy(classHierarchy); \
64     return classHierarchy; \
65 } \
66 protected: \
67 virtual void Hierarchy(ClassHierarchy &vec) \
68 { \
69     vec.push_back(#__name); \
70 }
71
72 #define RTTI_OBJ(__name, __parent)\
73 public: \
74 virtual bool IsTypeOf(std::string _className) const \
75 { \
76     if(#__name == _className) \
77         return true; \
78     return __parent::IsTypeOf(_className); \
79 } \
80 virtual const char * TypeName() const \
81 { \
82     return #__name; \
83 }\
84 virtual const char * ParentName() const \
85 { \
86     return #__parent; \
87 }\
88 virtual void Hierarchy(ClassHierarchy &vec) \
89 { \
90     vec.push_back(#__name);\
```

```
 91     if(ParentName()) \
 92         __parent::Hierarchy(vec); \
 93 }
 94
 95 #define RTTI_OBJ2(__name, __parent1, __parent2) \
 96 public: \
 97 virtual bool IsTypeOf(std::string _className) const \
 98 { \
 99     if(#__name == _className) \
100         return true; \
101     return __parent1::IsTypeOf(_className) || __parent2::IsTypeOf(
        _className); \
102 } \
103 virtual const char * TypeName() const \
104 { \
105     return #__name; \
106 } \
107 virtual const char * ParentName() const \
108 { \
109     return #__parent1; \
110 } \
111 virtual void Hierarchy(ClassHierarchy &vec) \
112 { \
113     vec.push_back(#__name); \
114     if(ParentName()) \
115         __parent1::Hierarchy(vec); \
116 }
117
118 #endif /* __RTTI_H__ */
```

2.18 include/rexio/tk/scrollbar.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
```

```
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __SCROLLBAR_H__
29 #define __SCROLLBAR_H__
30
31 #include <rexio/tk/widget.h++>
32
33 namespace Scr
34 {
35     namespace Tk
36     {
37         const DisplayStyle _DEFAULT_SCROLLBAR_BUTTON(
38             Fg::Black, Fg::Dark, Bg::White);
39         const DisplayStyle _DEFAULT_SCROLLBAR_BUTTONPRESSED(
40             Fg::Black, Fg::Bright, Bg::White);
41
42         const wchar_t _DEFAULT_SCROLLBAR_BUTTONUP = 0x2191;
43         const wchar_t _DEFAULT_SCROLLBAR_BUTTONDOWN = 0x2193;
44         const wchar_t _DEFAULT_SCROLLBAR_BUTTONLEFT = 0x2190;
45         const wchar_t _DEFAULT_SCROLLBAR_BUTTONRIGHT = 0x2192;
46         const DisplayStyle _DEFAULT_SCROLLBAR_SCROLLBG(
47             Fg::White, Fg::Dark, Bg::Black);
48         const wchar_t _DEFAULT_SCROLLBAR_SCROLLFIELD = 0x2592;
49         const DisplayStyle _DEFAULT_SCROLLBAR_SCROLLFG(
50             Fg::Black, Fg::Dark, Bg::White);
51         const wchar_t _DEFAULT_SCROLLBAR_SCROLLHANDLEV = 0x2195;
52         const wchar_t _DEFAULT_SCROLLBAR_SCROLLHANDLEH = 0x2194;
53
54         //! Scrollbars specific style.
55         /*!
56           Describes the way a specific scrollbar is drawn.
57         */
58         struct ScrollbarStyle {
59             /*!
60               \param _button style for directional buttons
61               \param _buttonPressed style for pressed buttons
62               \param _buttonUp symbol for drawing up button
63               \param _buttonDown symbol for drawing down button
64               \param _buttonLeft symbol for drawing left button
65               \param _buttonRight symbol for drawing right button
66               \param _scrollBg style for drawing scrollbar's
67               \param _scrollField symbol for drawing scrollbar's area
68               \param _scrollFg style for drawing scrollbar's area
69               \param _scrollHandleV symbol for vertical handle
70               \param _scrollHandleH symbol for horizontal handle
71             */
72             ScrollbarStyle(
73                 const DisplayStyle& _button =
74                 _DEFAULT_SCROLLBAR_BUTTON,
75                 const DisplayStyle& _buttonPressed =
76                 _DEFAULT_SCROLLBAR_BUTTONPRESSED,
77                 wchar_t _buttonUp = _DEFAULT_SCROLLBAR_BUTTONUP,
78                 wchar_t _buttonDown = _DEFAULT_SCROLLBAR_BUTTONDOWN,
79                 wchar_t _buttonLeft = _DEFAULT_SCROLLBAR_BUTTONLEFT,
80                 wchar_t _buttonRight = _DEFAULT_SCROLLBAR_BUTTONRIGHT,
81                 const DisplayStyle& _scrollBg =
                        _DEFAULT_SCROLLBAR_SCROLLBG,
82                 wchar_t _scrollField = _DEFAULT_SCROLLBAR_SCROLLFIELD,
83                 const DisplayStyle& _scrollFg =
                        _DEFAULT_SCROLLBAR_SCROLLFG,
84                 wchar_t _scrollHandleV = _DEFAULT_SCROLLBAR_SCROLLHANDLEV,
85                 wchar_t _scrollHandleH = _DEFAULT_SCROLLBAR_SCROLLHANDLEH
```

```
86                )throw():
87                button(_button), buttonPressed(_buttonPressed),
88                buttonUp(_buttonUp), buttonDown(_buttonDown),
89                buttonLeft(_buttonLeft), buttonRight(_buttonRight),
90                scrollBg(_scrollBg), scrollField(_scrollField),
91                scrollFg(_scrollFg),
92                scrollHandleV(_scrollHandleV), scrollHandleH(
                      _scrollHandleH)
93                {;}
94            //! style for directional buttons
95            DisplayStyle button;
96            //! style for pressed buttons
97            DisplayStyle buttonPressed;
98            //! symbol for drawing up button
99            wchar_t buttonUp;
100           //! symbol for drawing down button
101           wchar_t buttonDown;
102           //! symbol for drawing left button
103           wchar_t buttonLeft;
104           //! symbol for drawing right button
105           wchar_t buttonRight;
106           //! style for drawing scrollbar's
107           DisplayStyle scrollBg;
108           //! symbol for drawing scrollbar's area
109           wchar_t scrollField;
110           //! style for drawing scrollbar's area
111           DisplayStyle scrollFg;
112           //! symbol for vertical handle
113           wchar_t scrollHandleV;
114           //! symbol for horizontal handle
115           wchar_t scrollHandleH;
116       };
117
118       //! Base for implementing scrollbars.
119       /*!
120         This class implements interface for HorizontalScrollbar
121         and VerticalScrollbar. Allows setting progress, offsets,
122         size, style.
123       */
124       class ScrollbarBase : public Widget
125       {
126       protected:
127           ScrollbarBase(Uint _width, Uint _height,
128                     const ScrollbarStyle& _scrollbarStyle
129                     = ScrollbarStyle()
130               )throw();
131           ScrollbarStyle scrollbarStyle;
132           Uint scrollSize;
133           Uint scrollOffset;
134       public:
135           virtual void OnRedraw(Screen& screen)throw() = 0;
136           /*!
137             \param _scrollSize
138             Set virtual area that the scrollbar should cover.
139           */
140           virtual void SetScrollSize(Uint _scrollSize)throw();
141           /*!
142             \return virtual size the scrollbar covers.
143           */
144           virtual Uint GetScrollSize() const throw();
145           /*!
146             \param _scrollOffset
```

```
147             Set number of virtual offset.
148           */
149           virtual void SetScrollOffset(Uint _scrollOffset)throw();
150           /*!
151             Return virtual offset.
152           */
153           virtual Uint GetScrollOffset() const throw();
154           /*!
155             \param progress
156             Provided for convenience. Sets the scrollOffset
157             in respect to scrollSize accordingly to given
158             progress.
159           */
160           virtual void SetScrollProgress(float progress)throw();
161           /*!
162             \return Current scrolling progress.
163           */
164           virtual float GetScrollProgress() const throw();
165
166           /*! \param _scrollStyle new style
167             Set scrollbar specific style.
168           */
169           virtual void SetScrollbarStyle(
170               const ScrollbarStyle& _scrollStyle)throw();
171           /*!
172             \return current scrollbar specific style
173           */
174           virtual const ScrollbarStyle& GetScrollbarStyle() const throw
                  ();
175
176           virtual void SetStylesheet(Stylesheet* _styleSheet)throw() {
177               Widget::SetStylesheet(_styleSheet);
178               __FetchProperty(scrollbarStyle.button, "buttonStyle");
179               __FetchProperty(scrollbarStyle.buttonPressed,
180                               "buttonPressed");
181               __FetchProperty(scrollbarStyle.buttonUp, "buttonUp");
182               __FetchProperty(scrollbarStyle.buttonDown, "buttonDown");
183               __FetchProperty(scrollbarStyle.buttonLeft, "buttonLeft");
184               __FetchProperty(scrollbarStyle.buttonRight, "buttonRight")
                      ;
185               __FetchProperty(scrollbarStyle.scrollBg, "scrollBg");
186               __FetchProperty(scrollbarStyle.scrollField, "scrollField")
                      ;
187               __FetchProperty(scrollbarStyle.scrollFg, "scrollFg");
188               __FetchProperty(scrollbarStyle.scrollHandleV,
189                               "scrollHandleV");
190               __FetchProperty(scrollbarStyle.scrollHandleH,
191                               "scrollHandleH");
192           }
193
194           RTTI_OBJ(ScrollbarBase, Widget);
195       };
196
197       //! Horizontal scrollbar
198       class HorizontalScrollbar : public ScrollbarBase {
199       public:
200           /*!
201             \param _width
202             \param _scrollbarStyle
203           */
204           HorizontalScrollbar(Uint _width,
205                               const ScrollbarStyle& _scrollbarStyle
```

```
206                                 = ScrollbarStyle())throw();
207             virtual void OnRedraw(Screen& screen)throw();
208
209             RTTI_OBJ(HorizontalScrollbar, ScrollbarBase);
210         };
211         //! Vertical scrollbar
212         class VerticalScrollbar : public ScrollbarBase {
213         public:
214             /*!
215               \param _height
216               \param _scrollbarStyle
217             */
218             VerticalScrollbar(Uint _height,
219                               const ScrollbarStyle& _scrollbarStyle
220                               = ScrollbarStyle())throw();
221             virtual void OnRedraw(Screen& screen)throw();
222             RTTI_OBJ(VerticalScrollbar, ScrollbarBase);
223         };
224     }
225 }
226
227 #endif // __SCROLLBAR_H__
```

2.19 include/rexio/tk/selectbox.h++

```
  1 ////////////////////////////////////////////////////////////////////////
  2 //
  3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
  4 //
  5 // Permission is hereby granted, free of charge, to any person
  6 // obtaining a copy of this software and associated documentation
  7 // files (the "Software"), to deal in the Software without
  8 // restriction, including without limitation the rights to use,
  9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
 10 // copies of the Software, and to permit persons to whom the
 11 // Software is furnished to do so, subject to the following
 12 // conditions:
 13 //
 14 // The above copyright notice and this permission notice shall be
 15 // included in all copies or substantial portions of the Software.
 16 //
 17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
 18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
 24 // OTHER DEALINGS IN THE SOFTWARE.
 25 //
 26 ////////////////////////////////////////////////////////////////////////
 27
 28 #ifndef __SELECTBOX_H__
 29 #define __SELECTBOX_H__
 30
 31 #include <rexio/tk/activewidget.h++>
 32 #include <rexio/tk/label.h++>
```

```
33 #include <rexio/tk/scrollbar.h++>
34 #include <rexio/tk/framedwindow.h++>
35 #include <rexio/tk/verticalgroup.h++>
36 #include <rexio/tk/rootwindow.h++>
37
38 namespace Scr
39 {
40     namespace Tk
41     {
42         const DisplayStyle SELECTBOX_DEFAULT_STYLE(
43             ACTIVEWIDGET_DEFAULT_STYLE);
44         const DisplayStyle SELECTBOX_DEFAULT_ACTIVESTYLE(
45             ACTIVEWIDGET_DEFAULT_ACTIVESTYLE);
46
47         const wchar_t _DEFAULT_SELECTBOX_OPENBUTTON = 0x2193;
48         const DisplayStyle _DEFAULT_SELECTBOX_OPENSTYLE(
49             ACTIVEWIDGET_DEFAULT_ACTIVESTYLE);
50
51         //! Selectbox specific style
52         /*!
53           Describes the way a specific selectbox is drawn.
54         */
55         struct SelectboxStyle {
56             /*!
57               \param _openButton symbol for drawing opening symbol
58               \param _openStyle color for drawing the opening symbol
59             */
60             SelectboxStyle(
61                 const wchar_t _openButton = _DEFAULT_SELECTBOX_OPENBUTTON,
62                 const DisplayStyle& _openStyle =
                        _DEFAULT_SELECTBOX_OPENSTYLE
63                 )throw() :openButton(_openButton), openStyle(_openStyle)
                        {;};
64             wchar_t openButton;
65             DisplayStyle openStyle;
66         };
67         //! Selection form widget.
68         namespace Detail{class Selector;}
69         /*!
70           Widget allowing to select one of available options.
71         */
72         class Selectbox : public ActiveWidget
73         {
74             friend class Detail::Selector;
75         private:
76             //! Actual list of available options at Selectbox.
77             class _SelectList : public FramedWindow
78             {
79             private:
80                 class SelectGroup:public VerticalGroup
81                 {
82                 public:
83                     SelectGroup(Uint _height,
84                                 Uint _width,
85                                 const DisplayStyle & _style
86                                 = DisplayStyle(Fg::White,Fg::Dark,Bg::
                                        Black))
87                     throw()
88                         :
89                         BoxGroup(_height,_width,_style),
90                         VerticalGroup(_height,_width,_style){;}
91                     using Window::elements;
```

```
92                  using Window::activeWidget;
93              };
94          public:
95              //! Scrollbar.
96              VerticalScrollbar scroll;
97              SelectGroup group;
98              /*!
99                previous active widget at RootWindow to which the
100               focus will have to be returned.
101             */
102             Widget *prevActive;
103
104             _SelectList(Uint _width, Uint _height,
105                         const DisplayStyle& _style)throw();
106             void OnResize()throw();
107
108             void CloseSelectList();
109             void OnKeyDown(Key key)throw();
110             void OnFocus(FocusPolicy focustype)throw();
111             void OnUnFocus(FocusPolicy focustype)throw();
112
113             virtual ~_SelectList()throw();
114         };
115     protected:
116         //! internal style
117         SelectboxStyle selectboxStyle;
118         //! list of options
119         _SelectList selectList;
120         //! indicated whether the list of options is open
121         bool opened;
122     public:
123         __DE(NoSuchOption, Exception);
124
125         /*!
126           \param width
127           \param _style
128           \param _activeStyle
129           \param _selectboxStyle
130         */
131         Selectbox(Uint width,
132                   const DisplayStyle& _style =
133                   SELECTBOX_DEFAULT_STYLE,
134                   const DisplayStyle& _activeStyle =
135                   SELECTBOX_DEFAULT_ACTIVESTYLE,
136                   const SelectboxStyle& _selectboxStyle =
137                   SelectboxStyle())throw();
138         Selectbox(const DisplayStyle& _style =
139                   SELECTBOX_DEFAULT_STYLE,
140                   const DisplayStyle& _activeStyle =
141                   SELECTBOX_DEFAULT_ACTIVESTYLE,
142                   const SelectboxStyle& _selectboxStyle =
143                   SelectboxStyle())throw();
144
145         /*!
146           \param name
147           \return unique identifier
148
149           Adds new option to the list.
150         */
151         Uint AddOption(const std::string& name)throw();
152         /*!
153           \param id
```

```
154               \return Selected option
155               \exception NoSuchOption if no option is selected
156             */
157             const std::string& GetOption() const throw(NoSuchOption);
158             /*!
159               \param id identifier of option to delete
160               Deletes option from the list.
161             */
162             void DelOption(Uint id)throw(NoSuchOption) {;};
163             void OnAction()throw();
164             void OnRedraw(Screen& screen)throw();
165             void OnFocus(FocusPolicy focusPolicy)throw();
166             void OnUnFocus(FocusPolicy focusPolicy)throw();
167             RTTI_OBJ(Selectbox, ActiveWidget);
168         };
169     }
170 }
171
172 #endif // __SELECTBOX_H__
```

2.20 include/rexio/tk/stylesheet.h++

```
1  ////////////////////////////////////////////////////////////////////
2  //
3  // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4  //
5  // Permission is hereby granted, free of charge, to any person
6  // obtaining a copy of this software and associated documentation
7  // files (the "Software"), to deal in the Software without
8  // restriction, including without limitation the rights to use,
9  // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __STYLESHEET_H__
29 #define __STYLESHEET_H__
30 #include <rexio/screen.h++>
31 #include <map>
32 #include <rexio/throw.h++>
33 namespace Scr
34 {
35     namespace Tk
```

```
36      {
37          class Widget;
38
39          //! CSS-like properties holder
40          /*!
41            Stylesheet is a class which can hold different properties
42            for different classes. There are few value types supported.
43            It incomporates complete parser.
44          */
45          class Stylesheet
46          {
47          public:
48              class Properties;
49              struct Property;
50
51              __DE(ParsingError, Exception);
52              __DE(UnexpectedCharacter, ParsingError);
53              __DE(BadValue, ParsingError);
54              __DE(UnexpectedEndOfSheet, ParsingError);
55
56          private:
57              //! Type to bind class names to their properties
58              typedef std::map<std::string, Properties *> ClassMap;
59              //! Allows accessing properties of different classes
60              ClassMap classes;
61
62              /*!
63                \param valuestr unparsed value string
64                \return Property properly interpretted and converted
                      valuestr
65
66                The function takes a crude string which is the following
                      part
67                of CSS syntax:
68                \b property: \b value;
69                and converts it into the internal value holder.
70
71                \exception BadValue is throws if the valuestr cannot be
                      parsed.
72                \exception Screen::InvalidUTF8 is thrown if an UTF-8
                      character
73                enclosed in single braces ' ' is not in correct UTF-8 format
                      .
74              */
75              Property ParseValue(const std::string& valuestr)
76                  throw(BadValue, Screen::InvalidUTF8);
77
78          public:
79              //! Type specifying Property value
80              typedef enum {Style, Symbol, Number, String} PropertyType;
81
82              //! Class holding multiple possible types of values.
83              class Property {
84              private:
85  //              Property();
86                  //! Current type.
87                  PropertyType type;
88                  union {
89                      //! Holding of DisplayStyle
90                      DisplayStyle *style;
91                      union {
92                          //! Holding of unicode character
```

```
93                      wchar_t symbol;
94                      //! Holding of integer
95                      Uint32 number;
96                  };
97                  //! Holding of string
98                  std::string *str;
99              };
100         public:
101             /*!
102               \param old
103               Assign operator handling the allocated objects.
104             */
105             const Property& operator=(const Property &old) {
106                 type = old.type;
107                 switch(type) {
108                 case Style:
109                     style = new DisplayStyle(*(old.style)); break;
110                 case String:
111                     str = new std::string(*(old.str)); break;
112                 case Symbol:
113                     symbol = old.symbol; break;
114                 case Number:
115                     number = old.number; break;
116                 }
117                 return *this;
118             }
119             /*!
120               \param old
121               Copy constructor handling the allocated objects.
122             */
123             Property(const Property &old) {
124                 *this = old;
125             }
126             /*!
127               \param _style data to hold
128               Specialized constructor for holding DisplayStyle data.
129             */
130             Property(const DisplayStyle& _style)
131                 :type(Style), style(new DisplayStyle(_style)) {;}
132             /*!
133               \param _symbol data to hold
134               Specialized constructor for holding wchar_t data.
135             */
136             Property( wchar_t _symbol)
137                 :type(Symbol), symbol(_symbol) {;}
138             /*!
139               \param _number data to hold
140               Specialized constructor for holding Uint32 data.
141             */
142             Property( Uint32 _number)
143                 :type(Number), number(_number) {;}
144             /*!
145               \param _str data to hold
146               Specialized constructor for holding std::string data.
147             */
148             Property(const std::string& _str)
149                 :type(String), str(new std::string(_str)) {;}
150             /*!
151               \return type of a Property
152             */
153             PropertyType GetType() const throw() { return type;}
154
```

```
155                 __DE(WrongPropertyConversion, Exception);
156
157                 /*!
158                   Autoconversion to DisplayStyle.
159                 */
160                 operator DisplayStyle() const {
161                     if(type != Style) THROW(WrongPropertyConversion);
162                     return DisplayStyle(*style);
163                 };
164                 /*!
165                   Autoconversion to std::string..
166                 */
167                 operator const std::string() const {
168                     if(type != String) THROW(WrongPropertyConversion);
169                     return *str;
170                 };
171                 /*!
172                   Autoconversion to Uint32.
173                 */
174                 operator Uint32() const {
175                     if(type != Number) THROW(WrongPropertyConversion);
176                     return number;
177                 };
178                 /*!
179                   Autoconversion to wchar_t.
180                 */
181                 operator wchar_t() const {
182                     if(type != Symbol) THROW(WrongPropertyConversion);
183                     return symbol;
184                 };
185
186                 /*!
187                   Smart destructor, deleting type specific data.
188                 */
189                 ~Property() {
190                     if(type == Style)
191                         delete style;
192                     else if(type == String)
193                         delete str;
194                 }
195             };
196             class Properties {
197             private:
198                 //! Type to bind different properties to their actual
                        values.
199                 typedef std::map<std::string, Property *> PropertyMap;
200
201                 //! Allows accessing properties by property names
202                 PropertyMap properties;
203             public:
204                 __DE(NoSuchProperty, Exception);
205
206                 /*!
207                   \param propertyName
208
209                   \exception NoSuchProperty is thrown if no such
                          propertyName
210                   has been defined.
211                 */
212                 const Property& operator[](const std::string& propertyName
                        )
213                     throw(NoSuchProperty);
```

```
214
215                 /*!
216                   \param propertyName name
217                   \param property value
218
219                   Set the property value.
220                 */
221                 void SetProperty(const std::string& propertyName,
222                                  const Property& property)throw();
223
224                 ~Properties();
225
226             };
227
228         public:
229             __DE(NoSuchClass, Exception);
230             /*!
231               \param w widget to check
232               \param property
233               \return reference to found property
234
235               Find certain property value for a widget.
236
237               \exception Properties::NoSuchProperty is thrown if no data
238               has been found.
239             */
240             const Property& GetProperty(const Widget& w,
241                                         const std::string& property)
242                 throw(Properties::NoSuchProperty);
243
244             /*!
245               \param className
246               \param property
247               \param value
248
249               Bind a certain vlaue to certain class's property.
250             */
251             void SetProperty(const std::string& className,
252                              const std::string& property,
253                              const Property& value)throw();
254
255             /*!
256               \param ss stream of CSS-like formatted data
257
258               Parses the specified buffer for later access.
259
260               \exception ParsingError is thrown had the buffer was not
261               properly formatted.
262               \exception Screen::InvalidUTF8 is thrown if an UTF-8
                      character
263               enclosed in single braces ' ' is not in correct UTF-8 format
                      .
264             */
265             Stylesheet(std::istream &ss)throw(ParsingError,
266                                               Screen::InvalidUTF8);
267
268             ~Stylesheet();
269         };
270     }
271 }
272
273 #endif // __STYLESHEET_H__
```

2.21 include/rexio/tk/toolkit.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __TOOLKIT_H__
29 #define __TOOLKIT_H__
30
31 #include <rexio/screen.h++>
32 #include <rexio/tk/widget.h++>
33 #include <rexio/tk/window.h++>
34 #include <rexio/tk/rootwindow.h++>
35 #include <rexio/tk/activewidget.h++>
36 #include <rexio/tk/widgetgroup.h++>
37 #include <rexio/tk/verticalgroup.h++>
38 #include <rexio/tk/horizontalgroup.h++>
39 #include <rexio/tk/label.h++>
40 #include <rexio/tk/button.h++>
41 #include <rexio/tk/checkbox.h++>
42 #include <rexio/tk/inputbox.h++>
43 #include <rexio/tk/framedwindow.h++>
44 #include <rexio/tk/selectbox.h++>
45
46 #endif //__TOOLKIT_H__
```

2.22 include/rexio/tk/verticalgroup.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
```

```
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __VERTICALGROUP_H__
29 #define __VERTICALGROUP_H__
30
31 #include "boxgroup.h++"
32
33 namespace Scr
34 {
35     namespace Tk
36     {
37         //! Vertical widget grouping capabilities.
38         /*!
39           Intelligently places the containing widgets among allocated
                space.
40           Widgets are placed vertically.
41         */
42         class VerticalGroup:virtual public BoxGroup
43         {
44         protected:
45             virtual void ArrangeContents()throw();
46
47         public:
48             VerticalGroup(const WidgetGroup & base)throw();
49             VerticalGroup(Uint _height,
50                           Uint _width,
51                           const DisplayStyle & _style
52                           = DisplayStyle(Fg::White,Fg::Dark,Bg::Black))
53                 throw();
54             virtual  ~VerticalGroup()throw();
55             RTTI_OBJ(VerticalGroup, BoxGroup);
56         };
57     }
58 }
59 #endif // __VERTICALGROUP_H__
```

2.23 include/rexio/tk/virtualwindow.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __VIRTUALWINDOW_H__
29 #define __VIRTUALWINDOW_H__
30
31 #include <rexio/tk/window.h++>
32
33 namespace Scr
34 {
35     namespace Tk
36     {
37
38         /*!
39           \param W class of inside's window.
40           Template for all framed windows.
41           FramedWindowBase is basically a window having a separate
                internal
42           window to which most of the calls(like AddWidget) are routed.
43         */
44         template <class W>
45         class VirtualWindow : public Window
46         {
47         protected:
48             //! internal area, should have Window compatible interface.
49             W inside;
50         public:
51             VirtualWindow(Uint _height,  Uint _width,
52                           const DisplayStyle& _style =
                                  WINDOW_DEFAULT_STYLE,
53                           const DisplayStyle& _inStyle =
54                           WINDOW_DEFAULT_STYLE
55                           )throw()
56                 : Window(_height, _width, _style), inside(_height, _width,
57                                                           _inStyle)
58                 {
59                     Window::AddWidget(inside);
60                 }
```

```
61            /*!
62              \param screen cut-down to actual content area
63
64              Similiar to OnRedraw with an exception of providing cut-down
65              screen.
66            */
67            virtual void OnRedrawInside(Screen& screen)throw() {
68                ;
69            }
70            virtual void OnRedraw(Screen& screen)throw() {
71                Window::OnRedraw(screen);
72                OnRedrawInside(
73                    *screen.CreateSubScreen(inside.GetRow(), inside.GetCol
                          (),
74                                            inside.GetHeight(),
75                                            inside.GetWidth()));
76            }
77
78            /*! \copydoc Window::AddWidget(Widget&)
79              \a VirtualWindow specific:
80              Passes the call to its internal window.
81            */
82            virtual void AddWidget(Widget& widget)throw() {
83                inside.AddWidget(widget);
84                inside.SetPosition(0, 0);
85            }
86            /*! \copydoc Window::DelWidget(Widget&)
87              \a VirtualWindow specific:
88              Passes the call to its internal window.
89            */
90            virtual void DelWidget(Widget& widget)throw() {
91                inside.DelWidget(widget);
92            }
93            /*! \copydoc Widget::OnResize()
94              \a VirtualWindow specific:
95              Has to be overloaded in deriving classes to handle
96              proper resizing of containing window.
97            */
98            virtual void OnResize()throw() = 0;
99
100           RTTI_OBJ(VirtualWindow, Window);
101       }; // VirtualWindow
102   } // Tk
103 } // Scr
104 #endif // __VIRTUALWINDOW_H__
```

2.24 include/rexio/tk/widgetgroup.h++

```
1 ////////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
```

```
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __WIDGETGROUP_H__
29 #define __WIDGETGROUP_H__
30
31 #include "window.h++"
32
33 namespace Scr
34 {
35     namespace Tk
36     {
37         //! General class for grouping widgets and managing them.
38         /*!
39           This class is a base class for all sorts of of grouping widgets.
40           Widgets inside of
41         */
42         class WidgetGroup:public Window
43         {
44         protected:
45             WidgetGroup(Uint _height,
46                         Uint _width,
47                         const DisplayStyle & _style
48                         = WINDOW_DEFAULT_STYLE)throw();
49             WidgetGroup(const WidgetGroup & base)throw();
50             /*!
51               where all magic is done :)
52              */
53             virtual void ArrangeContents()throw();
54         public:
55             /*!
56               \param widget1 First widget
57               \param widget2 Second widget
58
59               Swap two widgets with together, provided that they are being
60               contained by the WidgetGroup.
61             */
62             virtual void SwapWidgets(Widget& widget1, Widget &widget2)
                  throw();
63             /*!
64               \param widget Targetted widget
65
66               Move the widget further away on the containing widget list.
67               Upon end of the list, move to the beginning.
68             */
69             virtual void ShiftFWidget(Widget &widget)throw();
70             /*!
```

```
71               \param widget Targetted widget
72
73               Move the widget closer on the containing widget list.
74               Upon beginning of the list, move to the end.
75             */
76             virtual void ShiftBWidget(Widget &widget)throw();
77
78             virtual ~WidgetGroup()throw();
79
80             RTTI_OBJ(WidgetGroup, Window);
81         };
82     }
83 }
84
85 #endif // __WIDGETGROUP_H__
```

2.25 include/rexio/tk/widget.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
6 // obtaining a copy of this software and associated documentation
7 // files (the "Software"), to deal in the Software without
8 // restriction, including without limitation the rights to use,
9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __WIDGET_H__
29 #define __WIDGET_H__
30 #include <rexio/screen.h++>
31 #include <rexio/tk/stylesheet.h++>
32
33 // #ifdef __GNUC__
34 // #define HASH_MAP_NAMESPACE __gnu_cxx
35 // #include <ext/hash_map>
36 // #else
37 // #include <hash_map>
38 // #define HASH_MAP_NAMESPACE std
39 // #endif
```

```
40
41 #include <rexio/tk/rtti.h++>
42
43 #define __FetchProperty(saveto, name) \
44 try { \
45     (saveto) = static_cast<__typeof__(saveto)>( \
46 _styleSheet->GetProperty(*this, name)); \
47 } \
48 catch(Stylesheet::Property::WrongPropertyConversion) { \
49 ; \
50 } \
51 catch(Stylesheet::Properties::NoSuchProperty) { \
52 ; \
53 }
54
55 #define __FetchPropertyDefault(saveto, name, default) \
56 try { \
57     (saveto) = static_cast<__typeof__(saveto)>(\
58 _styleSheet->GetProperty(*this, name)); \
59 } \
60 catch(Stylesheet::Property::WrongPropertyConversion) { \
61     (saveto) = (default); \
62 } \
63 catch(Stylesheet::Properties::NoSuchProperty) { \
64     (saveto) = (default); \
65 }
66
67 namespace Scr
68 {
69     namespace Tk
70     {
71         const DisplayStyle WIDGET_DEFAULT_STYLE(Fg::System, Fg::Dark,
72                                                 Bg::System);
73
74         class Window;
75         //! Base UI element.
76         /*!
77           Widget - according to the dictionary, a device that is very
                useful
78           for a particular job. In our case that can be any UI job and
                thus
79           all the UI elements shall thereby be children of hit.
80           Note that most widgets do not have their own buffer.
81         */
82         class Widget
83         {
84         private:
85             friend class Window; //FIXME
86             /*!
87               All widgets have a pointer to their parent window.
88               For \a RootWindow, it is a pointer to itself
89               \note For not assigned widgets it is NULL.
90             */
91             Window* parentWindow;
92             /*!
93               Pointer to the stylesheet. If NULL, the widget's properties
94               should be left default.
95             */
96             Stylesheet *styleSheet;
97         protected:
98             /*!
99               This constructor should be used for widgets manually
```

```
100             positioned. Widgets managed by \a WidgetGroup should be
101             constructed with a more simple constructor.
102             \param _height desired height
103             \param _width desired width
104             \param _style optional style
105           */
106           Widget(Uint _height,
107                  Uint _width,
108                  const DisplayStyle& _style
109                  = WIDGET_DEFAULT_STYLE)throw();
110           /*!
111             This constructor should be a preferred one if geometry
112             and position of a Widget are to be managed by some
113             \a WidgetGroup.
114             \param _style optional style
115           */
116           Widget(const DisplayStyle& _style
117                  = WIDGET_DEFAULT_STYLE)throw();
118
119           __DE(ParentNotDefined, Exception);
120           __DE(ParentAlreadySet, Exception);
121
122           /*!
123             \param window parent of this widget
124
125             Parent of a widget can be set generally only once.
126             After doing this, widget is ready to face the world
127             so better prepare it properly first.
128             This design decision has been made because of the
                    constructor's
129             primitive nature not being able to sustain all the
130             possibilities.
131
132             \sa \a ReParent
133
134             \exception ParentAlreadySet is thrown had the parent already
135             been set.
136           */
137           void SetParent(Window& window)throw(ParentAlreadySet);
138           /*!
139             \return reference to parent window
140
141             Get reference to parent window.
142
143             \exception ParentNotSet is thrown if the parent window has
                    not
144             been yet specified.
145           */
146           Window& GetParent()throw(ParentNotDefined);
147
148           /*!
149             \param window pointer to parent of this widget, pass NULL
150             after detaching the widget from window.
151
152             Provided for convenience. Sets the parent disregarding any
153             conditions.
154
155             \sa \a SetParent for general use.
156           */
157           void ReParent(Window* window)throw();
158
159
```

```
160             /*!
161               Focus policy defines a condition upon a widget
162               can be focused.
163             */
164             typedef enum {
165                 //! Nothing can focus.
166                 NoFocus = 0x1,
167                 //! Tabulator(or other switching key) can focus.
168                 TabFocus = 0x1,
169                 //! Mouse click can focus.
170                 ClickFocus = 0x2,
171
172                 WheelFocusUp = 0x4,
173                 WheelFocusDown = 0x8,
174
175                 //! Mouse wheel can focus.
176                 WheelFocus = WheelFocusUp|WheelFocusDown,
177                 //! TabFocus + Clickfocus.
178                 StrongFocus = TabFocus|ClickFocus,
179                 //! Full service focus. :-)
180                 AllFocus = TabFocus|ClickFocus|WheelFocus
181             } FocusPolicy;
182             /*
183               Current focus policy.
184             */
185             FocusPolicy focusPolicy;
186
187             /*!
188               Position regarding the \a parentWindow.
189               i.e. (position.row == 3) means that row 3 of \a parentWindow
190               is 0th row of this widget.
191             */
192             Position position;
193
194             /*!
195               Current size.
196             */
197             Size size;
198             /*!
199               Maximal size that the Widget can be expanded to
200               by for example a \a WidgetGroup.
201             */
202             Size sizeMax;
203             /*!
204               Minimal size that the Widget can be shrinked to
205               by for example a \a WidgetGroup.
206             */
207             Size sizeMin;
208
209             /*!
210               Basic style.
211             */
212             DisplayStyle style;
213
214             /*!
215               Implies whether the element is hidden.
216               /note When hidden, the element want be a subject
217               into positioning algorithms and its OnRedraw event
218               won't invoked.
219             */
220             bool hidden;
221
```

```
222
223         public:
224
225             // most of the widgets do not have their own buffers -
226             // this is a sane design decision, not a bug ;-)
227
228             /*!
229               \param _styleSheet pointer to style data
230
231               Apply Stylesheet to this widget. Reinitialize any style
232               properties if their alternatives are supplied.
233             */
234             virtual void SetStylesheet(Stylesheet* _styleSheet)throw();
235
236             /*!
237               \param focustype Type of the event, i.e. mouse click.
238
239               Element focused. Only matters if a proper \a focusPolicy
240               is set.
241             */
242             virtual void OnFocus(FocusPolicy focustype)throw();
243             /*!
244               \param focustype Type of the event, i.e. mouse click.
245
246               Element unfocused. Only matters if a proper \a focusPolicy
247               is set.
248             */
249             virtual void OnUnFocus(FocusPolicy focustype)throw();
250             /*
251               First event after the constructor call.
252             */
253             virtual void OnStart()throw();
254
255             /*!
256               \param screen reference to the screen on which to draw
257
258               This is the main thing, the core of the Widget.
259               Upon this event, the whole content should be redrawn.
260
261               \note the screen parameter is not a real screen,
262               it is a cutdown to our size screen or even some other
263               overloaded screen flavour.
264             */
265             virtual void OnRedraw(Screen& screen)throw();
266
267             /*!
268               If the widget is attached to a window, it invokes
269               parent's RedrawRequest with this widget.
270               If it isn't attached, the function does nothing.
271
272               \sa Window::RedrawRequest(Widget &w)
273             */
274             virtual void RedrawRequest()throw();
275
276             /*!
277               Resize event. Do something i.e. adjust content to the
278               new size.
279             */
280             virtual void OnResize()throw();
281             /*!
282               \param key keycode
283
```

```
284             Keyboard button press event.
285           */
286           virtual void OnKeyDown(Key key)throw();
287
288           /*!
289             Last event BEFORE the destructor call.
290           */
291           virtual void OnExit()throw();
292
293           /*!
294             \param _pos position new position
295
296             Set position of the Widget regarding to the
297             \a parentWindow.
298
299             \exception ParentNotDefined is thrown had the widget not
                  been
300             assigned to any window. Use \a AddWidget.
301           */
302           virtual void SetPosition(const Position& _pos)
303               throw(ParentNotDefined);
304           /*!
305             \param _row new row position
306             \param _col new column position
307
308             Set position of the Widget regarding to the
309             \a parentWindow.
310
311             \exception ParentNotDefined is thrown had the widget not
                  been
312             assigned to any window. Use \a AddWidget.
313           */
314           virtual void SetPosition(Uint _row, Uint _col)
315               throw(ParentNotDefined);
316           /*!
317             \return position
318
319             Get position of the Widget regarding to the
320             \a parentWindow.
321
322             \exception ParentNotDefined is thrown had the widget not
                  been
323             assigned to any window. Use \a AddWidget.
324           */
325           virtual Position GetPosition() const throw(ParentNotDefined);
326           /*!
327             \param _row new row position
328
329             Set position of the Widget regarding to the
330             \a parentWindow .
331
332             \exception ParentNotDefined is thrown had the widget not
                  been
333             assigned to any window. Use \a AddWidget.
334           */
335           virtual void SetRow(Uint _row)throw(ParentNotDefined);
336           /*!
337             \return row position
338
339             Get position of the Widget regarding to the
340             \a parentWindow.
341
```

```
342             \exception ParentNotDefined is thrown had the widget not
                    been
343             assigned to any window. Use \a AddWidget.
344           */
345           virtual Uint GetRow() const throw(ParentNotDefined);
346           /*!
347             \param _col new column position
348
349             Set position of the Widget regarding to the
350             \a parentWindow.
351
352             \exception ParentNotDefined is thrown had the widget not
                    been
353             assigned to any window. Use \a AddWidget.
354           */
355           virtual void SetCol(Uint _col)throw(ParentNotDefined);
356           /*!
357             \return col position
358
359             Get position of the Widget regarding to the
360             \a parentWindow.
361
362             \exception ParentNotDefined is thrown had the widget not
                    been
363             assigned to any window. Use \a AddWidget.
364           */
365           virtual Uint GetCol() const throw(ParentNotDefined);
366
367           /*!
368             \param _size new size
369
370             Set size of the Widget.
371             \note If entered size is bigger than \a GetMaxSize()
372             or smaller than \a GetMinSize(), it will crop the entered
373             value to the boundaries.
374           */
375           virtual void SetSize(const Size& _size)throw();
376           /*!
377             \param _height new height
378             \param _width new width
379
380             Set size of the Widget.
381             \note If entered size is bigger than \a GetMaxSize()
382             or smaller than \a GetMinSize(), it will crop the entered
383             value to the boundaries.
384           */
385           virtual void SetSize(Uint _height, Uint _width)throw();
386           /*!
387             \return size
388
389             Get size of the Widget.
390           */
391           virtual const Size& GetSize() const throw();
392           /*!
393             \param _height new height
394
395             Set height of the Widget.
396             \note If entered size is bigger than \a GetMaxSize()
397             or smaller than \a GetMinSize(), it will crop the entered
398             value to the boundaries.
399           */
400           virtual void SetHeight(Uint _height)throw();
```

```
401             /*!
402               \return height
403
404               Get height of the Widget.
405             */
406             virtual Uint GetHeight() const throw();
407             /*!
408               \param _width new width
409
410               Set width of the Widget.
411               \note If entered size is bigger than \a GetMaxSize()
412               or smaller than \a GetMinSize(), it will crop the entered
413               value to the boundaries.
414             */
415             virtual void SetWidth(Uint _width)throw();
416             /*!
417               \return width
418
419               Get width of the Widget.
420             */
421             virtual Uint GetWidth() const throw();
422
423             /*!
424               \param _pos position new position
425               \param _size new size
426
427               Set both position and size of the Widget regarding to the
428               \a parentWindow.
429               \note If entered size is bigger than \a GetMaxSize()
430               or smaller than \a GetMinSize(), it will crop the entered
431               value to the boundaries.
432
433               \exception ParentNotDefined is thrown had the widget not
                      been
434               assigned to any window. Use \a AddWidget.
435             */
436             virtual void SetGeometry(const Position& _pos, const Size&
                    _size)
437                 throw(ParentNotDefined);
438             /*!
439               \param _row new row position
440               \param _col new column position
441               \param _height new height
442               \param _width new width
443
444               Set both position and size of the Widget regarding to the
445               \a parentWindow.
446               \note If entered size is bigger than \a GetMaxSize()
447               or smaller than \a GetMinSize(), it will crop the entered
448               value to the boundaries.
449
450               \exception ParentNotDefined is thrown had the widget not
                      been
451               assigned to any window. Use \a AddWidget.
452             */
453             virtual void SetGeometry(Uint _row, Uint _col,
454                                      Uint _height, Uint _width)
455                 throw(ParentNotDefined);
456
457             /*!
458               \param _size new minimal size
459
```

```
460             Set minimal size of the Widget, \a minSize property.
461             \note If size is bigger than \a GetMaxSize(), it will
462             crop the entered value to the boundary.
463           */
464           virtual void SetMinSize(const Size& _size)throw();
465           /*!
466             \param _height new minimal height
467             \param _width new minimal width
468
469             Set minimal size of the Widget, \a minSize property.
470             \note If size is bigger than \a GetMaxSize(), it will
471             crop the entered value to the boundary.
472           */
473           virtual void SetMinSize(Uint _height, Uint _width)throw();
474           /*!
475             \return minimal size
476
477             Get minimal size of the Widget.
478           */
479           virtual const Size& GetMinSize() const throw();
480           /*!
481             \param _height new minimal height
482
483             Set minimal height of the Widget, \a minSize property.
484             \note If size is bigger than \a GetMaxSize(), it will
485             crop the entered value to the boundary.
486           */
487           virtual void SetMinHeight(Uint _height)throw();
488           /*!
489             \return minimal height
490
491             Get minimal height of the Widget.
492           */
493           virtual Uint GetMinHeight() const throw();
494           /*!
495             \param _width new minimal width
496
497             Set minimal width of the Widget, \a minSize property.
498             \note If size is bigger than \a GetMaxSize(), it will
499             crop the entered value to the boundary.
500           */
501           virtual void SetMinWidth(Uint _width)throw();
502           /*!
503             \return minimal width
504
505             Get minimal width of the Widget.
506           */
507           virtual Uint GetMinWidth() const throw();
508
509           /*!
510             \param _size new maximal size
511
512             Set maximal size of the Widget, \a minSize property.
513             \note If size is smaller than \a GetMinSize(), it will
514             crop the entered value to the boundary.
515           */
516           virtual void SetMaxSize(const Size& _size)throw();
517           /*!
518             \param _height new maximal height
519             \param _width new maximal width
520
521             Set maximal size of the Widget, \a minSize property.
```

```
522             \note If size is smaller than \a GetMinSize(), it will
523             crop the entered value to the boundary.
524           */
525           virtual void SetMaxSize(Uint _height, Uint _width)throw();
526           /*!
527             \return maximal size
528
529             Get maximal size of the Widget.
530           */
531           virtual const Size& GetMaxSize() const throw();
532           /*!
533             \param _height new maximal height
534
535             Set maximal height of the Widget, \a minSize property.
536             \note If size is smaller than \a GetMinSize(), it will
537             crop the entered value to the boundary.
538           */
539           virtual void SetMaxHeight(Uint _height)throw();
540           /*!
541             \return maximal height
542
543             Get maximal height of the Widget.
544           */
545           virtual Uint GetMaxHeight() const throw();
546           /*!
547             \param _width new maximal width
548
549             Set maximal width of the Widget, \a minSize property.
550             \note If size is smaller than \a GetMinSize(), it will
551             crop the entered value to the boundary.
552           */
553           virtual void SetMaxWidth(Uint _width)throw();
554           /*!
555             \return maximal width
556
557             Get maximal width of the Widget.
558           */
559           virtual Uint GetMaxWidth() const throw();
560
561
562           /*!
563             \param _policy new focus policy
564
565             Set focus policy.
566           */
567           virtual void SetFocusPolicy(FocusPolicy _policy)throw();
568
569           /*!
570             \return current focus policy
571
572             Get current focus policy.
573           */
574           virtual FocusPolicy GetFocusPolicy() const throw();
575
576           /*!
577             \param style
578
579             Set style.
580           */
581           virtual void SetStyle(const DisplayStyle& style
582                                 = DisplayStyle(Fg::System,Fg::Dark,
583                                                Bg::System))throw();
```

```
584
585             /*!
586               \return current style
587
588               Get style.
589             */
590             virtual const DisplayStyle& GetStyle() const throw();
591
592             /*!
593               \param _hidden new state value
594
595               Set the hidden state.
596             */
597             void SetHidden(bool _hidden)throw();
598             /*!
599               \return current hidden state
600             */
601             bool IsHidden() const throw();
602
603             virtual ~Widget()throw();
604
605             //! Object name. Used for style targetting.
606             std::string objectName;
607
608             RTTI_BASE(Widget);
609             /*! \typedef std::vector<std::string> ClassHierarchy;
610               Container holding the list of class names.
611             */
612             /*! \fn bool Widget::IsTypeOf(std::string _className) const
613               \param _className name of a class
614               \return whether the _className is in class hierarchy of this
                      ' class.
615             */
616             /*! \fn const char * Widget::TypeName() const
617               \return class name of this widget.
618             */
619             /*! \fn const char * Widget::ParentName() const
620               \return parent class of this widget.
621             */
622             /*! \fn const Widget::ClassHierarchy& Hierarchy()
623               \return class hierarchy of this widget.
624             */
625         }; // Widget
626     } // namespace Tk
627 } // namespace Scr
628
629 #include "stylesheet.h++"
630
631 #endif // __WIDGET_H__
```

2.26 include/rexio/tk/window.h++

```
1 ////////////////////////////////////////////////////////////////////
2 //
3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
4 //
5 // Permission is hereby granted, free of charge, to any person
```

```
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////////
27
28 #ifndef __WINDOW_H__
29 #define __WINDOW_H__
30
31 // TODO: Honor IsHidden() for widgets focus passing.
32
33 #include <list>
34 #include <rexio/tk/widget.h++>
35 #include <rexio/tk/autolist.h++>
36
37 #ifdef __GNUC__
38 #define HASH_MAP_NAMESPACE __gnu_cxx
39 #else
40 #define HASH_MAP_NAMESPACE std
41 #endif
42
43 namespace Scr
44 {
45     namespace Tk
46     {
47         class RootWindow;
48
49         const DisplayStyle WINDOW_DEFAULT_STYLE(WIDGET_DEFAULT_STYLE);
50
51         /*!
52           Window, a buffered ancestor of \a Widget. It can also group
53           other widgets and pass all the events down the path.
54           \sa \a WidgetGroup for an automated Widget grouping solution.
55         */
56         class Window : public Widget
57         {
58         protected:
59             /*!
60               Focuses on a next contained element that has a proper
61               \a focusPolicy.
62               Specifically, \a activeWidget iterator is incremented.
63             */
64             void NextWidget(); // activeWidget ++
65
66             /*!
67               Widget dedicated container.
```

```
68            */
69            typedef AutoList<Widget*> WidgetList;
70
71            /*!
72              Represensts all contained widgets, including subwindows.
73            */
74            WidgetList elements;
75            /*!
76              Currently active widget.
77            */
78            WidgetList::iterator activeWidget;
79
80            /*!
81              \return Screen handler reference.
82
83              Returns the top-level Screen handler.
84
85              \exception ParentNotDefined is thrown had the window
86              not been attached to any other.
87            */
88            virtual Screen& GetScreen()throw(ParentNotDefined);
89
90        public:
91            __DE(WidgetAlreadyAdded, Exception);
92            __DE(WidgetNotPresent, Exception);
93
94            /*!
95              Returns an absolute column the window is positioned
96              on a RootWindow
97
98              \exception ParentNotDefined is thrown had the window
99              not been attached to any other.
100           */
101           virtual Uint GetAbsoluteColumn()throw(ParentNotDefined);
102           /*!
103             Returns an absolute row the window is positioned
104             on a RootWindow
105
106             \exception ParentNotDefined is thrown had the window
107             not been attached to any other.
108           */
109           virtual Uint GetAbsoluteRow()throw(ParentNotDefined);
110
111           /*!
112             \param _height desired height
113             \param _width desired width
114             \param _style optional style
115           */
116           Window(Uint _height,
117                  Uint _width,
118                  const DisplayStyle& _style
119                  = DisplayStyle(Fg::White,Fg::Dark,Bg::Black))throw();
120           /*!
121             \copydoc Widget::SetStylesheet(Stylesheet*)
122             \a Window specific:
123             Recursively passes this call to all its children.
124           */
125           virtual void SetStylesheet(Stylesheet* _styleSheet)throw();
126
127           /*!
128             \param widget widget to attach to this window
129
```

```
130             Attach a widget to this window.
131             Specifically, add it to the \a elements.
132
133             \exception ParentAlreadySet is thrown if the widget
134             has already been attached to some other window.
135             \exception WidgetAlreadyAdded if the widget
136             is already attached to THIS window.
137           */
138           virtual void AddWidget(Widget& widget)
139               throw(ParentAlreadySet, WidgetAlreadyAdded);
140           /*!
141             \param widget widget to detach from this window
142
143             Detach a widget from this window.
144             Specifically, del it from the \a elements.
145
146             \exception WidgetNotPresent is thrown if the widget
147             is not attached to this window.
148           */
149           virtual void DelWidget(Widget& widget)throw(WidgetNotPresent);
150
151           /*!
152             \return RootWindow
153
154             \exception ParentNotDefined is thrown if the window hasn't
155             been attached to any other and thus is not in relation
156             with the root one.
157           */
158           virtual RootWindow& GetRootWindow()
159               throw(ParentNotDefined);
160
161
162           /*!
163             Need to redraw, pass the \a OnRedraw() event to all
164             contained widgets.
165           */
166           virtual void RedrawRequest()throw();
167           /*!
168             \param widget reference to widget which needs redrawing
169
170             Redraw one specific widget. Pass the \a OnRedraw() event
171             to it.
172           */
173           virtual void RedrawRequest(Widget& widget)throw();
174
175
176           virtual void OnFocus(FocusPolicy focustype)throw();
177           virtual void OnUnFocus(FocusPolicy focustype)throw();
178           /*!
179             \param focustype focus policy of this event
180
181             This event is triggered when containing event does want
182             to revoke its focus.
183           */
184           virtual void PassFocusRequest(FocusPolicy focustype)throw();
185
186           /*!
187             \param w widget to activate
188
189             Activates a given widget. Widget has to be directly
190             contained by this window.
191
```

```
192             \note Widget might directly revoke its activity.
193
194             \exception WidgetNotPresent is thrown if the widget
195             is not attached to this window.
196           */
197           virtual void SetActiveWidget(Widget &w)
198               throw(WidgetNotPresent);
199
200
201           /*!
202             \return reference to current active widget
203
204             \exception WidgetNotPresent is thrown if no widget is
                    currently
205             active.
206           */
207           virtual Widget& GetActiveWidget()const throw(WidgetNotPresent)
                ;
208
209           virtual void OnStart()throw();
210           virtual void OnResize()throw();
211           virtual void OnRedraw(Screen& screen)throw();
212           virtual void OnKeyDown(Key key)throw();
213
214           /*!
215             \param _size new size
216
217             Set size of the Window. Invoke \a OnResize() event
                    afterwards.
218             \note If entered size is bigger than \a GetMaxSize()
219             or smaller than \a GetMinSize(), it will crop the entered
220             value to the boundaries.
221             \note Since all the other size functions depend on this one,
222             all of them get the \a OnResize() event for free.
223           */
224           virtual void SetSize(const Size& _size)throw();
225
226           RTTI_OBJ(Window, Widget);
227       }; // Window
228   } // Tk
229 } // Scr
230 #endif // __WINDOW_H__
```

2.27 include/rexio/trace.h++

```
 1 ////////////////////////////////////////////////////////////////////////
 2 //
 3 // Copyright (c) 2007-2008 Damian Kaczmarek, Maciej Kaminski
 4 //
 5 // Permission is hereby granted, free of charge, to any person
 6 // obtaining a copy of this software and associated documentation
 7 // files (the "Software"), to deal in the Software without
 8 // restriction, including without limitation the rights to use,
 9 // copy, modify, merge, publish, distribute, sublicense, and/or sell
10 // copies of the Software, and to permit persons to whom the
11 // Software is furnished to do so, subject to the following
12 // conditions:
```

```
13 //
14 // The above copyright notice and this permission notice shall be
15 // included in all copies or substantial portions of the Software.
16 //
17 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
18 // EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
19 // OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
20 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
21 // HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
22 // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
23 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
24 // OTHER DEALINGS IN THE SOFTWARE.
25 //
26 ////////////////////////////////////////////////////////////////////
27
28 #ifndef __TRACE_H__
29 #define __TRACE_H__
30 #include <iostream>
31
32 #ifdef NDEBUG
33 #ifdef DEBUG
34 #undef DEBUG
35 #endif
36 #endif
37
38 #ifdef DEBUG
39 #ifndef LOGGING_LEVEL
40 #define LOGGING_LEVEL LogLevelModerate
41 #endif
42 #endif
43
44 namespace Scr
45 {
46     /*!
47      * Logging levels acceptable for RexIOLog macro
48      */
49     enum{LogLevelQuiet, LogLevelLow, LogLevelModerate,
50     LogLevelVerbose};
51 }
52
53 //!
54 //! RexIOLog macro prints log message if adequate log level is set. it
       does
55 //! not print anything otherwise. it is    primarily used for debugging
       RexIO
56 //! library itself.
57 //!
58 #ifdef DEBUG
59 #define RexIOLog(level) if(level<=LOGGING_LEVEL) std::clog
60 #else
61 #define RexIOLog(level) if (0) std::clog
62 #endif
63
64 #endif
```

2.28 include/rexio/fileno_hack.h++ by Richard B. Kreckel

```
#ifndef __FILENO_HACK__
#define __FILENO_HACK__

/*! \file fileno_hack.h++
  \brief extract file descriptor from C++ stream.
  Author of this code is Richard B. Kreckel
*/

//#include "fileno.hpp"
#include <cstdio>  // declaration of ::fileno
#include <fstream>  // for basic_filebuf template
#include <cerrno>

#if defined(__GLIBCXX__) || (defined(__GLIBCPP__) && __GLIBCPP__
    >=20020514)  // GCC >= 3.1.0
# include <ext/stdio_filebuf.h>
#endif
#if defined(__GLIBCXX__) // GCC >= 3.4.0
# include <ext/stdio_sync_filebuf.h>
#endif

//!
//! \param stream a C++-style stream to extract FD from
//! \return  The integer file descriptor associated with the stream, or -1
     if
//!   that stream is invalid.  In the latter case, for the sake of keeping
     the
//!   code as similar to fileno(3), errno is set to EBADF.
//! \see  The <A HREF="http://www.ginac.de/~kreckel/fileno/">upstream page
     at
//!   http://www.ginac.de/~kreckel/fileno/</A> of this code provides more
//!   detailed information.
//!
//! Similar to fileno(3), but taking a C++ stream as argument instead of a
//! FILE*.  Note that there is no way for the library to track what you do
     with
//! the descriptor, so be careful.
template <typename charT, typename traits>
inline int
fileno_hack(const std::basic_ios<charT, traits>& stream)
{
    // Some C++ runtime libraries shipped with ancient GCC, Sun Pro,
    // Sun WS/Forte 5/6, Compaq C++ supported non-standard file descriptor
    // access basic_filebuf<>::fd().  Alas, starting from GCC 3.1, the GNU
         C++
    // runtime removes all non-standard std::filebuf methods and provides
        an
    // extension template class __gnu_cxx::stdio_filebuf on all systems
        where
    // that appears to make sense (i.e. at least all Unix systems).
        Starting
    // from GCC 3.4, there is an __gnu_cxx::stdio_sync_filebuf, in
        addition.
    // Sorry, darling, I must get brutal to fetch the darn file descriptor
        !
    // Please complain to your compiler/libstdc++ vendor...
#if defined(__GLIBCXX__) || defined(__GLIBCPP__)
    // OK, stop reading here, because it's getting obscene.  Cross fingers
        !
# if defined(__GLIBCXX__)  // >= GCC 3.4.0
    // This applies to cin, cout and cerr when not synced with stdio:
    typedef __gnu_cxx::stdio_filebuf<charT, traits> unix_filebuf_t;
```

```
51      unix_filebuf_t* fbuf = dynamic_cast<unix_filebuf_t*>(stream.rdbuf());
52      if (fbuf != NULL) {
53          return fbuf->fd();
54      }
55
56      // This applies to filestreams:
57      typedef std::basic_filebuf<charT, traits> filebuf_t;
58      filebuf_t* bbuf = dynamic_cast<filebuf_t*>(stream.rdbuf());
59      if (bbuf != NULL) {
60          // This subclass is only there for accessing the FILE*.  Ouuwww,
                sucks!
61          struct my_filebuf : public std::basic_filebuf<charT, traits> {
62              int fd() { return this->_M_file.fd(); }
63          };
64          return static_cast<my_filebuf*>(bbuf)->fd();
65      }
66
67      // This applies to cin, cout and cerr when synced with stdio:
68      typedef __gnu_cxx::stdio_sync_filebuf<charT, traits> sync_filebuf_t;
69      sync_filebuf_t* sbuf = dynamic_cast<sync_filebuf_t*>(stream.rdbuf());
70      if (sbuf != NULL) {
71  #  if (__GLIBCXX__<20040906) // GCC < 3.4.2
72          // This subclass is only there for accessing the FILE*.
73          // See GCC PR#14600 and PR#16411.
74          struct my_filebuf : public sync_filebuf_t {
75              my_filebuf();  // Dummy ctor keeps the compiler happy.
76              // Note: stdio_sync_filebuf has a FILE* as its first (but
                    private)
77              // member variable.  However, it is derived from
                    basic_streambuf<>
78              // and the FILE* is the first non-inherited member variable.
79              FILE* c_file() {
80                  return *(FILE**)((char*)this + sizeof(std::basic_streambuf
                        <charT, traits>));
81              }
82          };
83          return ::fileno(static_cast<my_filebuf*>(sbuf)->c_file());
84  #  else
85          return ::fileno(sbuf->file());
86  #  endif
87      }
88  # else  // GCC < 3.4.0 used __GLIBCPP__
89  #  if (__GLIBCPP__>=20020514)  // GCC >= 3.1.0
90      // This applies to cin, cout and cerr:
91      typedef __gnu_cxx::stdio_filebuf<charT, traits> unix_filebuf_t;
92      unix_filebuf_t* buf = dynamic_cast<unix_filebuf_t*>(stream.rdbuf());
93      if (buf != NULL) {
94          return buf->fd();
95      }
96
97      // This applies to filestreams:
98      typedef std::basic_filebuf<charT, traits> filebuf_t;
99      filebuf_t* bbuf = dynamic_cast<filebuf_t*>(stream.rdbuf());
100     if (bbuf != NULL) {
101         // This subclass is only there for accessing the FILE*.  Ouuwww,
                sucks!
102         struct my_filebuf : public std::basic_filebuf<charT, traits> {
103             // Note: _M_file is of type __basic_file<char> which has a
104             // FILE* as its first (but private) member variable.
105             FILE* c_file() { return *(FILE**)(&this->_M_file); }
106         };
107         FILE* c_file = static_cast<my_filebuf*>(bbuf)->c_file();
```

```
108         if (c_file != NULL) {  // Could be NULL for failed ifstreams.
109             return ::fileno(c_file);
110         }
111     }
112 #  else  // GCC 3.0.x
113     typedef std::basic_filebuf<charT, traits> filebuf_t;
114     filebuf_t* fbuf = dynamic_cast<filebuf_t*>(stream.rdbuf());
115     if (fbuf != NULL) {
116         struct my_filebuf : public filebuf_t {
117             // Note: basic_filebuf<charT, traits> has a __basic_file<charT
                >* as
118             // its first (but private) member variable.  Since it is
                derived
119             // from basic_streambuf<charT, traits> we can guess its offset
                .
120             // __basic_file<charT> in turn has a FILE* as its first (but
121             // private) member variable.  Get it by brute force.  Oh, geez
                !
122             FILE* c_file() {
123                 std::__basic_file<charT>* ptr_M_file = *(std::__basic_file
                    <charT>**)((char*)this + sizeof(std::basic_streambuf<
                    charT, traits>));
124 #  if _GLIBCPP_BASIC_FILE_INHERITANCE
125                 // __basic_file<charT> inherits from __basic_file_base<
                    charT>
126                 return *(FILE**)((char*)ptr_M_file + sizeof(std::
                    __basic_file_base<charT>));
127 #  else
128                 // __basic_file<charT> is base class, but with vptr.
129                 return *(FILE**)((char*)ptr_M_file + sizeof(void*));
130 #  endif
131             }
132         };
133         return ::fileno(static_cast<my_filebuf*>(fbuf)->c_file());
134     }
135 #  endif
136 # endif
137 #else
138 #  error "Does anybody know how to fetch the bloody file descriptor?"
139     return stream.rdbuf()->fd();  // Maybe a good start?
140 #endif
141     errno = EBADF;
142     return -1;
143 }
144
145 // //! 8-Bit character instantiation: fileno(ios).
146 // template <>
147 // int
148 // fileno<char>(const std::ios& stream)
149 // {
150 //     return fileno_hack(stream);
151 // }
152
153 // #if !(defined(__GLIBCXX__) || defined(__GLIBCPP__)) || (defined(
       _GLIBCPP_USE_WCHAR_T) || defined(_GLIBCXX_USE_WCHAR_T))
154 // //! Wide character instantiation: fileno(wios).
155 // template <>
156 // int
157 // fileno<wchar_t>(const std::wios& stream)
158 // {
159 //     return fileno_hack(stream);
160 // }
```

```
161 // #endif
162
163 #endif /* __FILENO_HACK__ */
```

3 Header files of internal implementation details

3.1 lib/screen/include/bufferedinput.h++

```
1 #ifndef __BUFFERED_INPUT_H__
2 #define __BUFFERED_INPUT_H__
3 #include "keyboard.h++"
4
5 #include"fileno_hack.h++"
6 #include"throw.h++"
7 #include"commons.h++"
8 #include<queue>
9 #include"screen.h++"
10
11 namespace Scr
12 {
13
14     //!Intermediate between Scr::__ScreenConnection and std::istream
15     class BufferedInput
16     {
17     private:
18         static const Uint maxCharBufferSize = 32;
19         /*!
20           if after last read buffer was filled while still something
21           on input was availble
22          */
23         mutable bool filledToCapacity;
24         /*!
25           number of characters staying in buffer after last read.
26          */
27         mutable Uint currentCharBufferSize;
28
29         /*!
30           idx of current character
31          */
32         mutable Uint currentCharBufferIndex;
33         /*!
34           read some bytes from input, then transform em to keyboard
35           events (no direct access to istream outside of
36           ProcessConnection, where readsome() performed - to ensure )
37         */
38         mutable char charBuffer[maxCharBufferSize];
39         /*!
40           input stream
41         */
42         mutable std::istream & stream;
43
44         mutable std::queue<char> * q;
45
```

```
46          /*!
47            std::istream::readsome returned 0, while something needs to
48            be read.
49           */
50          void ForceBuffer()const throw();
51
52          void DoBuffer() const throw();
53      public:
54          __DE(BufferEmpty,Exception);
55
56          bool KbHit()const throw();
57          /*!
58            \param _stream stream to be contained
59           */
60          explicit BufferedInput(std::istream & _stream)throw()
61              :filledToCapacity(false),
62               currentCharBufferSize(1),
63               currentCharBufferIndex(1),
64               stream(_stream),
65               q(NULL)
66               {;}
67
68          /*!
69            Save some characters in internal buffer (it is not invoked
70            automatically when Get() is called and buffer is empty.
71           */
72          void Buffer()throw() {DoBuffer();}
73
74          /*!
75            Inquiry if object has some buffered text, or at least can
76            make this text availble instantly
77           */
78          bool HasBufferedText()const throw()
79              {
80                  if (currentCharBufferIndex <  currentCharBufferSize )
81                      return true;
82                  if (currentCharBufferIndex == currentCharBufferSize )
83                  {
84                      if (filledToCapacity)
85                      {
86                          DoBuffer();
87                          return true;
88                      }
89                      else
90                          return false;
91                  }
92                  THROW(FatalException);// kill app (assume that it is
                        impossible)
93              }
94
95          /*!
96            Peek if it won't block app
97           */
98          unsigned char TryPeek()const throw(BufferEmpty)
99              {
100                 EASSERT(HasBufferedText(),
101                         BufferEmpty);
102                 RexIOLog(LogLevelModerate) <<  "TryPeek: " <<
103                     static_cast<int>(charBuffer[currentCharBufferIndex])
                        <<'\n';
104                 return charBuffer[currentCharBufferIndex];
105             };
```

```
106
107         /*!
108           Get if it won't block app (throw exception otherwise)
109          */
110         unsigned char TryGet()throw(BufferEmpty)
111             {
112                 EASSERT(HasBufferedText(),
113                         BufferEmpty);
114                 RexIOLog(LogLevelModerate) <<  "TryGet: " <<
115                     static_cast<int>(charBuffer[currentCharBufferIndex])
                            <<'\n';
116                 return charBuffer[currentCharBufferIndex++];
117             };
118
119         /*!
120           Current character. The same will be availble after call to
121           this func.
122          */
123         unsigned char Peek() const throw()
124             {
125                 EASSERT(currentCharBufferIndex <= currentCharBufferSize,
126                         FatalException);
127                 if (currentCharBufferIndex == currentCharBufferSize)
128                     DoBuffer();
129                 RexIOLog(LogLevelModerate) <<  "Peek: " <<
130                     static_cast<int>(charBuffer[currentCharBufferIndex])
                            <<'\n';
131                 return charBuffer[currentCharBufferIndex];
132             };
133
134         /*!
135           Get character from stream
136          */
137         unsigned char Get()throw()
138             {
139                 EASSERT(currentCharBufferIndex <= currentCharBufferSize,
140                         FatalException);
141                 if (currentCharBufferIndex == currentCharBufferSize)
142                     Buffer();
143                 RexIOLog(LogLevelModerate) <<  "Get: " <<
144                     static_cast<int>(charBuffer[currentCharBufferIndex])
                            <<'\n';
145                 return charBuffer[currentCharBufferIndex++];
146             };
147
148         /*!
149           UnGet().
150
151           \note that this function may fail if called just after
152           buffering, or called too frequently: only one successful
153           UnGet per one Get is guaranteed.
154
155           \exception Scr::BufferedString::BufferEmpty is thrown when
156           too many UnGet's are processed oneafter another
157          */
158         void UnGet()throw(BufferEmpty)
159             {
160                 EASSERT(currentCharBufferIndex > 0,
161                         BufferEmpty);
162                 currentCharBufferIndex--;
163             };
164
```

```
165         /*!
166           Unix style file descriptor
167          */
168         int FD()const throw()
169             {
170                 return fileno_hack(stream);
171             }
172 
173         /*!
174           direct access to underlying std::stream - const version
175          */
176         const std::istream & Stream()const throw()
177             {
178                 return stream;
179             }
180         /*!
181           direct access to underlying std::stream
182          */
183         std::istream & Stream()throw()
184             {
185                 return stream;
186             }
187         /*!
188           contents of internal buffer as string
189          */
190         std::string String()throw();
191 
192         /*!
193           more than info returned by String(): function created
194           specifically for debugging/logging purposes
195         */
196         std::string DebugInfo()throw();
197 
198         /*!
199           more than info returned by String(): function created
200           specifically for debugging/logging purposes
201         */
202         const std::string DebugInfo()const throw();
203 
204         void SyncWithQueue(std::queue<char> & _q)throw()
205             {
206                 q=&_q;
207             }
208     };
209 
210 
211 }
212 #endif
```

3.2 lib/screen/include/connection.h++

```
1 #ifndef __CONNECTION_H__
2 #define __CONNECTION_H__
3 #include "keyboard.h++"
4 
5 #include"fileno_hack.h++"
6 
```

```
 7 namespace Scr
 8 {
 9
10     //!Intermediate between Scr::__ScreenConnection and std::istream
11     class BufferedInput
12     {
13     private:
14         static const Uint maxCharBufferSize = 32;
15         Uint currentCharBufferSize;
16         Uint currentCharBufferIndex;
17         /*!
18           read some bytes from input, then transform em to keyboard
19           events (no direct access to istream outside of
20           ProcessConnection, where readsome() performed - to ensure )
21         */
22         char charBuffer[maxCharBufferSize];
23         /*!
24           input stream
25         */
26         std::istream & stream;
27         /*!
28          * Blocking buffering function
29          */
30         void ForceBuffer()throw();
31     public:
32         __DE(BufferEmpty,Exception);
33         /*!
34          * \return true if input device is ready to transmit data
35          */
36         bool KbHit()throw();
37         /*!
38           \param _stream stream to be contained
39          */
40         explicit BufferedInput(std::istream & _stream)throw()
41             :
42              currentCharBufferSize(0),
43              currentCharBufferIndex(0),
44              stream(_stream)
45              {;}
46
47         /*!
48           Save some characters in internal buffer (it is not invoked
49           automatically when Get() is called and buffer is empty.
50          */
51         void Buffer()throw()
52             {
53                 currentCharBufferIndex=0;
54                 currentCharBufferSize=
55                     stream.readsome(
56                         static_cast<char*>(&charBuffer[0]),
57                         static_cast<std::streamsize>(maxCharBufferSize));
58                 if (currentCharBufferSize == 0)// no text read, while it
59                                                // has to be read.
60                     ForceBuffer();
61             }
62
63         /*!
64          * \return true if any character is availble in buffer
65          */
66         bool HasBufferedText()throw()
67             {
68                 if (currentCharBufferIndex <  currentCharBufferSize )
```

```
69                      return true;
70                  if (currentCharBufferIndex == currentCharBufferSize )
71                      return false;
72                  THROW(FatalException);// kill app (assume that it is
                        impossible)
73              }
74
75          /*!
76           * \return first availble character w/o moving pointer
77           */
78          unsigned char Peek()throw(BufferEmpty)
79              {
80                  EASSERT(currentCharBufferIndex < currentCharBufferSize,
81                          BufferEmpty);
82                  if (currentCharBufferIndex == currentCharBufferSize)
83                      Buffer();
84  //              std::RexIOLog(LogLevelModerate) <<  "Peek: " <<
85  //                  static_cast<int>(charBuffer[currentCharBufferIndex])
        <<'\n';
86                  return charBuffer[currentCharBufferIndex];
87              };
88
89          /*!
90           * get character
91           */
92          unsigned char Get()throw(BufferEmpty)
93              {
94                  EASSERT(currentCharBufferIndex < currentCharBufferSize,
95                          BufferEmpty);
96                  if (currentCharBufferIndex == currentCharBufferSize)
97                      Buffer();
98                  std::RexIOLog(LogLevelModerate) <<  "Get: " <<
99                      static_cast<int>(charBuffer[currentCharBufferIndex])
                            <<'\n';
100                 return charBuffer[currentCharBufferIndex++];
101             };
102
103         /*!
104          * \return character to buffer
105          */
106         void UnGet()throw()
107             {
108                 EASSERT(currentCharBufferIndex > 0,
109                         BufferEmpty);
110                 currentCharBufferIndex--;
111             };
112
113         /*!
114          * \return UNIX* file descriptor for associated stream.
115          *
116          * * UNIX is registered trademark of AT&T and OpenGroup.
117          */
118         int FD()throw()
119             {
120                 return fileno_hack(stream);
121             }
122
123         /*!
124          * \return associated C++ stream.
125          */
126         std::istream & Stream()throw()
127             {
```

```
128                 return stream;
129             }
130     };
131
132     //! \brief internal class which is base for all connection-specific
133     //! implementations of screen (multiple-inheritance case)
134     /*!
135       It represents internal interface between Scr::Connection and
136       Scr::Screen classes.
137     */
138     class __ScreenConnection
139     {
140     protected:
141         /*!
142           ProcessConnection will return this value upon successful finish
143         */
144         int exitcode;
145         /*!
146           is application running? does it have to stop?
147           (ExitConnection() is called by Connection::Exit(int), sets
148           exit code and breaks main loop performed in ProcessConnection)
149         */
150         Connection & connection;
151
152         /*!
153           break main loop if set to false
154          */
155         bool active;
156
157         BufferedInput input;
158         /*!
159           get key esc-code from std input stream. decode it into form
160           from keyboard.h++
161         */
162         virtual Key DecodeKeyPressed();
163
164     public:
165         /*!
166
167           \param _input input stream (used to capture some of events,
168           in particular keyboard events).
169           \param _connection newely estabilished connection to serve
170         */
171         __ScreenConnection(Connection & _connection,std::istream & _input)
                throw();
172         /*!
173
174           \return value of exitcode, as it was in the moment of
175           connection termination if successful.
176
177           Initialize, conduct and end connection in way apropriate to
178           connection type, operating system etc. Inform
179           Scr::Connection object supplied about all captured events
180
181           \note as function (for design reasons) lacks exception-set
182           specification, it may throw any exceptions, but it is
183           recommended, that only exceptions typical to
184           Scr::Connection::Start() will be thrown.
185
186         */
187         virtual int  ProcessConnection() =0;
188
```

```
189         /*!
190           \param _code exit code return from ProcessConnection after
191           successfully finished connection
192
193           Force stopping connection as soon as possible
194           \note as function (for design reasons) lacks exception-set
195           specification, it may throw any exceptions, but it is
196           recommended, that only exceptions typical to
197           Scr::Connection::Exit() will be thrown.
198         */
199         virtual void ExitConnection(int _code);
200         virtual ~__ScreenConnection()throw();
201     };
202 }
203 #endif
```

3.3 lib/screen/include/core.h++

```
 1 #ifndef __CORE_H__
 2 #define __CORE_H__
 3
 4 #include <iostream>
 5
 6 namespace Scr
 7 {
 8     //! \brief template class representing full implementation of
 9     //! Scr::Screen and Scr::__ScreenConnection
10     /*!
11       \param LOCATION local, telnet etc ..
12       \param TYPE frameless VT100-like, UTF8, Windows....
13
14       See figure attached to Scr:: namespace description for more details
15     */
16     template<class LOCATION,class TYPE>
17     class RScreen : public LOCATION, public TYPE
18     {
19     public:
20     RScreen(Connection & _connection,std::istream& _input, std::ostream&
           _output)throw()
21             : GenericScreen(_input,_output),
22               LOCATION(_connection,_input,_output),
23               TYPE(_input,_output)
24             {;}
25         ~RScreen()throw(){;}
26     };
27
28 }
29 #endif /* __CORE_H__ */
```

3.4 lib/screen/include/dictionary.h++

```
1 #ifndef __DICTIONARY_H__
```

```
 2 #define __DICTIONARY_H__
 3 #include <cstring>
 4 #include <memory>
 5 #include <iostream>
 6 #include <cstdlib>
 7 #include <exception>
 8 #include "throw.h++"
 9 #include "commons.h++"
10 namespace Scr
11 {
12
13     /*!
14      * \brief replacement of std::map<std::string,T> - optimized for
            string key random access
15      * using dictionary-tree data structure.
16      *
17      * Member functions are named in C++ library convention, that is
18      * w/ underscore and w/o capital letters.
19      * \note this class is not STL compatible. it is only STL-like.
20      */
21     template<typename T>
22     class Dictionary
23     {
24     protected:
25         static const int T_VECTOR = 2;
26         static const int T_RECORD = 4;
27         static const size_t num_characters = 256;
28
29         struct t_name_vector;
30         //! tree leaf (node containing just one pc. of information
31         struct t_name_record
32         {
33             int type;//!< magic value to test, whenever it is a vector or
                    a record
34             t_name_vector * parent;
35             char * name;//!< key itself
36             int num_occurrences;//!< number of occurences of specific key
37             T * datafield;//! datafield
38         } ;
39
40         //! node containing references to other nodes
41         struct t_name_vector
42         {
43             int type;//!< magic value to test, whenever it is a vector or
                    a record
44             t_name_vector * parent;
45             t_name_record * vector[num_characters];/*'\0' and 'A' to 'Z'*/
46         } ;
47
48         //! core information block (one per Dictionary)
49         typedef struct
50         {
51             int max_num_occurrences;//!< greatest recorded number of
                    occurrences
52             t_name_vector * first_vector;//!< first vector
53         } t_tree;
54
55     private:
56         t_tree tree; /*tree-like structure*/
57     protected:
58 #define vmark(x) (x)
59
```

```
60 #define record vector[ vmark( static_cast<size_t>(name[current]) ) ]
61         /*!
62          * add node to tree.
63          *
64          * \param name name associated w/ node
65          * \return pointer to new node (or NULL if adding it was
                unsuccessful)
66          * \exception std::bad_alloc if memory allocation failed
67          */
68         t_name_record * tree_add (const char * name);
69
70         /*!
71          * Attempts to search for a specific node. Doesn't modify tree (
                doesn't
72          * new node if search failed).
73          *
74          * \return Tf argument matches beginning of more than
75          * node key, t_name_vector is really returned (what may be
                detected
76          * by testing type member field), even if one of theese nodes
                matches
77          * completely. If nothing matches, 0 is returned. Otherwise ptr to
                 record is
78          * returned
79          *
80          * \param name key to look for
81          * \param current_vector where to start search
82          * \param current assume current depth in tree (start matching
                from this
83          *  character of name)
84          */
85         t_name_record * tree_partial_find (const char * name,
86                 t_name_vector * current_vector,  size_t current = 0) const
                    ;
87
88         /*!
89          * \return pointer to specific node if it exists, NULL otherwise.
                this
90          * function depends on tree_partial_search
91          *
92          * \param name key to look for
93          * \param current_vector where to look for
94          * \param current assume current depth in tree (start matching
                from this
95          *  character of name)
96          */
97         t_name_record * tree_find (const char * name,
98                 t_name_vector * current_vector, int current = 0) const;
99
100        /*!
101         * Find next node. If r points to vector, find it's first node.
102         * If nothing found, return 0.
103         *
104         * \param r record
105         */
106        static t_name_record * tree_find_next (t_name_record * r);
107
108
109        /*!
110         * erase record
111         *
112         * \param r record to be erased
```

```
113          */
114         static void tree_erase_record (t_name_record * r)
115         {
116             delete r->datafield;
117             free (r->name);
118             free (r);
119         }
120 
121         /*!
122          * erase vector
123          *
124          * \param v pointer to vector, that will be erased
125          *
126          * \note function not only recursively erases contents of vector,
                but
127          *  also erases vector itself
128          */
129         static void tree_erase_vector (t_name_vector * v);
130     public:
131 
132         //! \a iterator class for Dictionary
133         class iterator
134         {
135         private:
136             t_name_record * _node;
137 
138         public:
139             /*!
140              * Default constructor returns iterator, that equals end()
141              */
142             iterator ()
143             {
144                 _node = 0; // 0 is special code for end, as no node of
                        this address exists
145             }
146 
147             /*!
148              * copy constructor
149              *
150              * \param it base of construction
151              */
152             iterator (const iterator & it)
153             {
154                 _node = it._node;
155             }
156 
157         protected:
158 
159             /*!
160              * Constructor initialized w/ raw data node pointer (
                    t_name_record)
161              * is accessed by functions such as begin(), end() or find().
162              *
163              * \param __node node in tree mapped to this iterator
164              */
165             explicit iterator (t_name_record * __node)
166             {
167                 _node = __node;
168             }
169 
170         public:
171 
```

```
172             //! result of validity_test
173             enum validity
174             {   //! dereference (indirection) possible, iterator points to
175                 //!   single data object
176                 VALID,
177                 //! unique key, but no data object (dereference WILL fail)
178                 INVALID,
179                 //! not unique key: dreference WILL fail
180                 NOT_UNIQUE,
181                 //! end(): dreference WILL fail
182                 END
183             };
184
185             /*!
186              * Tests if iterator is valid. If it is VALID is returned. if
                    it is not,
187              * function says why
188              */
189             validity validity_test ()
190             {
191                 if (_node not_eq NULL)
192                 {
193                     if (_node->type==T_RECORD)
194                     {
195                         if  (_node->datafield)
196                             return VALID;
197                         else
198                             return INVALID;
199                     }
200                     else
201                         return NOT_UNIQUE;
202                 }
203                 else
204                     return END;
205             }//iterator::validity_test
206
207             /*!
208              * tests if iterator is valid
209              */
210             bool valid ()
211             {
212                 return validity_test ()==VALID;
213             }//iterator::valid
214
215             /*!
216              * Indirection operator returns reference to object
217              * \throw std::bad_exception happens when iterator is not
                    unique
218              */
219             T& operator* ()
220             {
221                 using namespace std;
222                 RexIOLog(LogLevelModerate) << "Dereferencing " << _node->
                        name << endl;
223                 switch(validity_test ())
224                 {
225                     case INVALID:
226                         _node->datafield = new T (); // make it valid, and
227                         // process it as if it was valid
228                     case VALID:
229                         return  * _node->datafield;
230
```

```
231                 default:;
232             }
233             throw std::bad_exception ();
234 
235         }// iterator::operator*
236 
237         /*!
238          * Indirection-and-element-access operator returns reference
                to object
239          * \throw std::bad_exception happens when iterator is not
                unique
240          */
241         T* operator -> ()
242         {
243             switch(validity_test ())
244             {
245                 case INVALID:
246                     _node->datafield = new T (); // make it valid, and
247                     // process it as if it was valid
248                 case VALID:
249                     return  * _node->datafield;
250                 case NOT_UNIQUE:
251                     THROW (LogicError);
252             }
253         }// iterator::operator ->
254 
255         /*!
256          * Assignment operator
257          *
258          * \param it other iterator
259          */
260         iterator & operator=(const iterator & it)
261         {
262             _node = it._node;
263             return *this;
264         }// iterator::operator=
265 
266         /*!
267          * Comparison operator
268          *
269          * \param it other iterator
270          */
271         bool operator==(const iterator & it)
272         {
273             return _node == it._node;
274         }// iterator::operator==
275 
276         /*!
277          * Comparison operator
278          *
279          * \param it other iterator
280          */
281         bool operator!=(const iterator & it)
282         {
283             return _node != it._node;
284         }// iterator::operator!=
285 
286         /*!
287          * tricky comparison operator comparing lexicographically w/
                other key
288          * \param it other iterator
289          */
```

```
290             bool operator<(const iterator & it)
291             {
292                 if (*this == it)
293                     return false;
294                 else
295                 {
296                     if (it._node==NULL)//everything is < end()
297                         return true;
298                     else if (_node==NULL)
299                         return false;
300                     else if (valid () and it.valid ())
301                         return strcmp (_node->name, it.node->name)<0;
302                     else
303                         return false;// any of operands is invalid;
304                 }
305             }//iterator::operator<
306 
307             /*!
308              * incrementation operator finds new element
309              */
310             iterator & operator++()
311             {
312                 _node = tree_find_next (_node);
313                 return *this;
314             }
315 
316             friend class Dictionary<T>;
317         };// class iterator
318 
319         iterator end ()
320         {
321             return iterator ();
322         }
323 
324         iterator begin ()
325         {
326             return iterator (tree_find_next (
327                     reinterpret_cast<t_name_record *>(
328                     tree.first_vector)));
329         }
330         /*
331          * Add element into field w/ selected key. if any element was
                already placed
332          * there, it will be replaced w/ new one
333          * \param name key
334          * \param n new element
335          */
336         void insert (const char * name, const T & n)
337         {
338             t_name_record * r = tree_add (name);
339             if (r->datafield == NULL)
340             {
341                 r->datafield = new T (n);
342             }
343             else
344             {
345                 * ( r->datafield ) = n;
346             }
347         }// insert
348 
349         /*
```

```
350          * return iterator to specific node. If failed, return iterator to
                 end()
351          * \note consider using contains(const char *), when need only to
                 check,
352          * whenever element  exists.
353          */
354         iterator find (const char * name)
355         {
356             // what happens when name == 0 ??
357             return iterator (tree_find (name, tree.first_vector));
358         }
359
360         iterator partial_find (const char * name)
361         {
362             return iterator (tree_partial_find (name, tree.first_vector));
363         }
364         /*
365          * Nonstandard function tests if container contains specific key
366          *  (this equals "find(name)!=end()")
367          * \param name key
368          */
369         bool contains (const char * name)
370         {
371             return tree_find (name, tree.first_vector) not_eq 0;
372         }// contains
373
374         /*
375          * Default constructor
376          * \exceptions std::bad_alloc when memory allocation fails
377          */
378         Dictionary<T> ()
379         {
380             using namespace std;
381             tree.max_num_occurrences=1;
382             tree.first_vector=(t_name_vector*)calloc (1, sizeof(
                    t_name_vector));
383             if (!tree.first_vector)
384                 throw bad_alloc ();// memory allocationfailed
385             tree.first_vector->type=T_VECTOR;
386         }
387         ~Dictionary<T>()
388         {
389             tree_erase_vector (tree.first_vector);
390         }
391
392     };// class Dictionary
393
394     //Add node to tree (more documentation is placed with declaration)
395     template<typename T>
396         typename Dictionary<T>::t_name_record *
397             Dictionary<T>::tree_add (const char * name)
398     {
399         using namespace std;
400         int current = 0;
401         t_name_vector * current_vector ;
402         current_vector = tree.first_vector;
403         while (1)
404         {
405             if (current_vector->record == 0)
406             {
407                 /*create record */
408                 current_vector -> record = (t_name_record *)
```

```
409                     calloc (1, sizeof(t_name_record));
410             if (! current_vector -> record )
411                 throw bad_alloc (); // C++ standard exception upon
                        memory
412             //  allocation failure
413             current_vector -> record -> name =
414                     (char*) malloc (1+strlen (name));
415             if (!current_vector -> record -> name)
416                 throw bad_alloc ();
417             sprintf (current_vector -> record -> name, name);
418             current_vector -> record -> type = T_RECORD;
419             current_vector -> record -> num_occurrences = 1;
420             current_vector -> record -> parent = current_vector;
421             return current_vector -> record;
422         }
423         else if (current_vector->record->type==T_RECORD)
424         {
425             if (0==strcmp (current_vector->record ->name, name))
426             {
427                 /*update record (already exists)*/
428                 current_vector -> record -> num_occurrences ++;
429                 tree.max_num_occurrences=
430                         max (tree.max_num_occurrences,
431                         current_vector -> record -> num_occurrences);
432                 return current_vector -> record;
433             }
434             else
435             {   /*name differs: must reorganize structure.*/
436                 /*replace record (leaf) with new vector (of leaves)*/
437                 t_name_vector * new_vector ;
438                 new_vector =
439                         (t_name_vector*)calloc (1, sizeof(
                                t_name_vector));
440                 if (!new_vector)
441                     throw bad_alloc ();
442                 new_vector->type=T_VECTOR;
443                 new_vector->parent=current_vector;
444                 current_vector->record->parent=new_vector;
445
446                 new_vector->vector[ vmark (
447                         static_cast<size_t>(current_vector->record ->
448                         name[1+current]))]=current_vector->record;
449                 current_vector -> record = (t_name_record*)new_vector;
450                 current++;
451                 current_vector = new_vector;
452             }
453         }
454         else
455         {
456             /*go into tree*/
457             current_vector = reinterpret_cast<t_name_vector *>
458                     (current_vector->record);
459             current++;
460         }
461     }
462     THROW (LogicError); /*function shouldn't go here*/
463 }//tree_add
464
465 template<typename T>
466     typename Dictionary<T>::t_name_record *
467         Dictionary<T>::tree_partial_find
468             (const char * name, t_name_vector * current_vector,
```

```
469                     size_t current) const
470     {
471         using namespace std;
472         size_t sl = strlen (name);
473         while (1)
474         {
475             if (current_vector->record == 0)// key does not exist
476             {
477                 return 0;
478             }
479             else if (current_vector->record->type==T_RECORD)
480             {
481                 if (0==strcmp (current_vector->record ->name, name))
482                     return current_vector->record;
483                 else
484                     return 0;
485             }
486             else //if (current_vector->record->type==T_VECTOR)
487             {
488                 current_vector = (t_name_vector *) current_vector->record;
489                 current++;
490                 if (current==sl)// non unique match (even complete)
                        returns vec
491                 {
492                     return reinterpret_cast<t_name_record *>
493                             (current_vector);
494                 }
495             }
496         }
497     }//tree_partial_find
498
499     template<typename T>
500         typename Dictionary<T>::t_name_record *
501             Dictionary<T>::tree_find (const char * name,
502                 t_name_vector * current_vector, int current)
503                     const
504     {
505         t_name_record * r = tree_partial_find (name, tree.first_vector,
                current);
506         if (not r)
507             return 0;
508         else
509         {
510             if (r->type==T_RECORD)
511                 return r;
512             else
513             {
514                 r =
515                         reinterpret_cast<t_name_vector *>(r) -> vector[
                                vmark (0)];
516                 if (r == NULL)
517                     return NULL; // no such record
518                 else if (r->type == T_VECTOR)
519                     return NULL; // begining of name OK, but it is not the
                            same.
520                 else
521                 {// note: this test is unnecessary and exceprion will
                        never be
522                     // thrown. this serves rather as invariant assertion
523                     if (0 == strcmp (r->name, name))
524                     {
525                         return r;
```

```
526                     }
527                     else
528                         THROW (LogicError);
529                 }
530             }
531         }
532     }//tree_find
533
534     template<typename T>
535         typename Dictionary<T>::t_name_record *
536             Dictionary<T>::tree_find_next (t_name_record * r)
537     {
538         size_t i = 0;
539         do
540         {
541             if (r->type==T_VECTOR)
542             {
543                 for (;i<num_characters; i++ )
544                 {
545                     if (reinterpret_cast<t_name_vector*>
546                             (r)->vector[vmark (i)])
547                     {//ascend
548                         r=reinterpret_cast<t_name_vector*>
549                                 (r)->vector[vmark (i)];
550                         if (r->type==T_RECORD)// found record
551                             return r;
552                         else// found vector
553                         {// browse it from it's begining
554                             i=-1;
555                             continue;
556                         }
557                     }
558                 }
559             }
560             if (r->parent not_eq 0)
561             {
562                 // increase i, to point to vector position AFTER current
                        record
563                 for (i=0;r->parent->vector[vmark (i++)]!=r; )
564                     ;
565                 // descend
566                 r=reinterpret_cast<t_name_record*>(r->parent);
567             }
568             else
569             {
570                 return NULL;
571             }
572         }// break loop if r->parent=0, as we have found tree.first
573         while (true);
574         THROW (LogicError);
575     }
576     //tree_find_next
577
578     template<typename T>
579         void Dictionary<T>::tree_erase_vector (t_name_vector * v)
580     {
581
582         for (size_t i=0;i<num_characters; i++ )
583         {
584             if (v->vector[i] not_eq NULL)
585             {
586                 if (v->vector[i]->type==T_VECTOR)
```

```
587                     tree_erase_vector (
588                             reinterpret_cast<t_name_vector *>
589                             (v->vector[i]));
590                 else
591                     tree_erase_record (v->vector[i]);
592             }
593         }
594         //free(v->vector); // vector is static field
595         free (v);
596     }
597     // tree_erase_vector
598
599 #undef vmark
600 #undef record
601 }
602 #endif  /* __DICTIONARY_H__ */
```

3.5 lib/screen/include/genericscreen.h++

```
 1
 2 #ifndef __GENERIC_SCREEN_H__
 3 #define __GENERIC_SCREEN_H__
 4 #include "screenbase.h++"
 5 #include "utf8.h++"
 6 #include "bufferedinput.h++"
 7 #include "screenbuffer.h++"
 8 #include "subscreen.h++"
 9 #include <queue>
10 #include <vector>
11 namespace Scr
12 {
13
14     //! Most basic implementation of whole Scr::Screen
15     /*!
16       This class provides generic implementation of large part of Scr::
              Screen
17       interface, including basic output subroutines, but some of them
18       lacks important platform-specific features
19     */
20     class GenericScreen: public virtual ScreenBase
21     {
22     private:
23         /*!
24          * Function used to compute width of text as well as width of each
25          * character. The function is designed to be called from within
                 all
26          * types of AddText
27          *
28          * \return width of string (correct value <= maxwidth)
29          *
30          * \param text is text, whose element widths need to be computed
31          * \param widths is C-type array of character widths, that need to
                  be
32          *   computed
33          * \param maxwidth is max width of whole text (if width of whole
                 text
34          *   exceeds allowed width, stop computation and throw exception)
```

```
35          *
36          * \exception Scr::Screen::RangeError exception is thrown when
                text is
37          *   too wide.
38          *
39          * \exception Scr::Screen::IllegalCharacter exception is thrown
                when UNICODE
40          *   encoding is incorrect (validation occurs only for _char_type=
                char)
41          */
42         template<typename _char_type>
43         Uint PrecomputeTextCharsWidth(_char_type * text, std::vector<char
               >& widths,
44                 Uint maxwidth)
45             throw(RangeError, IllegalCharacter);
46                                     //please note, that this template is
47                                     //implemented in genericscreen.c++
48                                     //and only there may be called with
49                                     //any _char_type
50
51     protected:
52         /*!
53           buffer used to implement all textual operations. All Add*
54           functions operate on it directly.
55         */
56         ScreenBuffer controlBuffer;
57         /*!
58           current properties (set w/ SetBg/FgColor/Style)
59         */
60         DisplayStyle properties;
61
62         /*!
63           cursorPosition
64          */
65         Position cursorPosition;
66
67         static const Uint cursorForced  = 1;
68         static const Uint cursorVisible = 2;
69         mutable Uint cursorFlags;
70
71         BufferedInput input;
72
73         /*!
74           Output file stream for writing
75         */
76         std::ostream & output;
77
78         /*!
79           get key esc-code from std input stream. decode it into form
80           from keyboard.h++
81         */
82         virtual Key DecodeKeyPressed()
83         throw(Connection::UnsupportedKey,Screen::InvalidUTF8);
84
85         virtual Key DecodeBasicKeyPressed()throw(Screen::InvalidUTF8);
86
87     public:
88
89         /*!
90          * \param _input
91          * \param _output
92          *
```

```
93          * GenericCcreen operates on C++ standard iostream.
94          */
95        GenericScreen(std::istream & _input,std::ostream & _output)throw()
              ;
96        /*!
97          empty controlBuffer
98        */
99        virtual void Clear()throw();
100
101       /*!
102         \param col new background colour to be set
103         \return  nothing upon successful execution
104
105
106         Function operates on properties member object.
107
108         Refer to manual for base class for action description.
109       */
110       virtual void SetBgColor(Bg::Color col)throw();
111
112       /*!
113         \param col new foreground colour to be set
114         \return  nothing upon successful execution
115
116         Function operates on properties member object.
117
118         Refer to manual for base class for action description.
119       */
120       virtual void SetFgColor(Fg::Color col)throw();
121
122       /*!
123         \param s new foreground text style to be set
124         \return  nothing upon successful execution
125
126         Function operates on properties member object.
127
128         Refer to manual for base class for action description.
129       */
130       virtual void SetFgStyle(Fg::Style s)throw();
131
132       /*!
133         \param y
134         \param x  new coordinates of active point (please
135         remember the order of theese attributes)
136
137         Operates on coordinate values inherited from ScreenBase
138       */
139       virtual void GotoYX(Uint y, Uint x)
140           throw(GotoOutOfRange);
141
142       /*!
143         \param c
144
145         Operates on controlBuffer and  coordinate values inherited from
              ScreenBase
146       */
147       virtual void AddCharacter(char c)throw(PrintOutOfRange);
148
149       /*!
150         \param c
151
```

```
152           Operates on controlBuffer and  coordinate values inherited from
                  ScreenBase
153         */
154         virtual void AddCharacter(wchar_t c)
155             throw(PrintOutOfRange, IllegalCharacter);
156
157         /*!
158           \param p position
159
160           visible after refresh
161          */
162         virtual void ForceCursorPosition(Position p )throw(RangeError);
163
164         /*!
165          * \param text text to be printed (as C string)
166          * \copydoc Screen::AddText(const char *)
167          *
168          * \note Operates on controlBuffer and  coordinate values
                 inherited
169          * from ScreenBase
170          */
171         virtual void AddText(const char * text)throw(PrintOutOfRange,
172                                                      IllegalCharacter);
173
174         /*!
175          * \param text what to be printed (as C++ string)
176          * \copydoc Screen::AddText(const std::string &)
177          * \note Operates on controlBuffer and  coordinate values
                 inherited
178          * from ScreenBase
179          */
180         virtual void AddText(const std::string & text)
181             throw(PrintOutOfRange,
182                   IllegalCharacter);
183
184         /*!
185          * \param text UTF-8 encoded character string
186          * \param cols length of string
187          * \param widths widths of subsequent characters
188          *
189          * Function prints specified text assuming, that its width is
                 EXACTLY
190          * specified by cols parameter
191          *
192          * \exception PrintOutOfRange is thrown if
193          * initial position of active point is invalid, or if text is
194          * too long (as function does not support line breaks).
195          *
196          * \exception IllegalCharacter will be
197          * thrown if text supplied is not a valid UTF-8 string (even
198          * "overlong sequences" will be considered illegal (according
199          * to an apropriate RFC
200          *
201          * \note function is NOT a part of Scr::Screen interface, and is
                 not
202          * accessible outside of screen module
203          *
204          * \sa Screen::AddText(const char * text) for extensive
                 description
205          */
206         virtual void AddText(const char * text, Uint cols,
207                 const std::vector<char> & widths)
```

```
208             throw(PrintOutOfRange, IllegalCharacter);
209
210         /*!
211          * \param text
212          *
213          * Operates on controlBuffer and  coordinate values inherited from
                ScreenBase
214         */
215         virtual void AddText(const std::wstring & text)
216             throw(PrintOutOfRange, IllegalCharacter);
217
218         /*!
219          * \param text
220          *
221          * Operates on controlBuffer and  coordinate values inherited from
                ScreenBase
222          */
223         virtual void AddText(const wchar_t * text)
224             throw(PrintOutOfRange, IllegalCharacter);
225
226         /*!
227          * \copydoc Screen::AddTextCols(const wchar_t *, Uint)
228          * \note Operates on controlBuffer and  coordinate values
                inherited
229          * from ScreenBase.
230          */
231         virtual Uint AddTextCols(const wchar_t * text, Uint limitcols)
232             throw(PrintOutOfRange, IllegalCharacter);
233         // Doxygen complains about 'not defined parameters'. they are
                defined,
234         // but in Scr::Screen (and \copydoc copies 'em)
235
236         /*!
237          * \copydoc Screen::AddTextCols(const wchar_t *, Uint)
238          * Operates on controlBuffer and  coordinate values inherited from
                ScreenBase
239          */
240         virtual Uint AddTextCols(const std::wstring & text, Uint limitcols
                )
241             throw(PrintOutOfRange, IllegalCharacter);
242
243         /*!
244          * Function adds "text in subscreen", that is text, which was to
                be
245          * be inserted in subscreen. This function is called by apropriate
246          * Scr::Subscreen::AddText .
247          *
248          * \param text UTF-8 encoded text to be printed
249          * \param width maximum number of columns to be printed
250          *
251          * \exception Scr::Screen::IllegalCharacter may be thrown if any
252          *   character of text is incorrectly encoded
253          *
254          * \exception Scr::Screen::PrintOutOfRange is thrown when text
                runs
255          *   out of root screen range or when it's width (as number of
256          *   columns, not characters) exceeds widthlimit.
257          */
258         void AddSubscreenText(const char * text, Uint widthlimit)
259             throw(PrintOutOfRange, IllegalCharacter);
260
261         /*!
```

```
262         * Purpose of this function is as above, but one of parameters
263         * slightly differs.
264         *
265         * \param text UNICODE text
266         * \param width maximum number of columns to be printed
267         *
268         * \exception Scr::Screen::PrintOutOfRange is thrown when text
                runs
269         *   out of root screen range or when it's width (as number of
270         *   columns, not characters) exceeds widthlimit.        *
271         */
272        void AddSubscreenText(const wchar_t * text, Uint widthlimit)
273            throw(PrintOutOfRange, IllegalCharacter);
274
275        // for following functions please refer to docmentation-comment
276        // in include/rexio/screen.h++
277        virtual void HorizontalLine(char c, Uint n)
278            throw(PrintOutOfRange, IllegalCharacter);
279
280        virtual void HorizontalLine(wchar_t c, Uint n)
281            throw(PrintOutOfRange, IllegalCharacter);
282
283        virtual void VerticalLine(char c, Uint n)
284            throw(PrintOutOfRange, IllegalCharacter);
285
286        virtual void VerticalLine(wchar_t c, Uint n)
287            throw(PrintOutOfRange, IllegalCharacter);
288
289        virtual void Rectangle(char c, const Size & s)
290            throw(PrintOutOfRange, IllegalCharacter);
291
292        virtual void Rectangle(wchar_t c, const Size & s)
293            throw(PrintOutOfRange, IllegalCharacter);
294
295        /*!
296          make cursor invisible
297         */
298        virtual void HideCursor()throw(CursorVisibilityNotSupported);
299
300        /*!
301          make it visible again
302         */
303        virtual void ShowCursor()throw(CursorVisibilityNotSupported);
304
305        /*!
306          Most basic implementation suitable really only for dumb
307          terminals or line printers: prints each line of buffer to
308          stdout. Created only for debugging reasons.
309        */
310        void Refresh()
311        // just a dumb proc to produce
312        // basic debug printout
313            throw(ConnectionError);
314
315        virtual Screen *
316        CreateSubScreen(Uint _y_offset,
317                        Uint _x_offset, Uint _h,
318                        Uint _w)throw(SubscreenOutOfRange);
319
320        /*!
321          \return always throw exceptn
322        */
```

```
323         virtual const char * GetType() const throw(TerminalTypeUnknown);
324 
325         /*!
326           \return height of controlBuffer
327         */
328         virtual Uint GetHeight() const throw();
329 
330         /*!
331           \return width of controlBuffer
332         */
333         virtual Uint GetWidth() const throw();
334 
335         virtual bool GetCursorVisibility() const throw();
336 
337         /*!
338           Cleans screen up: restore default colours and clear (it is
339           good to use this function while finishing application etc.)
340         */
341         virtual void CleanUp()  throw(ConnectionError);
342 
343         virtual void Resize(Uint rows, Uint cols)throw();
344 
345         ~GenericScreen()throw();
346 
347     };
348 
349 }
350 
351 #endif // __GENERICSCREEN_H__
```

3.6 lib/screen/include/localscreen.h++

```
 1 
 2 #ifndef __LOCALSCREEN_H__
 3 #define __LOCALSCREEN_H__
 4 #include <termios.h>
 5 #include "genericscreen.h++"
 6 #include "screenconnection.h++"
 7 namespace Scr
 8 {
 9     //! connection on localhost, using cin/cout
10     class LocalScreen: public virtual GenericScreen,
11                public __ScreenConnection
12     {
13     private:
14     /*!
15       Store initial terminal settings to restore them after
16       finishing connection (especially settings connected with
17       local echo.
18      */
19     struct termios term;
20     public:
21 
22         /*!
23           \param infd file descriptor
24           \return true if size changed
25 
```

```
26          Function checks if size set for object equals size of
27          appropriate screen. If it differs, Resize() is called to
28          match changes
29        */
30        void TestForResize();
31
32        /*!
33          \param _connection reference to object representing
34          connection itself
35          \param _input reference to input stream
36          \param _output reference to output stream
37         */
38        LocalScreen(Connection & _connection,std::istream & _input,std::
              ostream & _output)throw();
39
40        /*!
41          \return getenv("TERM");
42         */
43        virtual const char * GetType() const throw();
44
45
46        /*!
47          basic main loop of application using local screen
48        */
49        int ProcessConnection();
50        ~LocalScreen()throw();
51    };
52 }
53
54 #endif // __LOCALSCREEN_H__
```

3.7 lib/screen/include/remotescreen.h++

```
 1
 2 #ifndef __REMOTE_SCREEN_H__
 3 #define __REMOTE_SCREEN_H__
 4 #include "screenconnection.h++"
 5 #include "keyboard.h++"
 6 //not include termios.h as it declares macro ECHO conflicting with
 7 //ECHO in TELNET namespace
 8
 9 namespace Scr
10 {
11     //! TELNET connection
12     class RemoteScreen: public virtual GenericScreen,
13                         public __ScreenConnection
14     {
15     protected:
16         std::string clientname;
17
18         /*!
19           general subnegotiation switch.
20         */
21         void AnswerCommand();
22         /*!
23           Read window size and possibly call OnResize; Handle
24           subnegotiation end (SE) correctly.
```

```
25          ASSUME, that IAC SB NAWS was just recv, so process w h IAC
26          SE (check for correctnes after each).
27          \latexonly \index{RFC, reference to!1073}\endlatexonly
28        */
29        void SubnegotiateWindowSize();
30        /*!
31          read information on terminal type.
32          \latexonly \index{RFC, reference to!1091}\endlatexonly
33        */
34        void SubnegotiateTerminalType();
35        /*!
36          Process characters according to telnet protocol. Handle
37          variants of Enter key.
38
39          \latexonly \index{RFC, reference to!854}\endlatexonly
40        */
41        virtual Key DecodeKeyPressedHandleTelnet();
42
43        /*!
44          When resize request is pending, store requested dimensions
45          here.
46         */
47        Size requestedSize;
48        /*!
49          Client has requested resize. Let him wait until counter == 0.
50         */
51        bool resizeRequestPending;
52
53        /*!
54          1-2-3-...-254-255-0
55         */
56        char counter;
57
58        static const Uint windowSize = 1;
59        static const Uint terminalType = 2;
60        Uint telnetSettings;
61    public:
62        RemoteScreen(Connection & _connection,
63                     std::istream & _input,
64                     std::ostream & _output)throw();
65
66        /*!
67          \return returns information retrieved by
68          SubnegotiateTerminalType() if telnet client supports
69          TELNET::TTYPE extension (RFC 1091). If client does not
70          support this feature, dumb terminal type will be assumed and
71          NULL will be returned. "unknown" special value will be
72          returned
73          \latexonly \index{RFC, reference to!1091}\endlatexonly
74         */
75        virtual const char * GetType() const throw(TerminalTypeUnknown);
76
77        int ProcessConnection();
78        ~RemoteScreen()throw();
79        friend class TelnetCommandProcessor;
80    };
81 }
82 #endif
```

3.8 lib/screen/include/screenbase.h++

```
1 #ifndef __SCREENBASE_H__
2 #define __SCREENBASE_H__
3
4 #include "screen.h++"
5
6 namespace Scr
7 {
8     //! \brief Implements features common to subscreen and
9     //! generic screen
10    class ScreenBase: public virtual Screen
11    {
12    protected:
13        /*!
14          vertical and horizontal offset from the left edge of the screen
15        */
16        Position  aPoint;
17    public:
18        ScreenBase();
19
20        /*!
21          \return horizontal offset from the left edge of the screen
22        */
23        Uint GetX()const throw();
24        /*!
25          \return vertical offset from top of the screen
26        */
27        Uint GetY()const throw();
28        virtual void AddText(const std::string & text, Uint cols)
29            throw(PrintOutOfRange,
30                  IllegalCharacter);
31    }; // ScreenBase
32 } // Scr
33
34 #endif // __SCREENBASE_H__
```

3.9 lib/screen/include/screenbuffer.h++

```
1 #ifndef __SCREENBUFFER_H__
2 #define __SCREENBUFFER_H__
3
4 #include <vector>
5 #include "screen.h++"
6
7 namespace Scr
8 {
9
10    class ScreenBuffer;
11    //! \brief character to be displayed with all
12    //! it's properties
13    class ScreenCharacter
14    {
15    public:
16        /*!
17          \param _c character UNICODE code
18          \param _style colour etc.
```

```
19          */
20          ScreenCharacter(Uint _c, const DisplayStyle & _style);
21          /*!
22            \param other right-hand operand
23
24            Assignment operator copies  character and all it's properties
25          */
26          ScreenCharacter & operator=(const ScreenCharacter & other);
27          /*!
28            \param other right-hand operand
29
30            Comparison operator returns true if colour and character match
31          */
32          bool operator==(const ScreenCharacter & other);
33          /*!
34            \param other right-hand operand
35          */
36          bool operator!=(const ScreenCharacter & other);
37          DisplayStyle style;
38          Uint c;
39      };
40
41      //! \brief single row of ScreenBuffer object (which may
42      //! contain more rows)
43      /*!
44        Class implements single row of characters, so it encapsulates std::
              vector.
45      */
46      class ScreenRow
47      {
48      protected:
49          std::vector<ScreenCharacter> characters;
50          /*!
51            \param width number of characters
52            \param character initial content
53          */
54          ScreenRow(Uint width,
55                    const ScreenCharacter & character=
56                    ScreenCharacter(' ',
57                                    DisplayStyle(
58                                        Fg::White, Fg::Dark,Bg::Black)));
59
60          /*!
61            \param newWidth new width of specific row.
62            \param character if new width is greater, than current,
63            additional fields will be filled with this specific character
64            \note declared as protected function to prevent changing
65            width outside of ScreenBuffer, and therefore to assure, that
66            buffer will be rectangular (equal width for each row).
67          */
68          void Resize(Uint newWidth, const ScreenCharacter & character);
69
70      public:
71
72          /*!
73            \param other right-hand operand
74
75            copy content of one buffer to second one. If size differs,
76            target is resized to match source.
77          */
78          ScreenRow & operator=(const ScreenRow & other);
79          /*!
```

```
80          \param i index
81
82          Array element access operator returns reference to the
83          specific character.
84        */
85        ScreenCharacter & operator[] (Uint i)
86            {
87                return characters[i];
88            }
89        /*!
90          \param other right-hand operand
91
92          Comparison for equal compares each character, and returns
93          true if no difference found
94        */
95        bool operator==(const ScreenRow & other);
96        /*!
97          \param other right-hand operand
98
99          Comparison for equal compares each character, and returns
100         true if any difference found
101       */
102       bool operator!=(const ScreenRow & other);
103       Uint GetWidth()const; // get number of columns in row
104       friend class ScreenBuffer;
105   };
106
107   //! \brief buffer of characters, supporting colours
108   //! and unicode.
109   /*!
110     Class represents character buffer.
111   */
112   class ScreenBuffer
113   {
114   protected:
115       typedef std::vector<ScreenRow> RowVector;
116   private:
117       RowVector rows;
118   public:
119       /*!
120         \param  _rows initial height of screen buffer
121         \param columns initial width of screen buffer
122         \param character initial fill of screen buffer (by default
123         plain black background (filled with space))
124
125         \note buffer size may be changed runtime.
126       */
127       ScreenBuffer(Uint _rows, Uint columns,
128                    const ScreenCharacter & character=
129                    ScreenCharacter(' ',
130                                    DisplayStyle(
131                                        Fg::White, Fg::Dark,Bg::Black)));
132       /*!
133         \param _i row number (0..height-1)
134         \return reference to specific row
135         \note no range checking, and no exception-connected
136         warranties for this function.
137       */
138       ScreenRow & operator[] (Uint _i)
139           {
140               return rows[_i];
141           }
```

```
142
143         /*!
144           \param other right-hand operand
145
146           Assign other screen to this one. Function copies whole
147           contents, so complexity is O(width*height).
148         */
149         ScreenBuffer & operator=(const ScreenBuffer & other);
150
151         /*!
152           \param other right-hand operand
153
154           \return true if size of each buffer is equal, each character
155           equals  its counterpart on second buffer, both in terms of
156           unicode value and colour.
157         */
158         bool operator==(const ScreenBuffer & other);
159
160         /*!
161           \param other right-hand operand
162
163           \return true if any difference occours between two screens.
164         */
165         bool operator!=(const ScreenBuffer & other);
166         /*!
167           \param newHeight new height of screen buffer
168           \param newWidth new width of screen buffer
169           \param character character, to fill new rows or colums (if
170           their number grows) with.
171         */
172         void Resize(Uint newHeight,
173                     Uint newWidth,
174                     const ScreenCharacter & character=
175                     ScreenCharacter(' ',
176                                     DisplayStyle(
177                                         Fg::White, Fg::Dark,Bg::Black)));
178
179         /*!
180           \return current height of buffer (number of rows)
181         */
182         Uint GetHeight() const; // get number of rows
183         /*!
184           \return current width of buffer (number of characters in each
                row)
185         */
186         Uint GetWidth() const; // get number of columns in row
187
188         /*!
189           \param character character
190
191           Function fills whole buffer with specific character.
192         */
193         void Fill(const ScreenCharacter & character);//assign new
194                                                      //content
195
196     };
197 }
198
199 #endif // __SCREENBUFFER_H__
```

3.10 lib/screen/include/screenconnection.h++

```
1 #ifndef __SCREEN_CONNECTION_H__
2 #define __SCREEN_CONNECTION_H__
3 #include "keyboard.h++"
4 #include"fileno_hack.h++"
5 #include"throw.h++"
6
7 namespace Scr
8 {
9
10     //! \brief internal class which is base for all connection-specific
11     //! implementations of screen (multiple-inheritance case)
12     /*!
13       It represents internal interface between Scr::Connection and
14       Scr::Screen classes.
15     */
16     class __ScreenConnection
17     {
18     protected:
19         /*!
20           ProcessConnection will return this value upon successful finish
21         */
22         int exitcode;
23         /*!
24           is application running? does it have to stop?
25           (ExitConnection() is called by Connection::Exit(int), sets
26           exit code and breaks main loop performed in ProcessConnection)
27         */
28         Connection & connection;
29
30         /*!
31           break main loop if set to false
32          */
33         bool active;
34     public:
35         /*!
36
37           \param _input input stream (used to capture some of events,
38           in particular keyboard events).
39           \param _connection newely estabilished connection to serve
40         */
41         __ScreenConnection(Connection & _connection,std::istream & _input)
               throw();
42         /*!
43
44           \return value of exitcode, as it was in the moment of
45           connection termination if successful.
46
47           Initialize, conduct and end connection in way apropriate to
48           connection type, operating system etc. Inform
49           Scr::Connection object supplied about all captured events
50
51           \note as function (for design reasons) lacks exception-set
52           specification, it may throw any exceptions, but it is
53           recommended, that only exceptions typical to
54           Scr::Connection::Start() will be thrown.
55
56         */
57         virtual int  ProcessConnection() =0;
58
59         /*!
```

```
60          \param _code exit code return from ProcessConnection after
61          successfully finished connection
62
63          Force stopping connection as soon as possible
64          \note as function (for design reasons) lacks exception-set
65          specification, it may throw any exceptions, but it is
66          recommended, that only exceptions typical to
67          Scr::Connection::Exit() will be thrown.
68        */
69        virtual void ExitConnection(int _code);
70        virtual ~__ScreenConnection()throw();
71    };
72 }
73 #endif
```

3.11 lib/screen/include/subscreen.h++

```
1 #ifndef __SUB_SCREEN_H__
2 #define __SUB_SCREEN_H__
3 #include"screenbase.h++"
4 #include"genericscreen.h++"
5
6 namespace Scr
7 {
8     class GenericScreen;
9
10    /*!
11      Subscreen may be considered a specified region of screen limited
12      to one rectangle. Subscreen does not provide it's own buffer, so
13      it can be used as range for specific procedure rather than a
14      layer. It allows all actions, but limited to it's width and
15      height. It is useful for implementing procedures drawing
16      specific features, i.e. widgets in UI toolkit.
17
18      Strict range limitation is achieved by disabling of Scr::SubScreen::
            Resize
19      member function
20    */
21    class SubScreen:public ScreenBase
22    {
23    protected:
24        /*!
25          reference to parent screen
26        */
27        GenericScreen & parent;
28        /*!
29          vertical distance from top of containing
30          (parent) screen to top of this
31           and horizontal  distance from its left edge.
32        */
33        Position offset;
34
35        /*!
36          Width and height of screen
37         */
38        Size s;
39
```

```
40        /*!
41         * Call GotoYX for parent. Rethrow possible exception as
42         * Printing exception.
43         */
44        inline void ParentGotoYXForPrinting()throw(PrintOutOfRange);
45    public:
46        /*!
47          \param _parent reference to parent screen
48          \param _y_offset vertical distance from top of containing
49          (parent) screen to top of this
50          \param _x_offset horizontal  distance from left edge of
                containing
51          (parent) screen to left edge of this
52          \param _h height
53          \param _w width
54        */
55        SubScreen(GenericScreen & _parent,
56                  Uint _y_offset,
57                  Uint _x_offset,
58                  Uint _h,
59                  Uint _w)throw();
60
61        /*!
62          Fills rectangle defined by this subscreen with current
63          background color, directly on containing buffer (so it may
64          be later hidden by containing buffer)
65        */
66        virtual void Clear()throw();
67
68        /*!
69          \param col color
70
71          Subscreen does not have it's own DisplayProperties, so it
72          calls SetBgColor for parent screen
73        */
74        virtual void SetBgColor(Bg::Color col)throw();
75        /*!
76          \param col color
77
78          Subscreen does not have it's own DisplayProperties, so it
79          calls SetFgColor for parent screen
80        */
81        virtual void SetFgColor(Fg::Color col)throw();
82        /*!
83          \param s style
84
85          Subscreen does not have it's own DisplayProperties, so it
86          calls SetFgStyle for parent screen
87        */
88        virtual void SetFgStyle(Fg::Style s)throw();
89
90        /*!
91          \param x
92          \param y
93
94          this does not access directly to parent window, as SubScreen
95          has it's own YX coordinates
96        */
97        virtual void GotoYX(Uint y, Uint x)
98            throw(GotoOutOfRange);
99
100       /*!
```

```
101           \param text
102
103           Print text directly on parent buffer
104           \note it means, that first appropriate GotoYX must be called
105           for parent, so it modifies not only contents of buffer, but
106           also coordinates of its active point.
107         */
108         virtual void AddText(const char * text)throw(PrintOutOfRange,
109                                                      IllegalCharacter);
110
111         /*!
112           \param text
113
114           Same as above.
115          */
116         virtual void AddText(const std::string & text)
117             throw(PrintOutOfRange,
118                   IllegalCharacter);
119
120         /*!
121           \param text
122
123           Same as above
124         */
125         virtual void AddText(const wchar_t * text)
126             throw(PrintOutOfRange, IllegalCharacter);
127
128         /*!
129           \param text
130
131           Same as above, but UNICODE
132         */
133         virtual void AddText(const std::wstring & text)
134             throw(PrintOutOfRange, IllegalCharacter);
135
136
137         virtual Uint AddTextCols(const wchar_t * text, Uint limitcols)
138             throw(PrintOutOfRange,
139                   IllegalCharacter);
140
141         virtual Uint AddTextCols(const std::wstring & text, Uint limitcols
                )
142             throw(PrintOutOfRange, IllegalCharacter);
143
144         // for following functions please refer to docmentation-comment
145         // in include/rexio/screen.h++
146         virtual void HorizontalLine(char c, Uint n)
147             throw(PrintOutOfRange, IllegalCharacter);
148
149         virtual void HorizontalLine(wchar_t c, Uint n)
150             throw(PrintOutOfRange, IllegalCharacter);
151
152         virtual void VerticalLine(char c, Uint n)
153             throw(PrintOutOfRange, IllegalCharacter);
154
155         virtual void VerticalLine(wchar_t c, Uint n)
156             throw(PrintOutOfRange, IllegalCharacter);
157
158         virtual void Rectangle(char c, const Size & s)
159             throw(PrintOutOfRange, IllegalCharacter);
160
161         virtual void Rectangle(wchar_t c, const Size & s)
```

```
162             throw(PrintOutOfRange, IllegalCharacter);
163
164         /*!
165           \param c
166
167           Print character directly on parent buffer
168           \note as for AddText, it modifies not only contents of buffer,
                  but
169           also coordinates of its active point.
170         */
171         virtual void AddCharacter(char c)throw(PrintOutOfRange);
172
173         /*!
174           \param c
175
176           Print UNICODE character directly on parent buffer
177           \note as for AddText, it modifies not only contents of buffer,
                  but
178           also coordinates of its active point.
179         */
180         virtual void AddCharacter(wchar_t c)throw(PrintOutOfRange,
181                                                   IllegalCharacter);
182
183         /*!
184           \param p position
185
186           mapped to parent
187          */
188         virtual void ForceCursorPosition(Position p )throw(RangeError);
189
190         /*!
191           make cursor invisible
192          */
193         virtual void HideCursor()throw(CursorVisibilityNotSupported);
194
195         /*!
196           make it visible again
197          */
198         virtual void ShowCursor()throw(CursorVisibilityNotSupported);
199
200         /*!
201           force refresh of parent buffer
202         */
203         virtual void Refresh()throw(ConnectionError);
204
205         /*!
206           \param rows
207           \param cols
208           \exception Scr::Screen::SubscreenResize is
209           thrown always, as subscreen can not be resized
210         */
211         virtual void Resize(Uint rows, Uint cols)
212             throw(SubscreenResize);
213
214         /*!
215           Return type of parent screen type (effectively it is the
216           type of underlying real screen)
217         */
218         virtual const char * GetType() const throw(TerminalTypeUnknown);
219
220         virtual Uint GetHeight() const throw();
221
```

```
222         virtual Uint GetWidth() const throw();
223
224         virtual bool GetCursorVisibility() const throw();
225
226         virtual Screen *
227         CreateSubScreen(Uint _y_offset, Uint _x_offset, Uint _h,
228                         Uint _w)throw(SubscreenOutOfRange);
229
230         ~SubScreen()throw();
231     };
232 }
233
234 /*!
235   This particular macro is useful while implementing CreateSubScreen
236   function. it does full range checking according to prototype
237   declared in screen.h++
238  */
239 #define SubScreenRangeCheck()                                          \
240     /*throw exception on inproper position of subscreen */             \
241     if (_x_offset>=GetWidth())                                         \
242         THROW(SubscreenOutOfRange);                                    \
243     if (_y_offset>=GetHeight())                                        \
244         THROW(SubscreenOutOfRange); /* as theese are */                \
245     /* unsigned integers, an  */                                       \
246     /* exception is thrown also when _x_offset<0 or _y_offset<0  */ \
247                                                                        \
248     /* throw exception on inproper size of subscreen             */ \
249     if (_x_offset+_w>GetWidth())                                       \
250         THROW(SubscreenOutOfRange);                                    \
251     if (_y_offset+_h>GetHeight())                                      \
252         THROW(SubscreenOutOfRange);
253
254 #endif // __SUBSCREEN_H__
```

3.12 lib/screen/include/telnet.h++

```
 1 #ifndef __TELNET_H__
 2 #define __TELNET_H__
 3
 4 //! Telnet control codes
 5 /*!
 6   Whole set of constants useful for telnet negotiations as server or
 7   client. All of them are declared in apropriate RFC's.
 8 */
 9 namespace TELNET
10 {
11
12
13     //!\brief Binary mode
14     //!
15     const unsigned char   BINARY = 0x00;
16     //!\brief Local/remote echo mode
17     //!
18     //!IAC WILL ECHO sent by server disables local echo
19     //!\sa RFC 857
20     //!\latexonly \index{RFC, reference to!857}\endlatexonly
21     const unsigned char     ECHO = 0x01;
```

```
22    //!\brief Suppress go ahead
23    //!
24    //!\sa RFC 858
25    //!\latexonly \index{RFC, reference to!858}\endlatexonly
26    const unsigned char      SGA = 0x03;
27    //!\brief Terminal Type negotiation
28    //!
29    //!Detect terminal type and - possibly - detect it's additional
30    //!emulation modes and switch between them. Documentation for this
31    //!feature described in appropriate RFC.
32    //!
33    //!\sa RFC 1091
34    //!\latexonly \index{RFC, reference to!1091}\endlatexonly
35    const unsigned char    TTYPE = 0x18;
36
37    //!\brief request terminal type information
38    //!
39    //!Command code used by server while requesting TTYPE
40    //!\sa RFC 1091
41    const unsigned char     SEND = 0x01;
42
43    //!\brief inform about terminal type
44    //!
45    //!Command code used by client while informing about terminal type
46    //!during TTYPE subnegotiation
47    //!\sa RFC 1091
48    const unsigned char       IS = 0x00;
49
50    //!\brief Negotiate about Window Size
51    //!
52    //!\sa RFC 1073
53    //!\latexonly \index{RFC, reference to!1073}\endlatexonly
54    const unsigned char     NAWS = 0x1F;
55
56
57    //!\brief Line mode negotiation
58    //!
59    //!For description of this feature refer to appropriate RFC
60    //!\sa RFC 1184
61    //!\latexonly \index{RFC, reference to!1184}\endlatexonly
62    const unsigned char LINEMODE = 0x24;
63
64    //!\brief Subnegotiation end
65    //!
66    //!Special code inserted at the end of subnegotiation block
67    const unsigned char       SE = 0xF0;
68    //!\brief No operation
69    //!
70    //!Do not do anything
71    const unsigned char      NOP = 0xF1;
72    //!\brief Data mark
73    //!
74    const unsigned char       DM = 0xF2;
75    //! Break
76    const unsigned char      BRK = 0xF3;
77    //! Interrupt Process
78    const unsigned char       IP = 0xF4;
79    //! Abort Output
80    const unsigned char       AO = 0xF5;
81    //! Are you there?
82    const unsigned char      AYT = 0xF6;
83    //! Erase character
```

```
 84     const unsigned char       EC = 0xF7;
 85     //! Erase line
 86     const unsigned char       EL = 0xF8;
 87     //! Go ahead (allow other end to transmit)
 88     const unsigned char       GA = 0xF9;
 89     //! Subnegotiation begin
 90     const unsigned char       SB = 0xFA;
 91     //! Will (meaning depends on feature, we negotiate)
 92     const unsigned char     WILL = 0xFB;
 93     //! Won't (meaning depends on feature, we negotiate)
 94     const unsigned char     WONT = 0xFC;
 95     //! Do (meaning depends on feature, we negotiate)
 96     const unsigned char       DO = 0xFD;
 97     //! Don't (meaning depends on feature, we negotiate)
 98     const unsigned char     DONT = 0xFE;
 99     //!\brief Interpret as command
100     //!
101     //!Special code in the beginning of all control sequences.
102     const unsigned char      IAC = 0xFF;
103 }
104
105 #endif /* __TELNET_H__ */
```

3.13 lib/screen/include/terminal.h++

```
 1 #ifndef __TERMINAL_H__
 2 #define __TERMINAL_H__
 3 #include "screen.h++"
 4 #include "vt100compatible.h++"
 5
 6 namespace Scr
 7 {
 8
 9     /*!
10       \brief  base class containing data fields typical to any
11       terminal output type
12      */
13     class Terminal
14     {
15     protected:
16
17
18         /*!
19           Coordinates of cursor onscreen
20         */
21         struct
22         {
23             Uint x;//!< column
24             Uint y;//!<row
25         } termCoords;
26         /*!
27           Copy of expected screen contents - used to optimise
28           Refresh() for transfer
29         */
30         ScreenBuffer copyBuffer;
31     public:
32         Terminal()throw();
```

```
33     };
34 }
35
36 #endif
```

3.14 lib/screen/include/terminfoenabled.h++

```
 1 #ifndef __TERMINFO_ENABLED_H__
 2 #define __TERMINFO_ENABLED_H__
 3 #include "screen.h++"
 4 #include "terminal.h++"
 5 #include "terminfo.h++"
 6
 7 namespace Scr
 8 {
 9     //! class representing terminal controlled according to terminfo
         database
10     /*!
11       This class provides full implementation of Scr::Screen abstract
12       interface in terms of capabilities of any terminal described in
13       terminfo database.
14
15       \latexonly
16       Algorithm for Refresh()
17
18       \begin{figure}[H]
19       \begin{center}
20       \leavevmode
21       \includegraphics[width=260pt]{../
             Optimised_Refresh_algorithm_general_flowchart}
22       \end{center}
23       \end{figure}
24
25       Aux procedure used there
26       \begin{figure}[H]
27       \begin{center}
28       \leavevmode
29       \includegraphics[width=100pt]{../next_coords_algorithm}
30       \end{center}
31       \end{figure}
32
33       \endlatexonly
34      */
35     class TerminfoEnabledScreen:public virtual GenericScreen, public
         Terminal
36     {
37     private:
38         TI::TerminfoEntry * ti;
39
40
41         DisplayStyle p;
42
43     protected:
44
45         /*!
46             Minimum implementation supportingonly 12 basic functionkeys,
                 arrows
```

```
47              and few special, in several formats of VT100-like terminal
                    emulators.
48           */
49          virtual Key DecodeKeyPressed()
50          throw(Connection::UnsupportedKey,Screen::InvalidUTF8);
51      public:
52          explicit TerminfoEnabledScreen(std::istream & _input,
53                  std::ostream & _output)throw();
54
55          /*!
56            Full support for colour and refreshing algorithm optimized
57            for transfer
58          */
59          virtual void Refresh()throw(ConnectionError);
60
61          /*!
62            \param rows
63            \param cols
64
65            differs from Scr::GenericScreen::Resize only by the fact,
66            that it supports copyBuffer
67          */
68          virtual void Resize(Uint rows, Uint cols)
69              throw();
70          virtual void CleanUp()  throw(ConnectionError);
71          ~TerminfoEnabledScreen()throw();
72      };
73  }
74
75  #endif
```

3.15 lib/screen/include/terminfo.h++

```
 1  #ifndef __TERMINFO_CORE_H__
 2  #define __TERMINFO_CORE_H__
 3  #include"screen.h++"
 4  #include"dictionary.h++"
 5  #include<fstream>
 6  #include<iostream>
 7  #include <boost/thread/mutex.hpp>
 8  namespace Scr
 9  {
10      /*!
11        Terminfo database connectivity facilities
12       */
13      namespace TI
14      {
15
16          class Keymap;
17          __DE(DatabaseException,Exception);
18          __DE(FailedToLoadDatabaseEntry,DatabaseException);
19          __DE(FailedToOpenDatabase,DatabaseException);
20          __DE(DatabaseNotOpen,DatabaseException);
21          __DE(NotSupportedTerminalType,Exception);
22
23          class TerminfoCore;
24          /*!
```

```
25          \brief Terminfo entry for single terminal type
26
27          Terminfo entries will be read from system terminfo database
28          (hashed database or hierarchical directory tree). Only way
29          to obtain this class object is to call apropriate function
30          of TerminfoCore object;
31         */
32        class TerminfoEntry
33        {
34        private:
35            struct
36            {
37
38                //(2) the size, in bytes, of the names section;
39                short namesSize;
40
41                //(3) the number of bytes in the boolean section;
42                // (one boolean value in one byte)
43                short numBooleans;
44
45                //(4) the number of short integers in the numbers section;
46                short numIntegers;
47
48                //(5) the number of offsets (short integers) in the
                      strings section;
49                short numOffsets;
50
51                //(6) the size, in bytes, of the string table.
52                short stringTableSize;
53            }Meta;
54            struct
55            {// 3 C-style dumb vectors
56                char * names;
57                char * booleans;
58                short * numbers;
59                char ** strings;
60            }Data;
61            char * text; // raw content of file after metadata.
62
63            mutable boost::mutex textmod_mtx;
64        protected:
65            /*!
66              \param ifile - resource reference to compiled terminfo file,
                      that will be used
67              to initialize this entry
68
69              Default constructor opens the file and reads all
70              information in it.
71            */
72            explicit TerminfoEntry(std::ifstream & ifile)throw();
73
74            /*!
75              \param i cap. ID (enumerated in capabilities.h++)
76              \return i'th boolean value
77             */
78            bool GetBoolean (int i) const throw();
79
80            /*!
81              \param i cap. ID (enumerated in capabilities.h++)
82              \return positive integer if feature is supported; -1
                    otherwise.
83             */
```

```
84              short GetInteger (int i) const throw();
85
86              /*!
87                \param i cap. ID (enumerated in capabilities.h++)
88                \return c-style string if feature is supported. NULL
89                pointer otherwise.
90               */
91              const char * GetString (int i) const throw();
92          public:
93              __DE(CapabilityNotSupported,Exception);
94              __DE(ParseError,Exception);
95          protected:
96              /*!
97                \param i cap. ID (enumerated in capabilities.h++)
98                \param param parameters
99                (refer to terminfo(5) for parameter descriptions)
100
101               Parse parametrized string
102
103               \note implementation currently does not fully conform
104               specification, however it does what is needed for this
                      library.
105              */
106             std::string ParseString(int i, Uint * param)
107                 const throw(CapabilityNotSupported,ParseError);
108
109             Keymap * keymap;
110         public:
111
112             Keymap & GetKeymap() const;
113
114             /*!
115               \param newPosition new position (0,0 .. height-1,width-1)
116
117               \return control string to set cursor position specific
118               to this terminal type
119
120               Explicitly move cursor to the new position
121              */
122             const std::string GotoYX(const Scr::Position & newPosition)
                    const
123                 throw(CapabilityNotSupported);
124
125             /*!
126               \param newPosition new position of cursor (0,0 .. height-1,
                      width-1)
127               \param oldPosition current position
128
129               \return optimal control string to set cursor position
                      specific
130               to this terminal type
131
132               Recommended way of setting cursor position. This
133               function selects way of setting position, that consumes
134               least possible number of bytes.
135
136               \note dest and then source: the same argument order as
137               for C library functions.
138              */
139             const std::string GotoYX(const Scr::Position & newPosition,
140                                      const Scr::Position & oldPosition)
                                           const
```

```
141             throw(CapabilityNotSupported)  ;
142
143           /*!
144             \param s display style to be set
145             \return control string to set display style for text.
146
147            */
148           const std::string SetDisplayStyle(const Scr::DisplayStyle s)
                  const
149                throw(CapabilityNotSupported)  ;
150
151           /*!
152             \param newStyle display style to be set
153             \param oldStyle current style
154             \return control string to set display style for text.
155
156             if current style is known, it is highly recommended to
157             use this function as it will set minimum required subset
158             of style attributes
159            */
160           const std::string SetDisplayStyle(const Scr::DisplayStyle
                  newStyle,
161                                             const Scr::DisplayStyle oldStyle)
                                                  const
162             throw(CapabilityNotSupported) ;
163
164           /*!
165             Make cursor visible
166            */
167           const std::string ShowCursor() const throw(
                  CapabilityNotSupported);
168
169           /*!
170             Make cursor invisible
171            */
172           const std::string HideCursor() const throw(
                  CapabilityNotSupported);
173
174           /*!
175             Move cursor to the begining-of-the-screen position
176             ( the same effect as GotoYX(Position(0,0)), but possibly
177             faster )
178            */
179           const std::string CursorHome() const throw(
                  CapabilityNotSupported);
180
181
182           ~TerminfoEntry()throw();
183           friend class TerminfoCore;
184           friend class Keymap;
185       };
186
187
188       /*!
189         \brief Terminfo subsystem core: manages entries etc.
190
191         As this class is a singleton class, only one it's instance
192         may exist in the same time. don't bother calling it's
193         constructor manually, as this will result in exiting program
194         at all.
195       */
196       class TerminfoCore
```

```
197         {
198         private:
199             mutable Scr::Dictionary<TerminfoEntry* > entries;
200
201             /*!
202               Default constructor; called by static GetTerminfo
203               \exception Scr::TI::FailedToOpenDatabase
204               is thrown when no database found in supported format.
205              */
206             TerminfoCore() throw();
207
208             /*!
209               Default destructor
210              */
211             ~TerminfoCore() throw();
212
213             /*!
214               Function returns reference to TerminfoEntry object. If
215               it was already retrieved, reference to existing one is
216               returned. Otherwise new is created.
217              *
218              *\param name name of terminal type ($TERM)
219              */
220             const TerminfoEntry & __GetTerminfo(const char * name)
221                 throw(NotSupportedTerminalType);
222
223         public:
224
225             /*!
226               This function forces initialization of terminfo database
                      subsystem
227              */
228             static void Initialize()throw(FailedToOpenDatabase);
229
230             /*!
231               \return true if database was successfully opened
232              */
233             static bool GetDatabaseStatus()throw(DatabaseNotOpen);
234
235             /*!
236               \param name $TERM
237               \return const reference to terminfo entry object
238               \exception
239               Scr::TI::NotSupportedTerminalType is
240               thrown when not supported terminal type is requested
241
242               \exception Scr::TI::FailedToOpenDatabase
243               is thrown when no database found in supported format.
244              */
245             static const TerminfoEntry & GetTerminfo(const char * name)
246                 throw(NotSupportedTerminalType,FailedToOpenDatabase);
247
248             /*!
249               Force destruction of terminfo subsystem. This may cause
250               numerous problem while any objects are still referencing
251               terminfo entries. This function frees all TI resources
252               if any allocated. Otherwise it won't do anything (so
253               that there is no rish of "double free error").
254              */
255             void CleanUp()throw();
256
257             /*!
```

```
258              Function conditionally cleans up terminfo connectivity
                     subsystem.
259             */
260            static  void FreeTerminfoEntry()throw();
261
262        };
263
264    }
265
266 }
267
268 #endif
```

3.16 lib/screen/include/terminfokeymap.h++

```
 1 #ifndef __TERMINFO_KEYMAP_H__
 2 #define __TERMINFO_KEYMAP_H__
 3 #include "dictionary.h++"
 4 #include "terminfo.h++"
 5 #include "keyboard.h++"
 6
 7 namespace Scr
 8 {
 9     namespace TI
10     {
11         class TerminfoEntry;
12
13         /*!
14          * \brief Class responsible for mapping control sequences to
                unique key codes
15          *
16          *
17          */
18         class Keymap
19         {
20         private:
21             typedef Dictionary<Scr::Key> key_dictionary;
22
23             //!real engine of this module is Dictionary Tree.
24             key_dictionary keyboard;
25         protected:
26
27             /*!
28              * \param te Terminfo entry for which keymap will be generated
29              */
30             explicit Keymap(const TerminfoEntry & te)
31                 throw();
32
33             /*!
34              * Do real work of constructor. Way of doing this work may
                    slightly
35              * differ for specific terminal types, so we have to move this
                     action
36              * from the constructor to enable virtualization
37              * \param te Terminfo entry for which keymap will be generated
38              */
39             virtual void InitializeKeymap(const TerminfoEntry & te)
```

```
40                throw();
41
42
43        public:
44
45            typedef key_dictionary::iterator::validity validity;
46
47            /*!
48             * \param code keycode provided by client. i.e. "\x1b[24~"
                  means
49             * function key F12 for DEC VT220 Terminal.
50             *
51             * Test if string supplied matches any key code stored in tree
                  .
52             *
53             */
54            validity TestCode(const char * code)
55                throw();
56
57            /*
58 *           * \return valid key code if matched,
59             * \exception Connection::UnsupportedKey is thrown when no
                  such key
60             * is stored in tree.
61             */
62            Scr::Key GetCode(const char * code)
63                throw(Connection::UnsupportedKey);
64
65            virtual ~Keymap() throw() {;}
66            friend class TerminfoEntry;
67        };
68    }
69 }
70
71 #endif
```

3.17 lib/screen/include/utf8.h++

```
 1 #ifndef __UTF_8_H__
 2 #define __UTF_8_H__
 3 namespace Scr
 4 {
 5     /*!
 6       \param pstr pointer to NULL-terminated c-style string.
 7       \return RAW UNICODE value of utf8 encoded first character of
 8       string supplied.
 9
10       if length of u8 code is greater than 1 byte, pstr is moved
11       by this length-1 forward.
12
13       \exception Scr::Screen::InvalidFirstByte is
14       thrown when **pstr (or pstr[0][0]) does not match 1-byte,
15       2-byte, 3-byte nor 4-byte UTF-8 encoding pattern for first
16       byte.
17
18       \exception Scr::Screen::OverlongUTF8Encoding is
19       thrown when numeric value of result would fit in smaller
```

```
20        number of bytes with correct UTF-8.
21
22        \exception Scr::Screen::InvalidTrailingByte is
23        thrown if second or maybe third or fourth byte does not
24        match template (exactly (c[x]&0xC0)!=0x80)
25
26        \note if compiled without -DDO_VALIDATE_UTF_8_OUTPUT, none
27        of theese exceptions is thrown, and even none of theese
28        error conditions are checked (code assumes, that they never happen)
29        \sa RFC 3629
30        \latexonly \index{RFC, reference to!3629}\endlatexonly
31      */
32      wchar_t DecodeUTF8(const char ** pstr)
33          throw(Screen::InvalidUTF8); // returns RAW UNICODE
34      // value of utf8 encoded character
35      // if length of u8 code is greater than 1 byte, pstr is moved
36      // by this length-1 forward.
37
38      /*!
39        \param c character to encode
40        \param o reference to output stream
41
42        Print c directly to o in UTF8 encoded form
43      */
44      void EncodeUTF8(std::ostream &o,Uint c)throw(); // encode UNICODE
            character
45                                                  // passed and write it to
                                                        output.
46
47      //!\param s UTF-8 string
48      //!
49      //! Compute number of bytes in UTF-8 encoding of the FIRST
50      //! character of  UTF-8 string.
51      //!\note function assumes, that string is correct. No validation
52      //!or range checking is performed
53      Uint CharLengthUTF8(const char * s)
54          throw(Screen::InvalidUTF8);
55
56      //!\param s UTF-8 string
57      //!
58      //! Compute length of null-terminated utf-8 string, that is number
59      //! of UNICODE characters, not number of bytes in UTF-8 encoded
            version.
60      //!\note function assumes, that string is correct. No validation
61      //!or range checking is performed
62      Uint StringLengthUTF8(const char * s)
63          throw(Screen::InvalidUTF8);
64  }
65
66  #endif // __UTF_8_H__
```

3.18 lib/screen/include/vt100compatible.h++

```
1  #ifndef __VT100COMPATIBLE_H__
2  #define __VT100COMPATIBLE_H__
3  #include "screen.h++"
4
```

```
 5 #include "terminal.h++"
 6
 7 namespace Scr
 8 {
 9     //! terminal compatible w/ DEC VT-100
10     /*!
11       This class provides full implementation of Scr::Screen abstract
12       interface in terms of capabilities of DEC VT100 compatible
13       terminals.
14       It will be used as fallback implementation when terminfo
15       database is not availble
16      */
17     class VT100Compatible:public virtual GenericScreen, public Terminal
18     {
19     private:
20
21         void RealGotoYX(const Position & p)throw(ConnectionError);
22
23     protected:
24         /*!
25             Minimum implementation supportingonly 12 basic functionkeys,
                arrows
26             and few special, in several formats of VT100-like terminal
                emulators.
27          */
28         virtual Key DecodeKeyPressed()
29         throw(Connection::UnsupportedKey,Screen::InvalidUTF8);
30     public:
31         explicit VT100Compatible(std::istream & _input,std::ostream &
                _output)throw();
32
33         /*!
34           Full support for colour and refreshing algorithm optimized
35           for transfer
36         */
37         virtual void Refresh()throw(ConnectionError);
38
39         /*!
40           \param rows
41           \param cols
42
43           differs from Scr::GenericScreen::Resize only by the fact,
44           that it supports copyBuffer
45         */
46         virtual void Resize(Uint rows, Uint cols)
47             throw();
48         virtual void CleanUp()  throw(ConnectionError);
49         virtual ~VT100Compatible()throw();
50     };
51 }
52
53 #endif
```

3.19 lib/screen/src/real/vt100codes.h++

```
1 /*! \file vt100codes.h++
2   \brief VT100 terminal control macros.
```

```
 3  Contains macro for cursor positioning, attribute setting, character
 4  sets etc. Used by Scr::VT100Compatible class
 5 */
 6
 7 /* General setup */
 8 #define               RESET_DEVICE "\033c"
 9 //! enable line wrapping
10 #define           ENABLE_LINE_WRAP "\x1b[7h"
11 //! disable it
12 #define          DISABLE_LINE_WRAP "\x1b[7l"
13
14 /* Scrolling options. Note: there is no way to disable scrolling */
15 //! Whole screen is scrolled on SCROLL_UP/SCROLL_DOWN
16 #define       SCROLL_ENTIRE_SCREEN "\x1b[r"
17 //! Only rows from A to B are scrolled on  SCROLL_UP/SCROLL_DOWN, anything
       above A or below B is not scrolled
18 #define  SCROLL_SCREEN_REGION(A,B) "\x1b["<< (A) << ';' << (B) << 'r'
19
20 //! scroll up
21 #define                  SCROLL_UP "\x1b[M"
22 //! scroll down
23 #define                SCROLL_DOWN "\x1b[D"
24
25 //! make cursor invisible - xterm
26 #define                HIDE_CURSOR "\x1b[?25l"
27
28 //! restore it -xterm
29 #define                SHOW_CURSOR "\x1b[?25h"
30
31 /* Absolute cursor positioning. */
32 //! Set cursor position  to left-top position
33 #define                CURSOR_HOME "\x1b[H"
34 //! Set cursor position to specific y/x (note: y = 1..height, x = 1..width
     )
35 #define             CURSOR_YX(y,x) "\x1b["<< (y) << ';' << (x) << 'H'
36 /* Relative cursor positioning. */
37 //! move cursor one position up
38 #define                  CURSOR_UP "\x1b[A"
39 //! move cursor n positions up
40 #define              CURSOR_UP_(n) "\x1b["<< (n) <<'A'
41 //! move cursor one position down
42 #define                CURSOR_DOWN "\x1b[B"
43 //! move cursor n positions down
44 #define            CURSOR_DOWN_(n) "\x1b["<< (n) <<'B'
45 //! move cursor one position forward
46 #define             CURSOR_FORWARD "\x1b[C"
47 //! move cursor n positions forward
48 #define         CURSOR_FORWARD_(n) "\x1b["<< (n) <<'C'
49 //! move cursor one position backward
50 #define            CURSOR_BACKWARD "\x1b[D"
51 //! move cursor n positions backward
52 #define        CURSOR_BACKWARD_(n) "\x1b["<< (n) <<'D'
53 /* Unsave restores position after last save. */
54 //! One cursor position may be saved
55 #define                SAVE_CURSOR "\x1b[s"
56 //! and restored
57 #define              UNSAVE_CURSOR "\x1b[u"
58
59 /* Erase screen. */
60 //! Erase whole screen
61 #define                      ERASE "\x1b[2J"
62 //! same as above
```

```
 63 #define               ERASE_SCREEN ERASE
 64 //! erase above cursor
 65 #define                   ERASE_UP "\x1b[1J"
 66 //! erase below cursor
 67 #define                 ERASE_DOWN "\x1b[J"
 68
 69 /* Erase line. */
 70 //! erase current line
 71 #define                 ERASE_LINE "\x1b[K"
 72 //! erase current line left from the cursor
 73 #define        ERASE_START_OF_LINE "\x1b[1K"
 74 //! erase current line right from the cursor
 75 #define          ERASE_END_OF_LINE "\x1b[K"
 76
 77 /* a = one of following 23 attributes*/
 78 //! set specific attribute
 79 #define                SET_ATTR(a) "\x1b["<<a<<'m'
 80 //! if you have to set more attributes, separate them by  <<';'<<
 81 #define                   AND_ATTR <<';'<<
 82 /*generalattributes (0-8 without 3 and 6) */
 83 //!resets terminal defaults
 84 #define                  ATTR_RESET 0
 85 //!sets brighter fg color
 86 #define                 ATTR_BRIGHT 1
 87 //!turns off bright (sets darker fg color) note: not supported by most of
       platforms
 88 #define                   ATTR_DIM 2
 89 //!turns on text underline (not supported by MS Windows)
 90 #define            ATTR_UNDERSCORE 4
 91 //!turns on blink (Not supported by MS Windows, most of other
       implementations incompatible)
 92 #define                  ATTR_BLINK 5
 93 //! Inverts bg and fg color (incompatible implementation on MS windows)*/
 94 #define                ATTR_REVERSE 7
 95
 96 #define                 ATTR_HIDDEN 8 /*???*/
 97
 98 /*Foreground (text) colours*/
 99 #define              FG_COLOR_BLACK 30
100 #define                FG_COLOR_RED 31
101 #define              FG_COLOR_GREEN 32
102 #define             FG_COLOR_YELLOW 33
103 #define               FG_COLOR_BLUE 34
104 #define            FG_COLOR_MAGENTA 35
105 #define               FG_COLOR_CYAN 36
106 #define              FG_COLOR_WHITE 37
107
108 /*Background colors*/
109 #define              BG_COLOR_BLACK 40
110 #define                BG_COLOR_RED 41
111 #define              BG_COLOR_GREEN 42
112 #define             BG_COLOR_YELLOW 43
113 #define               BG_COLOR_BLUE 44
114 #define            BG_COLOR_MAGENTA 45
115 #define               BG_COLOR_CYAN 46
116 #define              BG_COLOR_WHITE 47
117
118 /* Character Set settings*/
119 #define                   CS_UK_G0 "\x1b(A"/*G Zero, not oh*/
120 #define                   CS_UK_G1 "\x1b)A"
121 //! Select UK character set
122 #define                      CS_UK CS_UK_G0
```

```
123 #define                   CS_US_G0 "\x1b(B"
124 #define                   CS_US_G1 "\x1b)B"
125 //! Select US character set
126 #define                      CS_US CS_US_G0
127 /* alt character set including frames etc */
128 #define                  CS_ALT_G0 "\x1b(0"
129 #define                  CS_ALT_G1 "\x1b)0"
130 //! Select one of alt character set to use frames etc
131 #define                     CS_ALT CS_ALT_G0
132
133 /* ALT character set symbols (i.e. if CS_ALT is set, then you can use
134  * frames etc.) */
135 #define                  ALT_BLANK '_'
136 #define                ALT_DIAMOND '`' /*no windows equiv.*/
137 #define           ALT_CHECKERBOARD 'a'
138 #define         ALT_HORIZONTAL_TAB 'b'
139 #define              ALT_FORM_FEED 'c'
140 #define                 ALT_RETURN 'd'
141 #define              ALT_LINE_FEED 'e'
142 #define                 ALT_DEGREE 'f'
143 #define             ALT_PLUS_MINUS 'g'
144 #define               ALT_NEW_LINE 'h'
145 #define           ALT_VERTICAL_TAB 'i'
146 #define            ALT_LOWER_RIGHT 'j'
147 #define            ALT_UPPER_RIGHT 'k'
148 #define             ALT_UPPER_LEFT 'l'
149 #define             ALT_LOWER_LEFT 'm'
150 #define               ALT_CROSSING 'n'
151 #define        ALT_HORIZONTAL_LINE 'q'
152 #define                 ALT_LEFT_T 't'
153 #define                ALT_RIGHT_T 'u'
154 #define               ALT_BOTTOM_T 'v'
155 #define                  ALT_TOP_T 'w'
156 #define          ALT_VERTICAL_LINE 'x'
157 #define          ALT_LESS_OR_EQUAL 'y'
158 #define       ALT_GREATER_OR_EQUAL 'z'
159 #define                    ALT_PI '{'  /*no windows equiv.*/
160 #define              ALT_NOT_EQUAL '|'
161 #define               ALT_UK_POUND '}'
162 #define           ALT_CENTERED_DOT '~'
163
164 /*VT100*/
165
166 // interesting extensions
167
168 //! resize entire vscreen (xterm, konsole)
169 #define         RESIZE_SCREEN(A,B) "\x1b[8;"<< (A) << ";"<<(B) << "t"
```

3.20 lib/screen/src/terminfo/capabilities.h++

```
1 #ifndef __CAPABILITIES_H__
2 #define __CAPABILITIES_H__
3
4 namespace Scr
5 {
6
7     namespace TI
```

```
 8    {
 9
10        /*!
11          ordering of booleans in compiled terminfo file. This is
12          based on /usr/include/term.h, by Zeyd M. Ben-Halim, Eric
13          S. Raymond and Thomas E. Dickey.
14         */
15        enum Booleans
16        {
17            AutoLeftMargin,
18            AutoRightMargin,
19            NoEscCtlc,
20            CeolStandoutGlitch,
21            EatNewlineGlitch,
22            EraseOverstrike,
23            GenericType,
24            HardCopy,
25            HasMetaKey,
26            HasStatusLine,
27            InsertNullGlitch,
28            MemoryAbove,
29            MemoryBelow,
30            MoveInsertMode,
31            MoveStandoutMode,
32            OverStrike,
33            StatusLineEscOk,
34            DestTabsMagicSmso,
35            TildeGlitch,
36            TransparentUnderline,
37            XonXoff,
38            NeedsXonXoff,
39            PrtrSilent,
40            Hardsor,
41            NonRevRmcup,
42            NoPadChar,
43            NonDestScrollRegion,
44            CanChange,
45            BackColorErase,
46            HueLightnessSaturation,
47            ColAddrGlitch,
48            CrCancelsMicroMode,
49            HasPrintWheel,
50            RowAddrGlitch,
51            SemiAutoRightMargin,
52            CpiChangesRes,
53            LpiChangesRes
54        };
55
56        /*!
57          ordering of numbers in compiled terminfo file. This is
58          based on /usr/include/term.h, by Zeyd M. Ben-Halim, Eric
59          S. Raymond and Thomas E. Dickey.
60         */
61        enum Numbers
62        {
63            Columns,
64            InitTabs,
65            Lines,
66            LinesOfMemory,
67            MagicCookieGlitch,
68            PaddingBaudRate,
69            VirtualTerminal,
```

```
 70            WidthStatusLine,
 71            NumLabels,
 72            LabelHeight,
 73            LabelWidth,
 74            MaxAttributes,
 75            MaximumWindows,
 76            MaxColors,
 77            MaxPairs,
 78            NoColorVideo,
 79            BufferCapacity,
 80            DotVertSpacing,
 81            DotHorzSpacing,
 82            MaxMicroAddress,
 83            MaxMicroJump,
 84            MicroColSize,
 85            MicroLineSize,
 86            NumberOfPins,
 87            OutputResChar,
 88            OutputResLine,
 89            OutputResHorzInch,
 90            OutputResVertInch,
 91            PrintRate,
 92            WideCharSize,
 93            Buttons,
 94            BitImageEntwining,
 95            BitImageType
 96        };
 97
 98        /*!
 99          ordering of strings in compiled terminfo file. This is
100          based on /usr/include/term.h, by Zeyd M. Ben-Halim, Eric
101          S. Raymond and Thomas E. Dickey.
102         */
103        enum Strings
104        {
105            BackTab,
106            Bell,
107            CarriageReturn,
108            ChangeScrollRegion,
109            ClearAllTabs,
110            ClearScreen,
111            ClrEol,
112            ClrEos,
113            ColumnAddress,
114            CommandCharacter,
115            CursorAddress,
116            CursorDown,
117            CursorHome,
118            CursorInvisible,
119            CursorLeft,
120            CursorMemAddress,
121            CursorNormal,
122            CursorRight,
123            CursorToLl,
124            CursorUp,
125            CursorVisible,
126            DeleteCharacter,
127            DeleteLine,
128            DisStatusLine,
129            DownHalfLine,
130            EnterAltCharsetMode,
131            EnterBlinkMode,
```

```
132             EnterBoldMode,
133             EnterCaMode,
134             EnterDeleteMode,
135             EnterDimMode,
136             EnterInsertMode,
137             EnterSecureMode,
138             EnterProtectedMode,
139             EnterReverseMode,
140             EnterStandoutMode,
141             EnterUnderlineMode,
142             EraseChars,
143             ExitAltCharsetMode,
144             ExitAttributeMode,
145             ExitCaMode,
146             ExitDeleteMode,
147             ExitInsertMode,
148             ExitStandoutMode,
149             ExitUnderlineMode,
150             FlashScreen,
151             FormFeed,
152             FromStatusLine,
153             Init1string,
154             Init2string,
155             Init3string,
156             InitFile,
157             InsertCharacter,
158             InsertLine,
159             InsertPadding,
160             KeyBackspace,
161             KeyCatab,
162             KeyClear,
163             KeyCtab,
164             KeyDc,
165             KeyDl,
166             KeyDown,
167             KeyEic,
168             KeyEol,
169             KeyEos,
170             KeyF0,
171             KeyF1,
172             KeyF10,
173             KeyF2,
174             KeyF3,
175             KeyF4,
176             KeyF5,
177             KeyF6,
178             KeyF7,
179             KeyF8,
180             KeyF9,
181             KeyHome,
182             KeyIc,
183             KeyIl,
184             KeyLeft,
185             KeyLl,
186             KeyNpage,
187             KeyPpage,
188             KeyRight,
189             KeySf,
190             KeySr,
191             KeyStab,
192             KeyUp,
193             KeypadLocal,
```

```
194             KeypadXmit,
195             LabF0,
196             LabF1,
197             LabF10,
198             LabF2,
199             LabF3,
200             LabF4,
201             LabF5,
202             LabF6,
203             LabF7,
204             LabF8,
205             LabF9,
206             MetaOff,
207             MetaOn,
208             Newline,
209             PadChar,
210             ParmDch,
211             ParmDeleteLine,
212             ParmDownsor,
213             ParmIch,
214             ParmIndex,
215             ParmInsertLine,
216             ParmLeftsor,
217             ParmRightsor,
218             ParmRindex,
219             ParmUpsor,
220             PkeyKey,
221             PkeyLocal,
222             PkeyXmit,
223             PrintScreen,
224             PrtrOff,
225             PrtrOn,
226             RepeatChar,
227             Reset1string,
228             Reset2string,
229             Reset3string,
230             ResetFile,
231             Restoresor,
232             RowAddress,
233             Savesor,
234             ScrollForward,
235             ScrollReverse,
236             SetAttributes,
237             SetTab,
238             SetWindow,
239             Tab,
240             ToStatusLine,
241             UnderlineChar,
242             UpHalfLine,
243             InitProg,
244             KeyA1,
245             KeyA3,
246             KeyB2,
247             KeyC1,
248             KeyC3,
249             PrtrNon,
250             CharPadding,
251             AcsChars,
252             PlabNorm,
253             KeyBtab,
254             EnterXonMode,
255             ExitXonMode,
```

```
256             EnterAmMode,
257             ExitAmMode,
258             XonCharacter,
259             XoffCharacter,
260             EnaAcs,
261             LabelOn,
262             LabelOff,
263             KeyBeg,
264             KeyCancel,
265             KeyClose,
266             KeyCommand,
267             KeyCopy,
268             KeyCreate,
269             KeyEnd,
270             KeyEnter,
271             KeyExit,
272             KeyFind,
273             KeyHelp,
274             KeyMark,
275             KeyMessage,
276             KeyMove,
277             KeyNext,
278             KeyOpen,
279             KeyOptions,
280             KeyPrevious,
281             KeyPrint,
282             KeyRedo,
283             KeyReference,
284             KeyRefresh,
285             KeyReplace,
286             KeyRestart,
287             KeyResume,
288             KeySave,
289             KeySuspend,
290             KeyUndo,
291             KeySbeg,
292             KeyScancel,
293             KeyScommand,
294             KeyScopy,
295             KeyScreate,
296             KeySdc,
297             KeySdl,
298             KeySelect,
299             KeySend,
300             KeySeol,
301             KeySexit,
302             KeySfind,
303             KeyShelp,
304             KeyShome,
305             KeySic,
306             KeySleft,
307             KeySmessage,
308             KeySmove,
309             KeySnext,
310             KeySoptions,
311             KeySprevious,
312             KeySprint,
313             KeySredo,
314             KeySreplace,
315             KeySright,
316             KeySrsume,
317             KeySsave,
```

```
318             KeySsuspend,
319             KeySundo,
320             ReqForInput,
321             KeyF11,
322             KeyF12,
323             KeyF13,
324             KeyF14,
325             KeyF15,
326             KeyF16,
327             KeyF17,
328             KeyF18,
329             KeyF19,
330             KeyF20,
331             KeyF21,
332             KeyF22,
333             KeyF23,
334             KeyF24,
335             KeyF25,
336             KeyF26,
337             KeyF27,
338             KeyF28,
339             KeyF29,
340             KeyF30,
341             KeyF31,
342             KeyF32,
343             KeyF33,
344             KeyF34,
345             KeyF35,
346             KeyF36,
347             KeyF37,
348             KeyF38,
349             KeyF39,
350             KeyF40,
351             KeyF41,
352             KeyF42,
353             KeyF43,
354             KeyF44,
355             KeyF45,
356             KeyF46,
357             KeyF47,
358             KeyF48,
359             KeyF49,
360             KeyF50,
361             KeyF51,
362             KeyF52,
363             KeyF53,
364             KeyF54,
365             KeyF55,
366             KeyF56,
367             KeyF57,
368             KeyF58,
369             KeyF59,
370             KeyF60,
371             KeyF61,
372             KeyF62,
373             KeyF63,
374             ClrBol,
375             ClearMargins,
376             SetLeftMargin,
377             SetRightMargin,
378             LabelFormat,
379             SetClock,
```

```
380             DisplayClock,
381             RemoveClock,
382             CreateWindow,
383             GotoWindow,
384             Hangup,
385             DialPhone,
386             QuickDial,
387             Tone,
388             Pulse,
389             FlashHook,
390             FixedPause,
391             WaitTone,
392             User0,
393             User1,
394             User2,
395             User3,
396             User4,
397             User5,
398             User6,
399             User7,
400             User8,
401             User9,
402             OrigPair,
403             OrigColors,
404             InitializeColor,
405             InitializePair,
406             SetColorPair,
407             SetForeground,
408             SetBackground,
409             ChangeCharPitch,
410             ChangeLinePitch,
411             ChangeResHorz,
412             ChangeResVert,
413             DefineChar,
414             EnterDoublewideMode,
415             EnterDraftQuality,
416             EnterItalicsMode,
417             EnterLeftwardMode,
418             EnterMicroMode,
419             EnterNearLetterQuality,
420             EnterNormalQuality,
421             EnterShadowMode,
422             EnterSubscriptMode,
423             EnterSuperscriptMode,
424             EnterUpwardMode,
425             ExitDoublewideMode,
426             ExitItalicsMode,
427             ExitLeftwardMode,
428             ExitMicroMode,
429             ExitShadowMode,
430             ExitSubscriptMode,
431             ExitSuperscriptMode,
432             ExitUpwardMode,
433             MicroColumnAddress,
434             MicroDown,
435             MicroLeft,
436             MicroRight,
437             MicroRowAddress,
438             MicroUp,
439             OrderOfPins,
440             ParmDownMicro,
441             ParmLeftMicro,
```

```
442             ParmRightMicro,
443             ParmUpMicro,
444             SelectCharSet,
445             SetBottomMargin,
446             SetBottomMarginParm,
447             SetLeftMarginParm,
448             SetRightMarginParm,
449             SetTopMargin,
450             SetTopMarginParm,
451             StartBitImage,
452             StartCharSetDef,
453             StopBitImage,
454             StopCharSetDef,
455             SubscriptCharacters,
456             SuperscriptCharacters,
457             TheseCauseCr,
458             ZeroMotion,
459             CharSetNames,
460             KeyMouse,
461             MouseInfo,
462             ReqMousePos,
463             GetMouse,
464             SetAForeground,
465             SetABackground,
466             PkeyPlab,
467             DeviceType,
468             CodeSetInit,
469             Set0DesSeq,
470             Set1DesSeq,
471             Set2DesSeq,
472             Set3DesSeq,
473             SetLrMargin,
474             SetTbMargin,
475             BitImageRepeat,
476             BitImageNewline,
477             BitImageCarriageReturn,
478             ColorNames,
479             DefineBitImageRegion,
480             EndBitImageRegion,
481             SetColorBand,
482             SetPageLength,
483             DisplayPcChar,
484             EnterPcCharsetMode,
485             ExitPcCharsetMode,
486             EnterScancodeMode,
487             ExitScancodeMode,
488             PcTermOptions,
489             ScancodeEscape,
490             AltScancodeEsc,
491             EnterHorizontalHlMode,
492             EnterLeftHlMode,
493             EnterLowHlMode,
494             EnterRightHlMode,
495             EnterTopHlMode,
496             EnterVerticalHlMode,
497             SetAAttributes,
498             SetPglenInch
499         };
500
501     }
502
503 }
```

```
504
505 #endif
```

3.21 lib/screen/src/terminfo/terminfodatabase.h++

```
 1 #ifndef __TERMINFO_DATABASE_H__
 2 #define __TERMINFO_DATABASE_H__
 3
 4 #include"terminfo.h++"
 5 #include"screen.h++"
 6 #include"dictionary.h++"
 7 #include<boost/shared_ptr.hpp>
 8 namespace Scr
 9 {
10
11     namespace TI
12     {
13
14         /*!
15           \brief terminfo database finds system database and fetches
                entries
16          */
17         class TerminfoDatabase
18         {
19         private:
20             std::string path;
21
22             bool status;
23         public:
24             /*!
25               Default constructor looks for terminfo resources
26              */
27             TerminfoDatabase()throw();
28
29             /*!
30               \param name $TERM
31               \return binary file containing term info.
32              */
33             boost::shared_ptr<std::ifstream>
34             OpenFile(const char * name)
35                 throw(FailedToOpenDatabase,
36                       NotSupportedTerminalType,
37                       FailedToLoadDatabaseEntry);
38
39             /*!
40               \return true if database was successfully opened
41              */
42             bool GetDatabaseStatus()throw();
43
44         };
45
46
47     }
48 }
49
50 #endif
```

4 C++ implementation files

4.1 lib/net/netconn.c++

```
 1
 2 #include <tr1/memory>
 3
 4 #include <iostream>
 5 #include <pthread.h>
 6 #include <fcntl.h>
 7 #include <sys/socket.h>
 8 #include <netinet/in.h>
 9 #include <arpa/inet.h>
10 #include <ext/stdio_filebuf.h>
11 #include <unistd.h> /* sleep*/
12 #include <stack>
13 #include <cstring>
14 #include <rexio/net.h++>
15 #include <rexio/throw.h++>
16 #include <rexio/commons.h++>
17 #include <set>
18 using namespace std;
19 using namespace RexIO::Networking;
20 using namespace std::tr1;
21
22 namespace {
23     pthread_mutex_t allprogsStackMutex;
24
25     /*!
26      * Management object for any RootWindows ran by specific instance as
            thread
27      **/
28     class ProgEntry {
29     private:
30         Scr::Tk::RootWindow* d;
31     public:
32         //! Constructor associates object with specific RootWindow
33
34         ProgEntry(Scr::Tk::RootWindow * _d) : d(_d) {
35             ;
36         }
37
38         //! destructor implements basic memory management activity :
                deletes managed
39         //! entity
40
41         ~ProgEntry() {
42             d->Exit(4);
43             cout << "Deleting connection\n";
```

```
44          }
45          friend bool operator<(const ProgEntry& a, const ProgEntry& b);
46
47      };
48
49      // for set<ProgEntry> underlaying tree.
50
51      bool operator<(const ProgEntry& a, const ProgEntry& b) {
52          // comparing addresses of a and b would be wrong, as they may
53          // differ while referencing the same Scr::Tk::RootWindow class
                object.
54          return a.d < b.d;
55      }
56  } // end - empty namespace
57
58  set<ProgEntry> allprogs;
59
60  #include <memory> // for smart pointers
61  void ServerImpl::starter(Scr::Tk::RootWindow * w) {
62      std::auto_ptr<Scr::Tk::RootWindow> prog(w);
63
64      pthread_mutex_lock(&allprogsStackMutex); // prevent accidental stack
65      allprogs.insert(ProgEntry(prog.get())); // data structure destruction
66      pthread_mutex_unlock(&allprogsStackMutex);
67      cout << "Trying to initialize connection" << endl;
68      try {
69          int i = prog->Start(); // start
70          cout << "Connection finished with code " << i << endl; // result
71          // on success
72      }    catch (exception)// exception caught. try to recover by ignoring
            it.
73      {
74          cout << "Connection finished with exception, but app is fine." <<
                endl;
75      }
76      pthread_mutex_lock(&allprogsStackMutex);
77      cout << "Requesting erase of 1 app out of " << allprogs.size() << endl
            ;
78      ;
79
80      // if ProgEntry exists (not deleted by localInterface func)
81      if (allprogs.find(ProgEntry(prog.get())) != allprogs.end())
82          allprogs.erase(ProgEntry(prog.get())); //erase it.
83      cout << endl;
84      pthread_mutex_unlock(&allprogsStackMutex);
85      return;
86  }
87
88
89
90
91  using Scr::FatalException;
92  namespace RexIO { namespace Networking {
93  /*Class for internal use representing reprezenting initialization
94   and termination of connection*/
95  class __Connection {
96  private:
97      int fd;
98      pthread_t th;
99
100     FILE * oF;
101     __gnu_cxx::stdio_filebuf<char> * obuf;
```

```
102     ostream * ostr;
103
104     FILE * iF;
105     __gnu_cxx::stdio_filebuf<char> * ibuf;
106     istream * istr;
107     ServerImpl * simpl;
108     static void * ServeConnection(void * conn);
109     shared_ptr<sockaddr>addr_in;
110 public:
111
112     __Connection(int _fd, ServerImpl * simpl,shared_ptr<sockaddr>&addr);
113     ~__Connection();
114 };
115 /* connection */
116 }}
117 /* callback function serving connection*/
118 //ConnectionFunc f;
119
120 /*for joining "dead" threads*/
121 stack<pthread_t> CleanerStack;
122 pthread_mutex_t CleanerStackMutex;
123
124 __Connection::__Connection(int _fd, ServerImpl * simpl,shared_ptr<sockaddr
        >&addr)
125 :addr_in(addr) {
126     fd = _fd;
127     this->simpl = simpl;
128     cout << "Accepted connection; fd = " << fd << endl;
129     iF = fdopen(fd, "r");
130     oF = fdopen(fd, "w");
131     ibuf = new __gnu_cxx::stdio_filebuf<char>(iF, std::ios_base::in, 1);
132     obuf = new __gnu_cxx::stdio_filebuf<char>(oF, std::ios_base::out, 1);
133     istr = new std::istream(ibuf);
134     ostr = new std::ostream(obuf);
135
136     pthread_create(&th, NULL, ServeConnection, this);
137 }
138
139 __Connection::~__Connection() {
140     delete istr;
141     delete ibuf;
142     fclose(iF);
143
144     delete ostr;
145     delete obuf;
146     fclose(oF);
147     close(fd);
148     pthread_mutex_lock(&CleanerStackMutex);
149     CleanerStack.push(th);
150     pthread_mutex_unlock(&CleanerStackMutex);
151
152 }
153
154 void * __Connection::ServeConnection(void * _conn) {
155     __Connection * conn = (__Connection *) _conn;
156     cout << "in thread for conn fd: " << conn->fd << endl;
157
158     Scr::Tk::RootWindow * rw =
159     conn->simpl->GenWindow(*(conn->istr),*(conn->ostr));
160     conn->simpl->starter(rw);
161     delete conn;
162     return 0;
```

```
163 }
164
165 void * CleanerFunc(void * activity_mark) {
166
167     while (* static_cast<bool*> (activity_mark)) {
168         sleep(2);
169         pthread_mutex_lock(&CleanerStackMutex);
170         while (!CleanerStack.empty()) {
171             pthread_t t = CleanerStack.top();
172             pthread_join(t, NULL);
173             CleanerStack.pop();
174             cout << "Joined thread" << endl;
175         }
176         pthread_mutex_unlock(&CleanerStackMutex);
177     }
178     return 0;
179 }
180
181 ServerImpl::ServerImpl() {
182     ;
183 }
184
185 void ServerImpl::Start(int portnum) {
186
187     active = true;
188     struct sockaddr_in srv;
189     socklen_t socklen;
190     int iSockFD;
191     if ((iSockFD = socket(PF_INET, SOCK_STREAM, 0)) < 0)
192         THROWP(FatalException, "socket");
193
194     int opt = 1, len = 4;
195     setsockopt(iSockFD, SOL_SOCKET, SO_REUSEADDR, &opt, len);
196
197     memset(&srv, 0, sizeof (srv));
198     srv.sin_family = AF_INET;
199     srv.sin_addr.s_addr = htonl(INADDR_ANY);
200     srv.sin_port = htons(portnum);
201
202     socklen = sizeof (srv);
203
204     if (bind(iSockFD, (struct sockaddr *) & srv, socklen) < 0)
205         THROWP(FatalException, "bind");
206
207     int fd;
208     listen(iSockFD, 5);
209
210     if (pthread_mutex_init(&CleanerStackMutex, NULL))
211         THROWP(FatalException, "mutex_initialize");
212     pthread_t cleaner_thread;
213
214     if (fcntl(iSockFD, F_SETFL, O_NDELAY) < 0)
215         THROWP(FatalException, "Can't make nonblocking socked");
216
217     if (pthread_create(&cleaner_thread, NULL, CleanerFunc, &active))
218         THROWP(FatalException, "pthread_create (&cleaner_thread,NULL,
            CleanerFunc,NULL)");
219
220     while (active) {
221         tr1::shared_ptr<sockaddr>addr_in(new sockaddr);
222         fd = accept(iSockFD, addr_in.get(), & socklen);
223         if (fd > 0)
```

```
224             new __Connection(fd,this,addr_in);
225         else
226             usleep(1000);
227     }
228     if (pthread_join(cleaner_thread, NULL))
229         THROWP(FatalException, "pthread_join(&cleaner_thread,NULL)");
230     pthread_mutex_destroy(&CleanerStackMutex);
231     close(iSockFD);
232 }
233
234 void ServerImpl::Stop() {
235     active = false;
236     allprogs.clear();
237 }
```

4.2 lib/net/netconnmgr.c++

```
 1 #incl
 2 pthread_mutex_t allprogsStackMutex;
 3
 4 class ProgEntry
 5 {
 6 private:
 7     Demo* d;
 8 public:
 9     ProgEntry(Demo * _d):d(_d){;}
10     ~ProgEntry()
11     {
12         d->Exit(4);
13         cout << "-";
14     }
15     friend bool operator<(const ProgEntry& a, const ProgEntry& b);
16
17 };
18
19 // for set<ProgEntry> underlaying tree.
20 bool operator<(const ProgEntry& a, const ProgEntry& b) {
21     // comparing addresses of a and b would be wrong, as they may
22     // differ while referencing the same Demo class object.
23     return a.d < b.d;
24 }
25
26 set<ProgEntry> allprogs;
27 Server s;
28
29 void starter(std::istream & in, std::ostream & out)
30 {
31     Demo prog(in,out);
32
33     pthread_mutex_lock(&allprogsStackMutex);// prevent accidental stack
34     allprogs.insert(ProgEntry(&prog));// data structure destruction
35     pthread_mutex_unlock(&allprogsStackMutex);
36     cout << "Trying to initialize connection" << endl;
37     try
38     {
39     int i = prog.Start(); // start
40     cout << "Connection finished with code " << i <<endl;// result
```

```
41                                  // on success
42     }
43     catch (exception)// exception caught. try to recover by ignoring it.
44     {
45     cout << "Connection finished with exception, but app is fine." <<endl;
46     }
47     pthread_mutex_lock(&allprogsStackMutex);
48     cout << "Requesting erase of 1 app out of "<<allprogs.size()<<endl;;
49
50     // if ProgEntry exists (not deleted by localInterface func)
51     if (allprogs.find(ProgEntry(&prog)) != allprogs.end())
52     allprogs.erase(ProgEntry(&prog));//erase it.
53     cout << endl;
54     pthread_mutex_unlock(&allprogsStackMutex);
55     return;
56 }
```

4.3 lib/rcurses/src/rcurses.c++

```
 1 #include "rcurses.h"
 2
 3
 4 static SCREEN* maincontext = NULL;
 5
 6 WINDOW* initscr()
 7 {
 8     maincontext = new CursesRootWindow();
 9     stdscr = &(maincontext->GetScreenPtr());
10     return stdscr;
11 }
12
13 int endwin()
14 {
15     if(!maincontext)
16         return ERR;
17     delete maincontext;
18     stdscr = NULL;
19     return OK;
20 }
21
22 int mvwaddstr(WINDOW *win, int y, int x, const char *str)
23 {
24     (*win)
25         << DisplayStyle(Fg::White, Fg::Bright, Bg::Black)
26         <<Control::GotoYX(y, x) << str;
27     return OK;
28 }
29
30 int wrefresh(WINDOW *win)
31 {
32     (*win) << Control::Refresh << Control::Clear;
33     return OK;
34 }
35
36 int wgetch(WINDOW *win)
37 {
38 /*      LocalScreen *lscr = dynamic_cast<LocalScreen *>(stdscr);
```

```
39
40
41     if(lscr->TestForResize(lscr->input.FD())) // PRIVATE
42     return KEY_RESIZE;*/
43     return 10;
44 }
45
46 int resize_term(int lines, int columns)
47 {
48     maincontext->OnResize(lines, columns);
49     return OK;
50 }
```

4.4 lib/screen/src/core/bufferedinput.c++

```
 1 #include <iostream>
 2 #include <string>
 3 #include"bufferedinput.h++"
 4 #include <sys/select.h>
 5 #include <sys/ioctl.h>
 6 #include <unistd.h>
 7 using namespace Scr;
 8 using namespace std;
 9 #include <cassert>
10 bool BufferedInput::KbHit()const throw()
11 {
12     struct timeval tv;
13     fd_set fds;
14     tv.tv_sec = 0;
15     tv.tv_usec = 10000;
16     FD_ZERO(&fds);
17     FD_SET(FD(), &fds); //STDIN_FILENO is 0
18     return select(FD()+1, &fds, NULL, NULL, &tv);
19 }
20
21 void BufferedInput::ForceBuffer()const throw()
22 {
23     while(!KbHit());
24     do
25     {
26         if (currentCharBufferSize==maxCharBufferSize)
27         {// yet something to read, but nowwhere to store it
28             filledToCapacity=true;
29             return;
30         }
31         charBuffer[currentCharBufferSize++]=stream.get();
32     }
33     while (KbHit());
34     RexIOLog(LogLevelModerate) << "ForceBuffer resulted in reading "
35          << currentCharBufferSize -1  << "characters:"
36          << DebugInfo()
37          << endl;
38
39 }
40
41 void BufferedInput::DoBuffer()const throw()
42 {
```

```
43      filledToCapacity=false;
44      //to allow at least one UnGet any time
45      charBuffer[0]=charBuffer[currentCharBufferSize-1];
46      currentCharBufferIndex=1;
47      if (q!=NULL)
48      {
49          if (q->empty())
50          {
51              q=NULL;
52          }
53          else
54          {
55              filledToCapacity=true;
56              charBuffer[1]=q->front();
57              q->pop();
58              currentCharBufferSize=2;
59              return;
60          }
61      }
62      currentCharBufferSize=
63          stream.readsome(
64              static_cast<char*>(&charBuffer[1]),
65              static_cast<std::streamsize>(maxCharBufferSize-1));
66      currentCharBufferSize++;
67      if (currentCharBufferSize==maxCharBufferSize)
68          filledToCapacity=true;
69      if (currentCharBufferSize == 1)// no text read, while it
70          // has to be read.
71          ForceBuffer();
72  }
73
74  string BufferedInput::String()throw()
75  {
76      return string(charBuffer,currentCharBufferSize);
77  }
78
79  string BufferedInput::DebugInfo()throw()
80  {
81      return
82      (static_cast<const BufferedInput &>(*this)).DebugInfo();
83  }
84  const string BufferedInput::DebugInfo()const throw()
85  {
86
87      std::stringstream dss;
88      for (Uint i = 1; i< currentCharBufferSize; i++)
89          dss << i<<':'
90              << ( (charBuffer[i]>31 && charBuffer[i]<0x7f) ?
91                   charBuffer[i] : '?' ) << '('
92              << static_cast<int>(charBuffer[i]) << ")  " ;
93      return dss.str();
94  }
```

4.5 lib/screen/src/core/commons.c++

```
1  #include "commons.h++"
2
```

```
 3 using namespace Scr;
 4
 5 Vector::Vector(Sint _rows, Sint _cols)
 6     :rows(_rows), cols(_cols){;}
 7
 8 Position::Position(Uint _row, Uint _col)
 9     :row(_row), col(_col){;}
10
11 Position Position::operator+(const Position& pos)
12 {
13     return Position(row + pos.row, col + pos.col);
14 }
15
16 Position Position::operator+(const Size& size)
17 {
18     return Position(row + size.height, col + size.width);
19 }
20
21 Position Position::operator+(const Vector& vec)
22 {
23     return Position(row + vec.rows, col + vec.cols);
24 }
25
26 Position& Position::operator+=(const Position& pos)
27 {
28     row += pos.row;
29     col += pos.col;
30     return *this;
31 }
32
33 Position& Position::operator+=(const Size& size)
34 {
35     row += size.height;
36     col += size.width;
37     return *this;
38 }
39
40 Position& Position::operator+=(const Vector& vec)
41 {
42     row += vec.rows;
43     col += vec.cols;
44     return *this;
45 }
46
47 Position Position::operator-(const Position &pos)
48 {
49     return Position(row - pos.row, col - pos.col);
50 }
51
52 Position Position::operator-(const Size& size)
53 {
54     return Position(row - size.height, col - size.width);
55 }
56
57 Position Position::operator-(const Vector& vec)
58 {
59     return Position(row - vec.rows, col - vec.cols);
60 }
61
62 Position& Position::operator-=(const Position& pos)
63 {
64     row -= pos.row;
```

```
65     col -= pos.col;
66     return *this;
67 }
68
69 Position& Position::operator-=(const Size& size)
70 {
71     row -= size.height;
72     col -= size.width;
73     return *this;
74 }
75
76 Position& Position::operator-=(const Vector& vec)
77 {
78     row -= vec.rows;
79     col -= vec.cols;
80     return *this;
81 }
82
83 Size::Size(Uint _height, Uint _width)
84     :height(_height),width(_width){;}
```

4.6 lib/screen/src/core/connection.c++

```
 1 #include"screen.h++"
 2 #include<iostream>
 3 #include"screenbuffer.h++"
 4 #include"genericscreen.h++"
 5 #include"vt100compatible.h++"
 6 #include"localscreen.h++"
 7 #include"remotescreen.h++"
 8 #include"screenconnection.h++"
 9 #include<sys/ioctl.h>
10 #include"fileno_hack.h++"
11 #include"core.h++"
12 #include"throw.h++"
13 #include"keyboard.h++"
14 #include"terminfo.h++"
15 #include"terminfoenabled.h++"
16
17 #ifndef TIOCGWINSZ
18 #error "TIOCGWINSZ not supported"
19 #endif  /* TIOCGWINSZ */
20
21 Scr::Connection::Connection(std::istream & _input, std::ostream & _output)
22         throw()
23 {
24     int ofd = fileno_hack(_output);
25     // ...
26     // terminal connection type detection here
27
28     try
29     {
30         TI::TerminfoCore::Initialize();
31         if (isatty(ofd))
32             screen = std::auto_ptr<Screen>
33                     (
34                         new RScreen<LocalScreen,TerminfoEnabledScreen>
```

```
35                              (*this,_input,_output)
36                      );
37          else
38              screen = std::auto_ptr<Screen>
39                      (
40                          new RScreen<RemoteScreen,TerminfoEnabledScreen>
41                              (*this,_input,_output)
42                      );
43      }
44      catch (TI::FailedToOpenDatabase)
45      {
46          //Failed to initialize TERMINFO driver, use fallback
                implementation
47          //instead (VT100Compatible is class implementing screen operations
48          //using hardcoded I/O sequences - works quite well for most
                popular
49          //terminal types.)
50          if (isatty(ofd))
51              screen = std::auto_ptr<Screen>
52                      (
53                          new RScreen<LocalScreen,VT100Compatible>
54                              (*this,_input,_output)
55                      );
56          else
57              screen = std::auto_ptr<Screen>
58                      (
59                          new RScreen<RemoteScreen,VT100Compatible>
60                              (*this,_input,_output)
61                      );
62      }
63      catch (...)
64      {
65          //other posible reason of failure - no idea what to do. leave
66          //screen==NULL - Connection::Start will return StartFailed,
67          //but nothing bad will happen. Nothing at all will happen  if
68          //Connection::Start will never be called.
69          ;
70          // (it's auto_ptr, and by default it points "to nothing")
71      }
72  }
73  //end of Scr::Connection::Connection
74  //   (std::istream & _input, std::ostream & _output)
75
76  int Scr::Connection::Start(int argc, char **argv)
77      throw(StartFailed,Screen::IllegalCharacter)
78  {
79      return Start();
80  }
81
82  int Scr::Connection::Start()
83      throw(StartFailed,Screen::IllegalCharacter)
84  {
85      int code;
86      __ScreenConnection * sc = dynamic_cast<__ScreenConnection*>( screen.
            get());
87      if (sc)
88      {
89          try
90          {
91              code =sc->ProcessConnection();
92          }
93          catch (Broken)
```

```
 94         {
 95             throw; // controlled rethrow
 96         }
 97     }
 98     else
 99         THROW(StartFailed);
100
101     OnExit(code);
102     return code;
103 }
104
105 void Scr::Connection::Exit(int code)throw(StopFailed)
106 {
107     __ScreenConnection * sc = dynamic_cast<__ScreenConnection*>(screen.get
            ());
108     if (sc)
109     {
110         sc->ExitConnection(code);
111     }
112     else
113         THROW(StopFailed);
114 }
115
116 void Scr::Connection::OnStart()throw()
117 {
118     ;
119 }
120
121 void Scr::Connection::OnResize(Uint rows, Uint cols)throw()
122 {
123     ;
124 }
125
126 void Scr::Connection::OnKeyDown(Key key)throw()
127 {
128     ;
129 }
130
131 void Scr::Connection::OnExit(int code)throw()
132 {
133     ;
134 }
135
136 Scr::Connection::~Connection()throw()
137 {
138     ;//screen destroyed by auto_ptr
139 //  TI::TerminfoCore::FreeTerminfoEntry(); called by destructor of *screen
        , if
140 //  it is TerminfoEnabledScreen
141 }
```

4.7 lib/screen/src/core/displaystyle.c++

```
1 #include"screen.h++"
2
3 Scr::DisplayStyle::DisplayStyle(Fg::Color _fgColor,
4                                 Fg::Style _fgStyle,
```

```
5                                Bg::Color _bgColor)throw()
6 {
7     style=0;
8     SetFgColor(_fgColor);
9     SetFgStyle(_fgStyle);
10    SetBgColor(_bgColor);
11 }
12
13 Scr::DisplayStyle::DisplayStyle(const DisplayStyle & base)throw()
14 {
15    style=base.style;
16 }
17
18 Scr::DisplayStyle::DisplayStyle()throw(){style=0;}
```

4.8 lib/screen/src/core/exception.c++

```
1 #include "commons.h++"
2 #include <iostream>
3
4 using namespace Scr;
5 Exception::Exception(std::string _m)throw()
6     :std::exception()
7 {
8     using namespace std;
9     RexIOLog(LogLevelLow) << "Exception occurred. Message:\n " << _m << "\
          n";
10    message=std::tr1::shared_ptr<std::string>(new std::string(_m));
11 }
12
13 Exception::Exception(const Exception& _base)throw()
14    :std::exception()
15    ,message(_base.message)
16 {
17    ;
18 }
19
20 const char* Exception::what() const throw()
21 {
22    return message->c_str();
23 }
24
25 Exception::~Exception()throw()
26 {
27    ;
28 }
```

4.9 lib/screen/src/core/glyphwidth.c++

```
1 #include "commons.h++"
2
3 extern "C" {
```

```
 4 #include "wcwidth.c"
 5 }
 6 using namespace Scr;
 7
 8 std::bitset<(1<<17)*2> Scr::GlyphWidth::glyphWidth;
 9
10 // unsigned long Scr::width(wchar_t c)
11 // {
12 //  return mk_wcwidth(c);
13 // }
14
15 GlyphWidth::GlyphWidth()
16 {
17     for(Uint i = 0;i<glyphWidth.size();i+=2) {
18         int w = mk_wcwidth((wchar_t)(i>>1));
19         if(w == 1) // make width 1 characters require only one lookup
20             glyphWidth[i] = true;
21         else if(w == 2) {
22             glyphWidth[i] = false;
23             glyphWidth[i+1] = true;
24         }
25         else {
26             glyphWidth[i] = false;
27             glyphWidth[i+1] = false;
28         }
29     }
30 }
```

4.10 lib/screen/src/core/keyboard.c++

```
 1 #include "throw.h++"
 2 #include "keyboard.h++"
 3 #include "commons.h++"
 4 using namespace Scr;
 5
 6 #define KEYD(x) case x: return # x
 7 const char * Key::GetKeyName()throw()
 8 {
 9     switch (key)
10     {
11         KEYD(Enter);
12         KEYD(Tab);
13         KEYD(Escape);
14         KEYD(Delete);
15         KEYD(Backspace);
16
17         KEYD(F1);
18         KEYD(F2);
19         KEYD(F3);
20         KEYD(F4);
21
22         KEYD(F5);
23         KEYD(F6);
24         KEYD(F7);
25         KEYD(F8);
26
27         KEYD(F9);
```

```
28          KEYD(F10);
29          KEYD(F11);
30          KEYD(F12);
31
32      case Left: return "\342\206\220";
33      case Up: return "\342\206\221";
34      case Right: return "\342\206\222";
35      case Down: return "\342\206\223";
36          /// KEYD(Up);
37          //KEYD(Down);
38          //KEYD(Right);
39          //KEYD(Left);
40
41          KEYD(CtrlUp);
42          KEYD(CtrlDown);
43          KEYD(CtrlRight);
44          KEYD(CtrlLeft);
45
46          KEYD(Insert);
47          KEYD(Home);
48          KEYD(PageUp);
49          KEYD(PageDown);
50          KEYD(End);
51
52      default:
53          return "unknown";
54      }
55 }
56
57 char Key::GetBasicKey()throw(NotABasicKey)
58 {
59      EASSERT(IsABasicKey(),NotABasicKey);
60      char masked = key& basicKeyMask;
61      return masked;
62 }
63
64 Key::Special Key::GetSpecialKey()throw(NotASpecialKey)
65 {
66      EASSERT(IsASpecialKey(),NotASpecialKey);
67      return static_cast<Key::Special>(key);
68 }
```

4.11 lib/screen/src/core/screenbase.c++

```
 1 #include <iostream>
 2 #include"screen.h++"
 3 #include"screenbase.h++"
 4 #include"subscreen.h++"
 5
 6 using namespace Scr;
 7
 8 Scr::ScreenBase::ScreenBase()
 9      :Screen(),aPoint(0,0)
10 {}
11
12 Uint Scr::ScreenBase::GetY()const throw()
13 {
```

```
14     return aPoint.row;
15 }
16 Uint Scr::ScreenBase::GetX()const throw()
17 {
18     return aPoint.col;
19 }
20
21 void Scr::ScreenBase::AddText(const std::string & text, Uint cols)
22              throw(PrintOutOfRange, IllegalCharacter)
23 {
24     AddText(text.c_str(),cols);
25 }
```

4.12 lib/screen/src/core/screenbuffer.c++

```
 1 #include "screenbuffer.h++"
 2 #include <cstring>
 3 #include <cstdlib>
 4 #include <iostream>
 5 using namespace Scr;
 6
 7 ////////////////////////////////////////////////
 8 //
 9 //   ScreenCharacter
10
11 ScreenCharacter::ScreenCharacter(Uint _c, const DisplayStyle & _style)
12     :style(_style),c(_c){;}
13
14 ScreenCharacter & ScreenCharacter::operator=(const ScreenCharacter & other
       )
15 {
16     c=other.c;
17     style=other.style;
18     return *this;
19 }
20
21 bool ScreenCharacter::operator==(const ScreenCharacter & other)
22 {
23     return (c == other.c) && (style == other.style);
24 }
25
26 bool ScreenCharacter::operator!=(const ScreenCharacter & other)
27 {
28     return (c != other.c) || (style != other.style);
29 }
30
31 //   ScreenRow
32 ScreenRow::ScreenRow(Uint width,
33                      const ScreenCharacter & character)
34     :characters(width,character){;}
35
36 void ScreenRow::Resize(Uint newWidth, const ScreenCharacter & character)
37 {
38     characters.resize(newWidth,character);
39 }
40
41 ScreenRow & ScreenRow::operator=(const ScreenRow & other)
```

```
42 {
43     characters=other.characters;
44     return *this;
45 }
46
47 bool ScreenRow::operator==(const ScreenRow & other)
48 {
49     if (characters.size()!=other.characters.size())
50     return false;
51     std::vector<ScreenCharacter>::iterator i = characters.begin();
52     std::vector<ScreenCharacter>::const_iterator j = other.characters.
           begin();
53     for ( ; i!= characters.end(); i++,j++)
54     if ( !((*i) == (*j)) )
55         return false;
56
57     return true;
58 }
59
60 bool ScreenRow::operator!=(const ScreenRow & other)
61 {
62     if (characters.size()!=other.characters.size())
63     return true;
64     std::vector<ScreenCharacter>::iterator i = characters.begin();
65     std::vector<ScreenCharacter>::const_iterator j = other.characters.
           begin();
66     for ( ; i!= characters.end(); i++,j++)
67     if ( !((*i) == (*j)) )
68         return true;
69
70     return false;
71 }
72
73 Uint ScreenRow::GetWidth() const
74 {
75     return characters.size();
76 }
77
78 //
       ////////////////////////////////////////////////////////////////////////

79 //
80 //   ScreenBuffer
81
82 ScreenBuffer::ScreenBuffer(Uint _rows, Uint columns,
83                const ScreenCharacter & character)
84     :rows(_rows,ScreenRow(columns,character)){;}
85
86 ScreenBuffer & ScreenBuffer::operator=(const ScreenBuffer & other)
87 {
88     rows=other.rows;
89     return *this;
90 }
91
92 bool ScreenBuffer::operator==(const ScreenBuffer & other)
93 {
94     if (rows.size()!=other.rows.size())
95     return false;
96     std::vector<ScreenRow>::iterator i = rows.begin();
97     std::vector<ScreenRow>::const_iterator j = other.rows.begin();
98     for ( ; i!= rows.end(); i++,j++)
99     if ( !((*i) == (*j)) )
```

```
100         return false;
101     return true;
102 }
103
104 bool ScreenBuffer::operator!=(const ScreenBuffer & other)
105 {
106     if (rows.size()!=other.rows.size())
107     return true;
108     std::vector<ScreenRow>::iterator i = rows.begin();
109     std::vector<ScreenRow>::const_iterator j = other.rows.begin();
110     for ( ; i!= rows.end(); i++,j++)
111     if ( !((*i) == (*j)) )
112         return true;
113     return false;
114 }
115
116 void ScreenBuffer::Resize(Uint newHeight,
117               Uint newWidth,
118               const ScreenCharacter & character)
119 {
120
121     for (std::vector<ScreenRow>::iterator i = rows.begin();
122      i!= rows.end(); i++)
123     i->Resize(newWidth,character);
124
125     if (newHeight < GetHeight())
126     rows.erase(rows.end()-(GetHeight()-newHeight),rows.end());
127     else if (newHeight > GetHeight())
128     rows.insert(rows.end(),newHeight-GetHeight(),
129             ScreenRow(newWidth,character));
130 }
131
132 Uint ScreenBuffer::GetWidth()const
133 {
134     return rows[0].GetWidth();
135 }
136
137 Uint ScreenBuffer::GetHeight() const
138 {
139     return rows.size();
140 }
141
142 void ScreenBuffer::Fill(const ScreenCharacter & character)
143 {
144     rows[0].characters.assign(GetWidth(),character);
145     rows.assign(GetHeight(),rows[0]);
146 }
```

4.13 lib/screen/src/core/screen.c++

```
1 #include <termios.h>
2 #include <iostream>
3 #include"screen.h++"
4 #include"subscreen.h++"
5 #include"screenbuffer.h++"
6 #include"genericscreen.h++"
7 #include"vt100compatible.h++"
```

```
 8 #include"localscreen.h++"
 9 #include"throw.h++"
10
11 using namespace std;
12 using namespace Scr;
13
14 Control::_PositionYX Control::GotoYX(Uint _y, Uint _x)
15 {
16     return _PositionYX(_y,_x);
17 }
18
19 Screen::Screen()throw()//:ss(new std::stringstream)
20 {
21     RexIOLog(LogLevelModerate) << "Screen - base" << endl;
22 }
23
24 Screen::~Screen()throw()
25 {
26     RexIOLog(LogLevelModerate) << "~Screen - base" << endl;
27 }
28
29 template<class T>
30 inline Screen& operatorIOS(Screen & screen,const T & whatto)
31 {
32     std::ostringstream ss;
33     ss <<whatto;
34
35     screen.AddText((ss.str().c_str()));
36     return screen;
37 }
38 namespace Scr{
39 Screen& operator<<(Screen & screen,const  wchar_t(&  whatto)[9])
40 {
41     screen.AddText(whatto);
42     return screen;
43 }
44
45 /*Screen& operator<<(Screen & screen,const EString & whatto)
46 {
47     screen.AddText(whatto);
48     return screen;
49 }*/
50
51 Screen& operator<<(Screen & screen,const std::wstring & whatto)
52 {
53     screen.AddText(whatto);
54     return screen;
55 }
56
57 Screen& operator<<(Screen & screen,wchar_t const * const & whatto)
58 {
59     screen.AddText(whatto);
60     return screen;
61 }
62
63 Screen& operator<<(Screen & screen,const std::string & whatto)
64 {
65     screen.AddText(whatto);
66     return screen;
67 }
68
69 Screen& operator<<(Screen & screen,char const * const & whatto)
```

```
 70 {
 71     screen.AddText(whatto);
 72     return screen;
 73 }
 74
 75 Screen& operator<<(Screen & screen,char * const & whatto)
 76 {
 77     screen.AddText(whatto);
 78     return screen;
 79 }
 80
 81 Screen& operator<<(Screen & screen,const Fg::Color & whatto)
 82 {
 83     screen.SetFgColor(whatto);
 84     return screen;
 85 }
 86
 87 Screen& operator<<(Screen & screen,const Fg::Style & whatto)
 88 {
 89     screen.SetFgStyle(whatto);
 90     return screen;
 91 }
 92
 93 Screen& operator<<(Screen & screen,const Bg::Color & whatto)
 94 {
 95     screen.SetBgColor(whatto);
 96     return screen;
 97 }
 98
 99 Screen& operator<<(Screen & screen,const Control::_PositionYX & whatto)
100 {
101     screen.GotoYX(whatto.row,whatto.col);
102     return screen;
103 }
104
105 Screen& operator<<(Screen & screen,const Control::_Refresh & whatto)
106 {
107     screen.Refresh();
108     return screen;
109 }
110
111 Screen& operator<<(Screen & screen,const Control::_Clear & whatto)
112 {
113     screen.Clear();
114     return screen;
115 }
116
117 Screen& operator<<(Screen & screen,const DisplayStyle & whatto)
118 {
119     screen.SetFgStyle(whatto.GetFgStyle());
120     screen.SetFgColor(whatto.GetFgColor());
121     screen.SetBgColor(whatto.GetBgColor());
122
123     return screen;
124 }
125
126 Screen& operator<<(Screen & screen, unsigned int whatto)
127 {
128     return operatorIOS(screen,whatto);
129 }
130 Screen& operator<<(Screen & screen,int  whatto)
131 {
```

```
132     return operatorIOS(screen,whatto);
133 }
134 Screen& operator<<(Screen & screen,std::_Setw  whatto)
135 {
136     return operatorIOS(screen,whatto);
137 }
138 Screen& operator<<(Screen & screen, unsigned long whatto)
139 {
140     return operatorIOS(screen,whatto);
141 }
142 Screen& operator<<(Screen & screen, long  whatto)
143 {
144     return operatorIOS(screen,whatto);
145 }
146
147 Screen& operator<<(Screen & screen, char  whatto)
148 {
149     screen.AddCharacter(whatto);
150     return screen;
151 }
152 Screen& operator<<(Screen & screen, wchar_t  whatto)
153 {
154     screen.AddCharacter(whatto);
155     return screen;
156 }
157
158 }
```

4.14 lib/screen/src/core/utf8.c++

```
 1 #include <iostream>
 2 #include"screen.h++"
 3 #include"throw.h++"
 4 #include"utf8.h++"
 5
 6 using namespace Scr;
 7 using namespace std;
 8
 9 #ifdef DO_VALIDATE_UTF_8_OUTPUT
10 # define VALIDATE_TRAILING(x)   \
11     if ( (c[x]&0xC0)!=0x80)  \
12         THROW(Screen::InvalidTrailingByte)
13 #else
14 # define VALIDATE_TRAILING(x)
15 #endif //DO_VALIDATE_UTF_8_OUTPUT
16
17 wchar_t Scr::DecodeUTF8(const char ** pstr)
18     throw(Screen::InvalidUTF8)
19 {
20     Uint result;
21     unsigned char c[4];
22     c[0]=**pstr;
23     //result=c[0];
24
25     if ( (c[0] & 0xF8) == 0xF0) // 4 byte char
26     {
27         (*pstr)++;
```

```
28          c[1]=**pstr;
29          VALIDATE_TRAILING(1); // only some values for trailing bytes
30          // are valid
31          (*pstr)++;
32          c[2]=**pstr;
33          VALIDATE_TRAILING(2);
34          (*pstr)++;
35          c[3]=**pstr;
36          VALIDATE_TRAILING(3);
37          c[0]&=0x0F;
38          c[1]&=0x7F;
39          c[2]&=0x7F;
40          c[3]&=0x7F;
41          result = c[3]+(((Uint)c[2])<<6)+(((Uint)c[1])<<12)
42              +(((Uint)c[0])<<18);
43
44 #ifdef DO_VALIDATE_UTF_8_OUTPUT
45          if (result < (1<<16))// overlong UTF8 encoding FORBIDDEN
46              // according to RFC 3629
47          {
48              (*pstr)-=3;
49              THROW(Screen::OverlongUTF8Encoding);
50          }
51 #endif
52      }
53      else if ((c[0] & 0xF0) == 0xE0) // 3 byte char
54      {
55          (*pstr)++;
56          c[1]=**pstr;
57          VALIDATE_TRAILING(1);
58          (*pstr)++;
59          c[2]=**pstr;
60          VALIDATE_TRAILING(2);
61          c[0]&=0x1F;
62          c[1]&=0x7F;
63          c[2]&=0x7F;
64          result = c[2]+(((Uint)c[1])<<6)+(((Uint)c[0])<<12);
65
66 #ifdef DO_VALIDATE_UTF_8_OUTPUT
67          if (result < (1<<11))// overlong UTF8 encoding FORBIDDEN
68              // according to RFC 3629
69          {
70              (*pstr)-=2;
71              THROW(Screen::OverlongUTF8Encoding);
72          }
73 #endif
74      }
75      else if ((c[0] & 0xE0) == 0xC0) // 2 byte char
76      {
77          (*pstr)++;
78          c[1]=**pstr;
79          VALIDATE_TRAILING(1);
80
81          c[0]&=0x3F;
82          c[1]&=0x7F;
83
84          result = c[1]+(((Uint)c[0])<<6);
85
86 #ifdef DO_VALIDATE_UTF_8_OUTPUT
87          if (result < (1<<7))// overlong UTF8 encoding FORBIDDEN
88              // according to RFC 3629
89          {
```

```
 90             (*pstr)--;
 91             THROW(Screen::OverlongUTF8Encoding);
 92         }
 93 #endif
 94     }
 95     else
 96 #ifdef DO_VALIDATE_UTF_8_OUTPUT
 97         if ((c[0] & 0x80) == 0)
 98 #endif
 99             // 1 byte char
100         {
101             result=c[0];
102         }
103 #ifdef DO_VALIDATE_UTF_8_OUTPUT
104         else
105             THROW(Screen::InvalidFirstByte);
106 #endif
107 
108     return result;
109 }
110 
111 //std namespace specifier for ostream is said explicitly. It would
112 //compilewithout it as "using namespace" is above, but doxygen
113 //dislikes such inconsistencies
114 void Scr::EncodeUTF8(std::ostream & o, Uint c)throw()
115 {
116     if (c<(1<<7))// 7 bit Unicode encoded as plain ascii
117     {
118         o << static_cast<char>(c);
119         return;
120     }
121     if (c<(1<<11))// 11 bit Unicode encoded in 2 UTF-8 bytes
122     {
123         o << static_cast<unsigned char>((c>>6)|0xC0)
124           << static_cast<unsigned char>((c&0x3F)|0x80);
125         return;
126     }
127     if (c<(1<<16))// 16 bit Unicode encoded in 3 UTF-8 bytes
128     {
129         o << static_cast<unsigned char>(((c>>12))|0xE0)
130           << static_cast<unsigned char>(((c>>6)&0x3F)|0x80)
131           << static_cast<unsigned char>((c&0x3F)|0x80);
132         return;
133     }
134 
135     if (c<(1<<21))// 21 bit Unicode encoded in 4 UTF-8 bytes
136     {
137         o << static_cast<unsigned char>(((c>>18))|0xF0)
138           << static_cast<unsigned char>(((c>>12)&0x3F)|0x80)
139           << static_cast<unsigned char>(((c>>6)&0x3F)|0x80)
140           << static_cast<unsigned char>((c&0x3F)|0x80);
141         return;
142     }
143 }
144 
145 Uint Scr::CharLengthUTF8(const char * s)
146     throw(Screen::InvalidUTF8)
147 {
148     if ( (*s) & 0x80 )
149     { // more than 1 byte
150         if ( ( (*s) & 0xF8) == 0xF0) // 4-byte
151             return 4;
```

```
152         if ( ( (*s) & 0xF0) == 0xE0)
153             return 3;
154         if ( ( (*s) & 0xE0) == 0xC0)
155             return 2;
156     }
157     else
158         return 1;
159     THROW(Screen::InvalidUTF8);
160 }
161
162 Uint Scr::StringLengthUTF8(const char * s)
163     throw(Screen::InvalidUTF8)
164 {
165     Uint result = 0;
166     while (*s)
167     {
168         s += CharLengthUTF8(s);
169         result++;
170     }
171     return result;
172 }
```

4.15 lib/screen/src/real/genericscreen.c++

```
 1 #include <iostream>
 2 #include<wchar.h>
 3 #include"screen.h++"
 4 #include"screenbuffer.h++"
 5 #include"genericscreen.h++"
 6 #include"throw.h++"
 7
 8 using namespace Scr;
 9 using namespace std;
10
11 template<typename _char_type>
12 Uint GenericScreen::PrecomputeTextCharsWidth(_char_type * text, vector<
       char>&
13     widths, Uint maxwidth)
14     throw(RangeError, IllegalCharacter)
15 {
16     Uint sum=0;
17     do
18     {
19         register Uint w = width(*text);
20         widths.push_back(w);
21         sum+=w;
22         text++;
23     }
24     while ( ( *text != 0 ) and (sum<maxwidth) );
25         //break loop if sum is equal or greater than maxwidth. it it is
               equal,
26         //and *text!=0, than surely in next pass of loop it will be
               greater
27
28     if (*text!=0)//loop broken, but not whole string calculated
29         THROW(RangeError);
30     return sum;
```

```
31 }
32
33 namespace Scr
34 {
35 //!local template specialization: adds UTF8 Decoding
36     template<>
37     Uint GenericScreen::PrecomputeTextCharsWidth (const char * text,
38             vector<char>& widths, Uint maxwidth)
39     throw(RangeError, IllegalCharacter)
40     {
41         Uint sum=0;
42         do
43         {
44             register Uint w = width (DecodeUTF8 (&text));
45             widths.push_back (w);
46             sum+=w;
47             text++;
48         }
49         while ( ( *text != 0 ) and (sum<maxwidth) );
50         //break loop if sum is equal or greater than maxwidth. it it is
               equal,
51         //and *text!=0, than surely in next pass of loop it will be
               greater
52
53         if (*text!=0)//loop broken, but not whole string calculated
54             THROW (RangeError);
55         return sum;
56     }
57
58 }
59
60 Scr::Key Scr::GenericScreen::DecodeKeyPressed()
61         throw(Connection::UnsupportedKey,Screen::InvalidUTF8)
62 {
63     Uint c = input.Get();
64
65     if (c==Key::LF)
66         return Key::Enter;
67
68     if (c==127)
69         return Key::Backspace;
70     if (c==8)
71         return Key::Backspace;
72
73     if (c>=' ')
74     {
75         input.UnGet();
76         return DecodeBasicKeyPressed();
77     }
78
79     if (input.HasBufferedText())
80         THROW(Scr::Connection::UnsupportedKey);
81     else if (c==0x1b)
82         return Key::Escape;
83     else
84         THROW(Scr::Connection::UnsupportedKey);
85 }
86
87 //simple macros for enabling and disabling utf-8 validation
88 #ifdef DO_VALIDATE_UTF_8_OUTPUT
89 # define VALIDATE_TRAILING(x)             \
90     if ( (c[x]&0xC0)!=0x80)               \
```

```
 91         THROW(Screen::InvalidTrailingByte)
 92 #else
 93 # define VALIDATE_TRAILING(x)
 94 #endif //DO_VALIDATE_UTF_8_OUTPUT
 95
 96 Scr::Key Scr::GenericScreen::DecodeBasicKeyPressed()throw(Screen::
       InvalidUTF8)
 97 {
 98
 99     Uint result;
100     unsigned char c[4];
101     c[0]=input.Get();
102
103     if ( (c[0] & 0xF8) == 0xF0) // 4 byte char
104     {
105
106         c[1]=input.Get();
107         VALIDATE_TRAILING(1); // only some values for trailing bytes
108         // are valid
109
110         c[2]=input.Get();
111         VALIDATE_TRAILING(2);
112
113         c[3]=input.Get();
114         VALIDATE_TRAILING(3);
115         c[0]&=0x0F;
116         c[1]&=0x7F;
117         c[2]&=0x7F;
118         c[3]&=0x7F;
119         result = c[3]+(((Uint)c[2])<<6)+(((Uint)c[1])<<12)
120             +(((Uint)c[0])<<18);
121
122         if (result < (1<<16))// overlong UTF8 encoding FORBIDDEN
123             // according to RFC 3629
124         {
125             THROW(Screen::OverlongUTF8Encoding);
126         }
127     }
128     else if ((c[0] & 0xF0) == 0xE0) // 3 byte char
129     {
130
131         c[1]=input.Get();
132         VALIDATE_TRAILING(1);
133
134         c[2]=input.Get();
135         VALIDATE_TRAILING(2);
136         c[0]&=0x1F;
137         c[1]&=0x7F;
138         c[2]&=0x7F;
139         result = c[2]+(((Uint)c[1])<<6)+(((Uint)c[0])<<12);
140
141         if (result < (1<<11))// overlong UTF8 encoding FORBIDDEN
142             // according to RFC 3629
143         {
144             THROW(Screen::OverlongUTF8Encoding);
145         }
146     }
147     else if ((c[0] & 0xE0) == 0xC0) // 2 byte char
148     {
149
150         c[1]=input.Get();
151         VALIDATE_TRAILING(1);
```

```
152
153         c[0]&=0x3F;
154         c[1]&=0x7F;
155
156         result = c[1]+(((Uint)c[0])<<6);
157         if (result < (1<<7))// overlong UTF8 encoding FORBIDDEN
158             // according to RFC 3629
159         {
160             THROW(Screen::OverlongUTF8Encoding);
161         }
162     }
163     else
164         if ((c[0] & 0x80) == 0)
165             // 1 byte char
166         {
167             result=c[0];
168         }
169         else
170             THROW(Screen::InvalidFirstByte);
171
172     return result;
173 }
174
175 /*!
176   Print block of text from specific source: while condition is true
177   add chr from source and perform additional action "finish"
178  */
179 #define PRINT_TEXT(condition,source,finish)                              \
180     bool fgt (properties.GetFgColor() ==Fg::Transparent);                \
181     bool bgt (properties.GetBgColor() ==Bg::Transparent);                \
182                                                                          \
183     if (fgt && bgt )/*both: background and foreground are transparent*/ \
184         while (condition)                                                \
185         {                                                                \
186             controlBuffer[aPoint.row][aPoint.col].style.SetFgStyle(
                    properties.GetFgStyle()); \
187             controlBuffer[aPoint.row][aPoint.col].c=source;              \
188             finish;                                                      \
189         }                                                                \
190     else if (fgt)/* foreground is transparent, but background isn't */  \
191         while (condition)                                                \
192         {                                                                \
193             controlBuffer[aPoint.row][aPoint.col].style.SetFgStyle(
                    properties.GetFgStyle()); \
194             controlBuffer[aPoint.row][aPoint.col].style.SetBgColor(
                    properties.GetBgColor()); \
195             controlBuffer[aPoint.row][aPoint.col].c=source;              \
196             finish;                                                      \
197         }                                                                \
198     else if (bgt)                                                        \
199         while (condition)                                                \
200         {                                                                \
201             controlBuffer[aPoint.row][aPoint.col].style.SetFgColor(
                    properties.GetFgColor()); \
202             controlBuffer[aPoint.row][aPoint.col].style.SetFgStyle(
                    properties.GetFgStyle()); \
203             controlBuffer[aPoint.row][aPoint.col].c=source;              \
204             finish;                                                      \
205         }                                                                \
206     else /*niether background, nor foreground is transparent */         \
207         while (condition)                                                \
208         {                                                                \
```

```
209             controlBuffer[aPoint.row][aPoint.col].style=properties;     \
210             controlBuffer[aPoint.row][aPoint.col].c=source;             \
211             finish;                                                     \
212         }
213
214 // end of macro PRINT_TEXT
215
216 // FOLLOWING MACRO HAS TO BE REWRITTEN IN TERMS OF widths VECTOR for
217 // improved efficiency (use precomputed widths)
218
219 /*! To achieve UNICODE compliance CJK must be supported - this macro
220  *  performs additional configuration after adding character, that
221  *  may be CJK*/
222 #define ADDWCHAR_BASE(op_1,op_CJK,w)                                    \
223     if (w==1)                                                           \
224     {                                                                   \
225         /*to balance column width w/ CJK*/                              \
226         if (aPoint.col != 0 &&                                          \
227             width(controlBuffer[aPoint.row][aPoint.col-1].c)==2)        \
228             controlBuffer[aPoint.row][aPoint.col-1].c=' ';              \
229         if (aPoint.col+1 <controlBuffer.GetWidth() )                    \
230         {                                                               \
231             if (controlBuffer[aPoint.row][aPoint.col+1].c==0)           \
232                 controlBuffer[aPoint.row][aPoint.col+1].c=' ';          \
233         }                                                               \
234         op_1;                                                           \
235     }                                                                   \
236     else if (w==2)                                                      \
237     {   /*Previous column CAN'T be CJK, as theese characters MUST be*/  \
238         /* separated by NULL character to balance width*/              \
239         /*  */                                                         \
240         if (aPoint.col != 0 &&                                          \
241             width(controlBuffer[aPoint.row][aPoint.col-1].c)==2)        \
242             controlBuffer[aPoint.row][aPoint.col-1].c=' ';              \
243         /* Fill subsequent character w/ 0*/                            \
244         if (aPoint.col+1 <controlBuffer.GetWidth() )                    \
245         {                                                               \
246             if (aPoint.col+2 <controlBuffer.GetWidth() &&               \
247                 width(controlBuffer[aPoint.row][aPoint.col+1].c)==2)    \
248             {                                                           \
249                 /* subsequent is CJK, so next is 0. fill it w/ space */ \
250                 /* to engorce correct layout of text during refresh  */ \
251                 controlBuffer[aPoint.row][aPoint.col+2].c=' ';          \
252                 controlBuffer[aPoint.row][aPoint.col+2].style=          \
253                     controlBuffer[aPoint.row][aPoint.col].style;        \
254             }                                                           \
255             controlBuffer[aPoint.row][aPoint.col+1].c=0;                \
256             controlBuffer[aPoint.row][aPoint.col+1].style=              \
257                 controlBuffer[aPoint.row][aPoint.col].style;            \
258         }                                                               \
259         op_CJK;                                                         \
260     }                                                                   \
261     else                                                                \
262         ;
263
264 //end of macro ADDWCHAR_BASE
265
266 /*!
267  * Add wide character. doo width lookup for character using C function
268  */
269 #define ADDWCHAR_DEFAULT(op_1,op_CJK)                                   \
270     int w = width(controlBuffer[aPoint.row][aPoint.col].c);             \
```

```
271     ADDWCHAR_BASE(op_1,op_CJK,w)
272
273 /*!
274  * Add character. Check width in lookup table.
275  */
276 #define ADDWCHAR_PRECOMPUTED_DEFAULT(op_1,op_CJK)\
277     ;ADDWCHAR_BASE(op_1,op_CJK,widths[i]);i++;
278
279 // macro for default wide char adding
280 #define ADDWCHAR ADDWCHAR_DEFAULT(aPoint.col++,aPoint.col+=2)
281
282 #define ADDWCHAR_PRECOMPUTED \
283     ADDWCHAR_PRECOMPUTED_DEFAULT(aPoint.col++,aPoint.col+=2)
284
285 // Constructor initializes base objects
286 GenericScreen::GenericScreen(std::istream & _input,std::ostream & _output)
        throw()
287     : ScreenBase(),
288       controlBuffer(25,80),// will be changed before object used
289                            // (now actual dimensions are unknown, and
                                    therefore
290                            //  25x80 is as good as 12x13 or 120x430)
291
292       cursorPosition(0,0),
293       cursorFlags(cursorVisible),
294       input(_input),
295       output(_output)
296     {;}
297
298 void GenericScreen::Clear()throw()
299 {
300     controlBuffer.Fill(ScreenCharacter(' ', properties));
301 }
302
303 void GenericScreen::SetBgColor(Bg::Color col)throw()
304 {
305     properties.SetBgColor(col);
306 }
307
308 void GenericScreen::SetFgColor(Fg::Color col)throw()
309 {
310     properties.SetFgColor(col);
311 }
312
313 void GenericScreen::SetFgStyle(Fg::Style s)throw()
314 {
315     properties.SetFgStyle(s);
316 }
317 void GenericScreen::GotoYX(Uint y, Uint x)
318         throw(GotoOutOfRange)
319 {
320     if (y>=GetHeight() || x>=GetWidth())
321         THROW(GotoOutOfRange);
322     aPoint.row = y;
323     aPoint.col = x;
324 }
325
326 void GenericScreen::AddText(const char * text)
327     throw(PrintOutOfRange,
328       IllegalCharacter)
329 {
330     vector<char> widths;
```

```
331     widths.reserve(controlBuffer.GetWidth());
332     try
333     {
334         AddText(text,
335             // number of columns needed for specific text (if throws
                    exception,
336             //      AddText(const char *, Uint) is not executed)
337             PrecomputeTextCharsWidth(text,widths,controlBuffer.GetWidth())
                    ,
338                                      widths);
339     }
340     catch (RangeError & e)
341     {
342         throw(PrintOutOfHorizontalRange(string(e.what())+__WHERE_AM_I__));
343     }
344 }
345
346 void GenericScreen::AddText(const std::string & text)
347     throw(PrintOutOfRange,
348           IllegalCharacter)
349 {
350     AddText(text.c_str());
351 }
352
353
354
355 void GenericScreen::AddText(const char * text, Uint cols,
356         const vector<char> &widths)
357         throw(PrintOutOfRange, IllegalCharacter)
358 {
359     if (cols> controlBuffer.GetWidth()-aPoint.col)
360         THROW(PrintOutOfHorizontalRange);
361     if (aPoint.row>=GetHeight())
362         THROW(PrintOutOfVerticalRange);
363     size_t i=0;
364     PRINT_TEXT(*text,DecodeUTF8(&text),text++;ADDWCHAR_PRECOMPUTED);
365 }
366
367 void GenericScreen::AddText(const std::wstring & text)
368     throw(PrintOutOfRange,
369           IllegalCharacter)
370 {
371     AddText(text.c_str());
372 }
373
374 #define DECLARE_WIDTHS_AND_COLS(m)
          \
375     vector<char> widths;
              \
376     widths.reserve(controlBuffer.GetWidth());
              \
377     Uint cols;
              \
378     try
              \
379     {
              \
380         cols=PrecomputeTextCharsWidth(text,widths,m);
                  \
381     }
              \
```

```
382     catch (RangeError & e)
                \
383     {
                \
384         throw(PrintOutOfHorizontalRange(string(e.what())
                    \
385                                 +"\n " __WHERE_AM_I__));
                                      \
386     }
                \
387
388 void GenericScreen::AddText(const wchar_t * text)
389     throw(PrintOutOfRange, IllegalCharacter)
390 {
391     DECLARE_WIDTHS_AND_COLS(controlBuffer.GetWidth());
392     if (cols> controlBuffer.GetWidth()-aPoint.col)
393         THROW(PrintOutOfHorizontalRange);
394     if (aPoint.row>=GetHeight())
395         THROW(PrintOutOfVerticalRange);
396
397     PRINT_TEXT(*text,*text,text++;ADDWCHAR);
398 }
399
400 //adding specified text, at most limitcols columns
401 Uint GenericScreen::AddTextCols(const wchar_t * text, Uint limitcols)
402     throw(PrintOutOfRange, IllegalCharacter)
403 {
404     vector<char> widths;
405     widths.reserve(controlBuffer.GetWidth());
406     Uint cols=0;
407     Uint i = 0;
408     while ( ( text[i] != 0 ) and (cols<=limitcols) )
409     {
410         const register Uint w = width (text[i++]);
411         widths.push_back (w);
412         cols+=w;
413     }
414     cols=min(cols,limitcols);
415
416     if (cols> controlBuffer.GetWidth()-aPoint.col)
417         THROW(PrintOutOfHorizontalRange);
418
419     if (aPoint.row>=GetHeight())
420         THROW(PrintOutOfVerticalRange);
421
422     Sint _i = limitcols;
423     i=0;
424
425     PRINT_TEXT((text[i]) && ((_i-=widths[i])>=0),text[i],
426                ADDWCHAR_PRECOMPUTED);
427
428
429     _i+=width(*(text));
430
431     return limitcols - _i;
432 }
433
434 Uint GenericScreen::AddTextCols(const std::wstring& text, Uint limitcols)
435     throw(PrintOutOfRange,
436           IllegalCharacter)
437 {
438     return AddTextCols(text.c_str(), limitcols);
```

```
439 }
440
441 void GenericScreen::AddSubscreenText(const char * text, Uint widthlimit)
442     throw(PrintOutOfRange, IllegalCharacter)
443 {
444     vector<char> widths;
445     widths.reserve(controlBuffer.GetWidth());
446     try
447     {
448         AddText(text,
449             // number of columns needed for specific text (if throws
                    exception,
450             //      AddText(const char *, Uint) is not executed)
451             PrecomputeTextCharsWidth(text,widths,widthlimit), widths);
452     }
453     catch (RangeError & e)
454     {
455         throw(PrintOutOfHorizontalRange(string(e.what())+"\n "
                __WHERE_AM_I__));
456     }
457 }
458
459 void GenericScreen::AddSubscreenText(const wchar_t * text, Uint widthlimit
        )
460     throw(PrintOutOfRange, IllegalCharacter)
461 {
462     vector<char> widths;
463     widths.reserve(controlBuffer.GetWidth());
464     Uint cols;
465     try
466     {
467         cols=PrecomputeTextCharsWidth(text,widths,widthlimit);
468     }
469     catch (RangeError & e)
470     {
471         throw(PrintOutOfHorizontalRange(string(e.what())+"\n "
                __WHERE_AM_I__));
472     }
473
474     if (cols> controlBuffer.GetWidth()-aPoint.col)
475         THROW(PrintOutOfHorizontalRange);
476     if (aPoint.row>=GetHeight())
477         THROW(PrintOutOfVerticalRange);
478
479     int i = 0;// variable used by ADDWCHAR_PRECOMPUTED macro
480     PRINT_TEXT(text[i],text[i],ADDWCHAR_PRECOMPUTED);
481 //      PRINT_TEXT(*text,*text,text++;ADDWCHAR_PRECOMPUTED);
482 }
483
484 void GenericScreen::HorizontalLine(char c, Uint n)
485     throw(PrintOutOfRange,
486           IllegalCharacter)
487 {
488     HorizontalLine(static_cast<wchar_t>(c),n);
489 }
490
491 void GenericScreen::HorizontalLine(wchar_t c, Uint n)
492     throw(PrintOutOfRange,
493           IllegalCharacter)
494 {
495     if (n> controlBuffer.GetWidth()-aPoint.col+1)
496         THROW(PrintOutOfHorizontalRange);
```

```
497     if (aPoint.row>=GetHeight())
498         THROW(PrintOutOfVerticalRange);
499     PRINT_TEXT(n--,c,ADDWCHAR);
500 }
501
502 void GenericScreen::VerticalLine(char c, Uint n)
503     throw(PrintOutOfRange,
504           IllegalCharacter)
505 {
506     VerticalLine(static_cast<wchar_t>(c),n);
507 }
508
509 void GenericScreen::VerticalLine(wchar_t c, Uint n)
510     throw(PrintOutOfRange,
511           IllegalCharacter)
512 {
513     if (n> controlBuffer.GetHeight()-aPoint.row+1)
514         THROW(PrintOutOfHorizontalRange);
515     if (aPoint.col>=GetWidth())
516         THROW(PrintOutOfVerticalRange);
517     PRINT_TEXT(n--,c,ADDWCHAR_DEFAULT(aPoint.row++,aPoint.row++));
518 }
519
520 void GenericScreen::Rectangle(char c, const Size & s)
521     throw(PrintOutOfRange,
522           IllegalCharacter)
523 {
524     Uint n = s.height;
525     while (n--)
526     {
527         HorizontalLine(c,s.width);
528         aPoint.col-=s.width;
529         aPoint.row++;
530     }
531 }
532
533 void GenericScreen::Rectangle(wchar_t c, const Size & s)
534     throw(PrintOutOfRange,
535           IllegalCharacter)
536 {
537     Uint n = s.height;
538     while (n--)
539     {
540         HorizontalLine(c,s.width);
541         aPoint.col-=s.width*width(c);
542         aPoint.row++;
543     }
544 }
545
546 void GenericScreen::AddCharacter(char c)
547     throw(PrintOutOfRange)
548 {
549     if (aPoint.col>=GetWidth())
550         THROW(PrintOutOfHorizontalRange);
551     if (aPoint.row>=GetHeight())
552         THROW(PrintOutOfVerticalRange);
553
554     controlBuffer[aPoint.row][aPoint.col].c=c;
555     if (properties.GetFgColor() !=Fg::Transparent)
556         controlBuffer[aPoint.row][aPoint.col].style.SetFgColor(properties.
              GetFgColor());
```

```
557     controlBuffer[aPoint.row][aPoint.col].style.SetFgStyle(properties.
            GetFgStyle());
558     if (properties.GetBgColor() !=Bg::Transparent)
559         controlBuffer[aPoint.row][aPoint.col].style.SetBgColor(properties.
                GetBgColor());
560     ADDWCHAR;
561 }
562 void GenericScreen::AddCharacter(wchar_t c)
563     throw(PrintOutOfRange,
564           IllegalCharacter)
565 {
566     if (aPoint.col>=GetWidth())
567         THROW(PrintOutOfHorizontalRange);
568     if (aPoint.row>=GetHeight())
569         THROW(PrintOutOfVerticalRange);
570     if (c >= (1<<21))
571         THROW(CharacterExceedsUTF8Range);
572     controlBuffer[aPoint.row][aPoint.col].c=c;
573
574     if (properties.GetFgColor() !=Fg::Transparent)
575         controlBuffer[aPoint.row][aPoint.col].style.SetFgColor(properties.
                GetFgColor());
576     controlBuffer[aPoint.row][aPoint.col].style.SetFgStyle(properties.
            GetFgStyle());
577     if (properties.GetBgColor() !=Bg::Transparent)
578         controlBuffer[aPoint.row][aPoint.col].style.SetBgColor(properties.
                GetBgColor());
579     ADDWCHAR;
580 }
581
582 void GenericScreen::ForceCursorPosition(Position p )throw(RangeError)
583 {
584     if (p.row>=GetHeight() || p.col>=GetWidth())
585         THROW(RangeError);
586
587     cursorFlags|=cursorForced;
588     cursorPosition=p;
589 }
590
591
592 void GenericScreen::HideCursor()throw(CursorVisibilityNotSupported)
593 {
594     cursorFlags &=~ cursorVisible;
595 }
596
597 void GenericScreen::ShowCursor()throw(CursorVisibilityNotSupported)
598 {
599     cursorFlags |=  cursorVisible;
600 }
601
602 void GenericScreen::Refresh() // just a dumb proc to produce
603 throw(ConnectionError)    // basic debug printout
604 {
605     for (Uint i=0;i<controlBuffer.GetHeight();i++)//for each row
606     {
607         for (Uint j=0;j<controlBuffer.GetWidth();j++)
608         {// print raw character if printable low ascii
609             unsigned char c = controlBuffer[i][j].c;
610             output << (char)(((c>31)&&(c<128))?c:'.');
611             // or leave dot otherwise
612         }
613         output << endl;
```

```
614         // flush each row
615     }
616 }
617
618 Screen * GenericScreen::
619 CreateSubScreen(Uint _y_offset, Uint _x_offset, Uint _h,
620                 Uint _w)throw(SubscreenOutOfRange)
621 {
622     SubScreenRangeCheck();
623     // if no exceptional conditions, just create and return new subscreen
624     return new SubScreen(*this, _y_offset, _x_offset, _h, _w);
625 }
626
627 void GenericScreen::Resize(Uint rows, Uint cols)
628         throw()
629 {
630     controlBuffer.Resize(rows,cols);
631 }
632
633 const char * Scr::GenericScreen::GetType() const throw(TerminalTypeUnknown
      )
634 {
635     THROW (TerminalTypeUnknown);
636     // this implementation does not support type.
637 }
638
639 Uint GenericScreen::GetHeight() const throw()
640 {
641     return controlBuffer.GetHeight();
642 }
643
644 Uint GenericScreen::GetWidth() const throw()
645 {
646     return controlBuffer.GetWidth();
647 }
648
649 bool GenericScreen::GetCursorVisibility() const throw()
650 {
651     return cursorFlags bitand cursorVisible;
652 }
653
654 void GenericScreen::CleanUp() throw(ConnectionError)
655 {
656     ;
657 }
658
659 GenericScreen::~GenericScreen()throw()
660 {
661     ;
662 }
```

4.16 lib/screen/src/real/localscreen.c++

```
1 #include <sys/time.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 #include <sys/ioctl.h>
```

```
 5 #include <termios.h>
 6 #include <iostream>
 7 #include <signal.h>
 8 #include"screen.h++"
 9 #include"screenbuffer.h++"
10 #include"genericscreen.h++"
11 #include"localscreen.h++"
12 #include <stdlib.h>
13 using namespace std;
14 #include <boost/thread/mutex.hpp>
15
16 namespace {
17     boost::mutex M;
18     Scr::LocalScreen * ls=0;
19     sighandler_t osh=0;
20     void sh(int i)
21     {
22         ls->TestForResize();
23     }
24
25 }
26
27
28
29 Scr::LocalScreen::LocalScreen(Connection & _connection,
30                               std::istream & _input,
31                               std::ostream & _output)throw()
32     :GenericScreen(_input,_output),__ScreenConnection(_connection,_input)
33 {
34
35     tcgetattr(fileno_hack(_output), &term);
36     term.c_cc[VMIN] = 1;
37     term.c_lflag &= ~(ECHO | ICANON); // disable echo on terminal
38     tcsetattr(fileno_hack(_output), 0, &term);
39     _input.sync_with_stdio(false);
40     _output.sync_with_stdio(false);
41     /*!
42       please note, that, turning sync. off for cin  may be detected as
              memory
43       leak by valgrind debugger. According to GNU folks thic behaviour
44       is normal (since desynchronizing means allocating special memory
45       block, which is never freed as standard streams are never deleted)
46       http://gcc.gnu.org/ml/gcc-bugs/2006-06/msg00824.html
47      */
48     RexIOLog(LogLevelModerate)<<"LocalScreen(std::ostream & _output)"<<
          endl;
49     boost::mutex::scoped_lock Lock(M);
50
51     if (!ls)
52     {
53         ls=this;
54 #ifdef SIGWINCH
55         osh=signal(SIGWINCH,sh);
56 #endif
57     }
58 }
59
60 /*function copied from original telnet client*/
61 void Scr::LocalScreen::TestForResize() {
62     boost::mutex::scoped_lock Lock(M);
63     int infd=input.FD();
64     struct winsize ws;
```

```
 65     bool result;
 66     if (ioctl(infd, TIOCGWINSZ, (char *)&ws) >= 0) {
 67         if (GetHeight()==ws.ws_row && GetWidth()==ws.ws_col)
 68             result= false;
 69         Resize(ws.ws_row,ws.ws_col);
 70         result= true;
 71     }
 72 
 73     result= false;
 74     connection.OnResize(GetHeight(),GetWidth());
 75 }
 76 
 77 const char * Scr::LocalScreen::GetType() const throw()
 78 {
 79     return getenv("TERM");
 80 }
 81 
 82 int Scr::LocalScreen::ProcessConnection()
 83 {
 84     connection.OnStart();
 85     active=true;
 86     char counter = 0;
 87     while (input.Stream().good() && active)
 88     {
 89         usleep(1);
 90         if (!counter) // test at most in every 255 000 useconds
 91             TestForResize(); // (CPU savings without
 92         //  usability loss)
 93         // - this is required, when system does not provide SIGWINCH
 94 
 95 //      boost::mutex::scoped_lock Lock(M);
 96         // returns true if we have anything to read
 97         if (input.KbHit())
 98         {
 99             input.Buffer();
100             do
101             {
102                 try
103                 {
104                     Key k = DecodeKeyPressed();
105                     if (k==0x1b)
106                     {
107                         input.UnGet();
108                         k = DecodeKeyPressed();
109                     }
110                     connection.OnKeyDown(k);
111                 }
112                 catch (Scr::Connection::UnsupportedKey)
113                 {
114                     while (input.HasBufferedText())
115                         input.Get();
116                 }
117             }
118             while (input.HasBufferedText());
119         }
120         counter++;
121     }
122     CleanUp();
123     return exitcode; // OnExit is called by Connection::Start()
124 }
125 
126 Scr::LocalScreen::~LocalScreen()throw()
```

```
127 {
128     term.c_lflag |= ECHO | ICANON; // reenable echo on terminal
129     tcsetattr(fileno_hack(output), 0, &term);
130     RexIOLog(LogLevelModerate)<<"~LocalScreen()"<<endl;
131 }
```

4.17 lib/screen/src/real/remotescreen.c++

```
  1 #include"fileno_hack.h++"
  2 #include"screen.h++"
  3 #include"screenbuffer.h++"
  4 #include"genericscreen.h++"
  5 #include"remotescreen.h++"
  6 #include"telnet.h++"
  7 #include"throw.h++"
  8 #include <iostream>
  9 #include <queue>
 10 #include <cassert>
 11 #include <sys/select.h>
 12 Scr::RemoteScreen::RemoteScreen(Connection & _connection,
 13                                 std::istream & _input,
 14                                 std::ostream & _output)throw()
 15     :GenericScreen(_input,_output),__ScreenConnection(_connection,_input),
 16      requestedSize(25,80),resizeRequestPending(false),counter(0),
 17      telnetSettings(0)
 18 {
 19     using namespace std;
 20     using namespace TELNET;
 21 
 22     clientname.reserve(64);
 23 
 24     //following two lines enable char-by-char mode for all
 25     //standard-compliant telnet clients by forcing them to disable
 26     //local echo and not waiting for GA (therefore each character
 27     //typed by user will be delivered to server as soon as possible,
 28     //and displayed only if server will allow it to)
 29     output << IAC << WILL << ECHO
 30            << IAC << WILL << SGA
 31 
 32     //Request TTYPE information. In this application specific case,
 33     //client MUST answer IAC WILL TTYPE. Refusal or ignoring this
 34     //request will result in considering client dumb terminal and
 35     //breaking connection, as no graphic advanced capabilities
 36     //provided disable usage of screen interfaces
 37            << IAC << DO << TTYPE//< enable TTYPE mode
 38            << IAC << SB << TTYPE << SEND
 39            << IAC << SE // < request TTYPE info RIGHT NOW
 40 
 41     //suggest negotiation about window size
 42     //client must answet  IAC WILL NAWS, and then supply proper number
 43     //to use this feature: otherwise default 24 row / 80column  will
 44     //be assumed (refer to GenericScreen constructor)
 45            << IAC << DO << NAWS << flush;
 46 
 47     //diagnostics
 48     RexIOLog(LogLevelModerate)<<"RemoteScreen(std::ostream & _output)"<<
        endl;
```

```
 49 }
 50
 51 Scr::Key Scr::RemoteScreen::DecodeKeyPressedHandleTelnet()
 52 {
 53     Uint result;
 54     try
 55     {
 56         result = DecodeKeyPressed();
 57     }
 58     catch(Scr::Connection::UnsupportedKey)
 59     {
 60         while (input.HasBufferedText())
 61             result=input.Get();
 62     }
 63     if (result == 0xd ) // Carriage return - ignore LF
 64     {
 65         Uint i = input.Get(); // ignore next
 66         if (i == 0 or i == 0xa)
 67             return Key::Enter; // some TELNET clients don't really provide
                  line
 68                        // feed after CR.
 69         else
 70             THROW(LogicError);
 71     }
 72     return result;
 73 }
 74
 75 #define SUBNDESC(feature) case feature: RexIOLog(LogLevelModerate) << #
       feature "\n"; break
 76 void Scr::RemoteScreen::AnswerCommand()
 77 {
 78     using namespace TELNET;
 79     using namespace std;
 80     unsigned char c[2];
 81     c[0]=input.Get();
 82     assert(c[0]==IAC);
 83     c[0]=input.Get();
 84     c[1]=input.Get();
 85     RexIOLog(LogLevelModerate) << "Client says "<< static_cast<int>(c[0])
           <<" what means, it ";
 86     switch (c[0])
 87     {
 88     case WILL:
 89     case DO:
 90         RexIOLog(LogLevelModerate) << "agrees on (or requests) ";
 91         switch(c[1])
 92         {
 93             SUBNDESC(SGA);
 94             SUBNDESC(NAWS);
 95             SUBNDESC(TTYPE);
 96             SUBNDESC(ECHO);
 97             SUBNDESC(LINEMODE);
 98         default:
 99             RexIOLog(LogLevelModerate) << "unsupported feature\n";
100         }
101         break;
102     case WONT:
103     case DONT:
104         RexIOLog(LogLevelModerate) << "disagrees on (or requests not to
               use) ";
105         switch(c[1])
106         {
```

```
107             SUBNDESC(SGA);
108             SUBNDESC(NAWS);
109             SUBNDESC(TTYPE);
110             SUBNDESC(ECHO);
111             SUBNDESC(LINEMODE);
112         default:
113             RexIOLog(LogLevelModerate) << "unsupported feature\n";
114         }
115         break;
116     case SB:
117         RexIOLog(LogLevelModerate) << "wants to subnegotiate ";
118         switch(c[1])
119         {
120         case NAWS:
121             RexIOLog(LogLevelModerate) << "Window size\n";
122             SubnegotiateWindowSize();
123             break;
124         case TTYPE:
125             RexIOLog(LogLevelModerate) << "Terminal type\n";
126             SubnegotiateTerminalType();
127             break;
128         default:
129             THROW(Scr::Connection::IllegalTelnetAction);
130             // if what client says is agreement or disagreement, just
131             // ignore it., but subnegotiations are way too
132             // unpredictable: even waiting for SE would not be sufficient.
133             // threrefore: we have to support as much of them as possible,
134             // and throw exception on unsupported!
135         }
136         break;
137     default:
138         THROW(Scr::Connection::UnsupportedTelnetFeature);
139     }
140
141 }
142 #undef SUBNDESC
143
144 void Scr::RemoteScreen::SubnegotiateWindowSize()
145 {
146
147     using namespace TELNET;
148     using namespace std;
149     requestedSize.width = input.Get();
150     requestedSize.width <<=8;
151     requestedSize.width += input.Get();
152     requestedSize.height = input.Get();
153     requestedSize.height <<=8;
154     requestedSize.height += input.Get();
155     RexIOLog(LogLevelModerate) << "New height is "<<requestedSize.height
            << ", width is "<<requestedSize.width<<"\n";
156
157     unsigned char c;
158     c = input.Get();
159     if (c!= IAC)
160         THROW(Scr::Connection::IncorrectWindowSizeSubnegotiation);
161     c = input.Get();
162     if (c!= SE)
163         THROW(Scr::Connection::IncorrectWindowSizeSubnegotiation);
164
165     RexIOLog(LogLevelModerate) << "Subnegotiation correct - setting\n";
166
```

```
167     if (requestedSize.height!=GetHeight() || requestedSize.width!=GetWidth
            ())
168     {
169         if (!resizeRequestPending)
170         {
171             counter=-10;// ok! first request should be supported instantly
                    !
172             resizeRequestPending=true;
173         }
174         else
175             counter = 100;//app is requesting too frequent resize!
176         //wait at least 0.156 s for refresh to prevent byte-flood DoS.
177     }
178     telnetSettings|=windowSize;
179 }
180
181 const char * Scr::RemoteScreen::GetType() const throw(TerminalTypeUnknown)
182 {
183     EASSERT(!clientname.empty(),TerminalTypeUnknown);
184     return clientname.c_str();
185 }
186 static const std::size_t maxInputExcess = 1024;// yet another anti-DoS
        insurance
187 int Scr::RemoteScreen::ProcessConnection()
188 {
189     using namespace TELNET;
190
191     active=true;
192     // initialization block: first initialize TELNET session, then
193     // start parsing input (keyboard input is meaningless unless we
194     // know term type).
195     {
196         std::queue<char> toDecode;
197         while (input.Stream().good() and active)
198         {
199             input.Buffer();
200             while (input.HasBufferedText())
201             {
202                 int c = input.Peek();
203                 if (c == -1)
204                 {
205                     THROW(Scr::Connection::Broken);
206                 }
207                 if (c==IAC)
208                     AnswerCommand();
209                 else
210                     toDecode.push(input.Get());
211             }
212             if (// OK. setup is finished.
213                 (telnetSettings & (windowSize|terminalType) )
214                 == (windowSize|terminalType)
215                 )
216                 break;
217             if (toDecode.size()>maxInputExcess)
218                 THROWP(Scr::Connection::Broken,
219                         "Maximum input lag limit exceeeded");
220         }
221
222         connection.OnStart(); // now app is initialized and may
223                               // successfully be started.
224
```

```
225         if (resizeRequestPending) // request may have been sent during
                initialization
226         {
227             Resize(requestedSize.height,requestedSize.width);
228             connection.OnResize(GetHeight(),GetWidth());
229             resizeRequestPending=false;
230         }
231 
232         // if some chars other than TELNET negotiations were recieved,
233         // now is time to parse 'em
234         while  (!toDecode.empty())
235             connection.OnKeyDown(DecodeKeyPressedHandleTelnet());
236 
237     }// free stack resources allocated for initialization (in
238      // particular: queue toDecode, which now is not needed.
239 
240 
241 
242     int inFD=input.FD();
243     struct timeval t;
244     fd_set fds;// to allow instant Exit() without waiting for char
245                // application must handle asynchronous input
246                // processing correctly, what isn't supported by
247                // standart libstdc++ in linux.
248 
249     FD_ZERO(&fds);
250     while (input.Stream().good() and active)
251     {
252         FD_SET(inFD,&fds);
253         t.tv_sec=0;
254         t.tv_usec=1000;
255         // 1+onlyFD = maximum FD
256         if (select (1+inFD, &fds, 0, 0, &t))
257         {
258             if (static_cast<std::size_t>
259                     (input.Stream().rdbuf()->in_avail())
260                     >maxInputExcess)
261                 THROWP(Scr::Connection::Broken,
262                         "Maximum input lag limit exceeeded");
263             input.Buffer();
264             while (input.HasBufferedText())
265             {
266                 int c = input.Peek();
267                 if (c == -1)
268                 {
269                     THROW(Scr::Connection::Broken);
270                 }
271                 if (c==IAC)
272                     AnswerCommand();
273                 else
274                 {
275                     connection.OnKeyDown(DecodeKeyPressedHandleTelnet());
276                 }
277             }
278         }
279         if (!counter)
280         {// check every 0.256 s. if resize request is pending.
281             if (resizeRequestPending)
282             {
283                 Resize(requestedSize.height,requestedSize.width);
284                 connection.OnResize(GetHeight(),GetWidth());
285                 resizeRequestPending=false;
```

```
286             }
287         }
288         counter++; // as counter is one-byte, it is equivalent to code:
289                    // if (counter == 255)
290                    //     counter = 0;
291                    // else
292                    //     counter++;
293     }              //
294
295     EASSERT(!active,FatalException);
296
297     CleanUp();
298
299     return exitcode; // OnExit is called by Connection::Start().
300 }
301
302 //SUBNEGOTIATE TTYPE  (assume, IAC SB TTYPE  already recvd)
303 void Scr::RemoteScreen::SubnegotiateTerminalType()
304 {
305     using namespace TELNET;
306 //IAC SB TERMINAL-TYPE IS ... IAC SE
307     unsigned char c;
308     c=input.Get();
309     EASSERT(c==IS,Scr::Connection::IncorrectTerminalTypeSubnegotiation);
310
311     Uint i=0;
312     while (  (c=input.Get())!=IAC )
313     {
314         clientname+=tolower(c);
315         i++;
316         EASSERT(i<clientname.capacity(),
317                 Scr::Connection::IncorrectTerminalTypeSubnegotiation);
318
319     }
320     c=input.Get();
321     EASSERT(c==SE,Scr::Connection::IncorrectTerminalTypeSubnegotiation);
322     std::cout << "Client term type is: " << clientname << '\n';
323     telnetSettings|=terminalType;
324 }
325
326 Scr::RemoteScreen::~RemoteScreen()throw()
327 {
328     using namespace std;
329     RexIOLog(LogLevelModerate)<<"~RemoteScreen()"<<endl;
330 }
```

4.18 lib/screen/src/real/screenconnection.c++

```
1 #include "screen.h++"
2 #include "screenbuffer.h++"
3 #include "genericscreen.h++"
4 #include "throw.h++"
5 #include "keyboard.h++"
6 #include "screenconnection.h++"
7
8 Scr::__ScreenConnection::__ScreenConnection(Connection & _connection,std::
      istream & _input)
```

```
 9     throw()
10     :
11     exitcode(0),connection(_connection),active(false){;}
12
13 void Scr::__ScreenConnection::ExitConnection(int _code)
14 {
15     exitcode=_code;
16     active = false;
17 }
18
19 Scr::__ScreenConnection::~__ScreenConnection()throw(){;}
```

4.19 lib/screen/src/real/terminal.c++

```
 1
 2 #include <termios.h>
 3 #include <iostream>
 4 #include"screen.h++"
 5 #include"throw.h++"
 6 #include"screenbuffer.h++"
 7 #include"genericscreen.h++"
 8 #include"vt100compatible.h++"
 9 #include"vt100codes.h++"
10 #include"terminfoenabled.h++"
11 using namespace std;
12
13 Scr::Terminal::Terminal()throw()
14     :copyBuffer(25,80,ScreenCharacter(0,DisplayStyle(Fg::System,Fg::Dark,
           Bg::System)))
15 {;}
```

4.20 lib/screen/src/real/terminfoenabled.c++

```
 1
 2 #include <termios.h>
 3 #include <iostream>
 4 #include"screen.h++"
 5 #include"throw.h++"
 6 #include"screenbuffer.h++"
 7 #include"genericscreen.h++"
 8 #include"vt100compatible.h++"
 9 #include"vt100codes.h++"
10 #include"terminfoenabled.h++"
11 #include"terminfokeymap.h++"
12 using namespace std;
13
14 Scr::TerminfoEnabledScreen::TerminfoEnabledScreen(
15         std::istream & _input,std::ostream & _output)throw()
16     :GenericScreen(_input,_output),
17      Terminal(),ti(NULL),p(Scr::Fg::System,Scr::Fg::Dark,Scr::Bg::System)
18 {
19     RexIOLog(LogLevelModerate)
```

```
20          << "TerminfoEnabledScreen(std::ostream & _output)"<<endl;
21      p=copyBuffer[copyBuffer.GetHeight()-1]
22                  [copyBuffer.GetWidth()-1].style;
23  }
24  Scr::Key Scr::TerminfoEnabledScreen::DecodeKeyPressed()
25          throw(Connection::UnsupportedKey,Screen::InvalidUTF8)
26  {
27      try
28      {
29          return GenericScreen::DecodeKeyPressed();
30      }
31      catch(Scr::Connection::UnsupportedKey)
32      {// special keys are exceptional, so treat them like exceptionals
33          // implement terminfo-driven algorithm here
34          input.UnGet();
35          {
36              char c = input.Peek();
37              if (c!=0x1b and c<' ')
38              {
39                  RexIOLog(LogLevelModerate) << "Processing input " <<
                        static_cast<int>(c)<< endl;
40                  c=input.Get();
41                  if (c==9)
42                      return Key(Key::Tab);
43                  else
44                      return Key(c);
45              }
46          }
47          std::string code;
48          code.push_back(input.Get());
49          while (true)
50          {
51              using namespace TI;
52              RexIOLog(LogLevelModerate) << "Processing input " << code <<
                    endl;
53              Keymap::validity v = ti->GetKeymap().TestCode(code.c_str());
54              if (v == Dictionary<Key>::iterator::VALID)
55                  return ti->GetKeymap().GetCode(code.c_str());
56              else
57              {
58                  try
59                  {
60                      if( not input.HasBufferedText() )
61                      {
62                          return ti->GetKeymap().GetCode(code.c_str());
63                      }
64                  }
65                  catch(...)
66                  {
67                      throw;
68                  }
69                  code.push_back(input.Get());
70                  continue;
71              }
72          }
73          THROW(LogicError);
74      }
75      catch(...)
76      {
77          throw;
78      }
79  }
```

```
 80 void Scr::TerminfoEnabledScreen::Refresh()throw(ConnectionError)
 81 {
 82     if (!ti)
 83     {
 84         try
 85         {
 86         ti= const_cast<Scr::TI::TerminfoEntry*>
 87             (& (TI::TerminfoCore::GetTerminfo(GetType()) ));
 88         }
 89         catch (Scr::TI::NotSupportedTerminalType)
 90         {
 91             THROW(TerminalTypeUnknown);
 92         }
 93         // test if all required requests are supported
 94         try
 95         {
 96             ti->CursorHome();
 97             ti->GotoYX(Position(7,7));// any coordinate
 98         }
 99         catch (Scr::TI::TerminfoEntry::CapabilityNotSupported)
100         {
101             THROW(TerminalTypeUnknown);
102         }
103         catch (...)
104         {
105             THROW(LogicError);
106         }
107     }
108
109     try
110     {
111         output << ti->HideCursor();
112     }
113     catch(...){;}
114     output << ti->CursorHome();
115     Uint I=0,
116     J=0;
117
118 //                      // (but don't do it when only foreground of
119 //                      // space changed
120 //                      and not (controlBuffer[i][j].c == ' '
121 //                               and copyBuffer[i][j].c == ' '
122 //                               and controlBuffer[i][j].style.GetBgColor
      ()
123 //                               == p.GetBgColor())
124     for (Uint i=0;i<controlBuffer.GetHeight();i++)//for each row
125         for (Uint j=0;j<controlBuffer.GetWidth();j++)// for each col
126         {
127             if (controlBuffer[i][j].c != 0)
128             {
129                 if (controlBuffer[i][j]!=copyBuffer[i][j])
130                 {
131                     if (i!=I || j!=J)
132                     {
133                         output<< ti->GotoYX(Position(i,j));
134                         I=i;
135                         J=j;
136                     }
137                     if (// If writing style have changed, update it
138                         p!= controlBuffer[i][j].style)
139                     {
140                         output << ti->SetDisplayStyle(
```

```
141                             controlBuffer[i][j].style,p);
142                         p= controlBuffer[i][j].style;
143                     }
144
145                     // print character itself only if it is printable
146                     if (controlBuffer[i][j].c == 0x7f)
147                         EncodeUTF8(output,0x2421);// D E L
148                     else if (controlBuffer[i][j].c >= ' ')
149                         EncodeUTF8(output,controlBuffer[i][j].c);
150                     else
151                         EncodeUTF8(output,0x2400+controlBuffer[i][j].c);
152                     // Unicode characters representing teletype mnemonics
                            for
153                     // first 31 ASCII characters
154
155                     J++;
156                 }
157             }// controlBuffer[i][j].c == 0  -- do not print anything
158             else// but mention, that last character was 2-column CJK.
159             {
160                 if (j!=0 && j==(J+1))
161                     J++;
162             }
163
164         }
165     // display cursor if requested
166     if (cursorFlags&cursorVisible)
167     {
168         if (cursorFlags&cursorForced)
169             output<< ti->GotoYX(cursorPosition);
170         try
171         {
172         output << ti->ShowCursor();
173         }
174         catch(...){;}
175     }
176
177     output<<flush;
178     if (!output.good())
179         THROW(ConnectionError);
180     copyBuffer=controlBuffer;
181
182 }
183
184 void Scr::TerminfoEnabledScreen::Resize(Uint rows, Uint cols)
185     throw()
186 {
187     GenericScreen::Resize(rows,cols);
188     copyBuffer.Resize(controlBuffer.GetHeight(),
189               controlBuffer.GetWidth());
190     copyBuffer.Fill(ScreenCharacter(0,DisplayStyle()));
191 }
192
193 void Scr::TerminfoEnabledScreen::CleanUp()  throw(ConnectionError)
194 {
195     output << SHOW_CURSOR;
196     output<<SET_ATTR(ATTR_RESET)<<ERASE_SCREEN  << CURSOR_HOME;
197 }
198
199 Scr::TerminfoEnabledScreen::~TerminfoEnabledScreen()throw()
200 {
201
```

```
202     if (ti)
203         TI::TerminfoCore::FreeTerminfoEntry();
204     RexIOLog(LogLevelModerate)<<"~TerminfoEnabledScreen()"<<endl;
205 }
```

4.21 lib/screen/src/real/vt100compatible.c++

```
 1
 2 #include <termios.h>
 3 #include <iostream>
 4 #include"screen.h++"
 5 #include"throw.h++"
 6 #include"screenbuffer.h++"
 7 #include"genericscreen.h++"
 8 #include"vt100compatible.h++"
 9 #include"vt100codes.h++"
10 using namespace std;
11
12 Scr::VT100Compatible::VT100Compatible(std::istream & _input,
13         std::ostream & _output)throw()
14     :GenericScreen(_input,_output),
15      Terminal()
16 {
17     RexIOLog(LogLevelModerate) << "VT100Compatible(std::ostream & _output)
           "<<endl;
18 }
19
20 Scr::Key Scr::VT100Compatible::DecodeKeyPressed()
21         throw(Connection::UnsupportedKey,Screen::InvalidUTF8)
22 {
23     std::string DebugInfo(input.DebugInfo());
24     Uint c = input.Get();
25
26     if (c==Key::LF)
27         return Key::Enter;
28
29     if (c==127)
30         return Key::Backspace;
31     if (c==8)
32         return Key::Backspace;
33
34     if (c!= Key::Escape)
35     {
36         input.UnGet();
37         return DecodeBasicKeyPressed();// go back and process UTF-8 input
38     }
39     try
40     {
41         c = input.TryGet();
42     }
43     catch (Scr::BufferedInput::BufferEmpty)
44     {
45         return Key::Escape;
46     }
47
48     if (c == 'O')  // capital letter O, not digit 0
49         // <ESC>OP,<ESC>OQ,<ESC>OR,<ESC>OS = F1..F4
```

```
50      {
51          c=input.Get();
52          if (c<'P' || c>'S')
53              THROWP(Scr::Connection::UnsupportedKey,DebugInfo);
54          return Key::F1+ c-'P';
55      }
56
57      if (c == '[')
58      {
59          switch (input.Get()) // key after [
60          {
61          case 'A':
62              return Key(Key::Up);
63          case 'B':
64              return Key(Key::Down);
65          case 'C':
66              return Key(Key::Right);
67          case 'D':
68              return Key(Key::Left);
69          case '1': // 1,2,3,4,5, 7,8,9 =  F1..F8
70              c=input.Get();
71              EASSERTP(input.Get()==0x7e, /* != '~'*/
72                       Scr::Connection::UnsupportedKey,DebugInfo);
73              if (c=='6')
74                  THROWP(Scr::Connection::UnsupportedKey,DebugInfo);
75              if (c>'9')
76                  THROWP(Scr::Connection::UnsupportedKey,DebugInfo);
77              if (c>'6')c--; // code sequence 1,2,3,4,5, 7,8,9 (note,
78                             // that 6 is omitted)
79              return Key(Key::F1+ c-'1');
80          case '2': // 0,1,3,4 =  F9..F12; only '~' - INS
81              c=input.Get();
82              if (c=='~')
83                  return Key(Key::Insert);
84              if (c=='2')
85                  THROWP(Scr::Connection::UnsupportedKey,DebugInfo);
86              if (c>'4')
87                  THROWP(Scr::Connection::UnsupportedKey,DebugInfo);
88              if (c>'2')
89                  c--;
90              if (input.Get()!=0x7e) // != '~'
91                  THROWP(Scr::Connection::UnsupportedKey,DebugInfo);
92              return Key(Key::F9+ c-'0');
93
94          case '3': // delete <ESC>[3~ - delete
95              EASSERTP(input.Get()==0x7e, /* != '~'*/
96                       Scr::Connection::UnsupportedKey,DebugInfo);
97              return Key(Key::Delete);
98          case '5':
99              EASSERTP(input.Get()==0x7e, /* != '~'*/
100                      Scr::Connection::UnsupportedKey,DebugInfo);
101             return Key(Key::PageUp);
102
103         case '6':
104             EASSERTP(input.Get()==0x7e, /* != '~'*/
105                      Scr::Connection::UnsupportedKey,DebugInfo);
106             return Key(Key::PageDown);
107         case '7':// alternative coding
108             EASSERTP(input.Get()==0x7e, /* != '~'*/
109                      Scr::Connection::UnsupportedKey,DebugInfo);
110             return Key(Key::Home);
111         case '8':// alternative coding
```

```
112             EASSERTP(input.Get()==0x7e, /* != '~'*/
113                     Scr::Connection::UnsupportedKey,DebugInfo);
114             return Key(Key::End);
115         case 'F':
116             return Key(Key::End);
117         case 'H':
118             return Key(Key::Home);
119
120
121         default:
122             ;
123                 THROWP(Scr::Connection::UnsupportedKey,DebugInfo);
124         }
125     }
126
127     input.UnGet();
128     return Key(Key::Escape);
129 }
130
131 void Scr::VT100Compatible::RealGotoYX(const Position &  p)throw(
        ConnectionError)
132 {
133     output
134         << CURSOR_YX(p.row+1,p.col+1);
135 }
136
137 void Scr::VT100Compatible::Refresh()throw(ConnectionError)
138 {
139     output << HIDE_CURSOR;
140     output << CURSOR_HOME;
141     DisplayStyle p=copyBuffer[0][0].style;
142     Uint I=0,
143     J=0;
144
145     for (Uint i=0;i<controlBuffer.GetHeight();i++)//for each row
146         for (Uint j=0;j<controlBuffer.GetWidth();j++)// for each col
147         {
148             if (controlBuffer[i][j].c != 0)
149             {
150                 if (controlBuffer[i][j]!=copyBuffer[i][j])
151                 {
152                     if (i!=I || j!=J)
153                     {
154                         RealGotoYX(Position(i,j));
155                         I=i;
156                         J=j;
157                     }
158                     if (p!= controlBuffer[i][j].style)
159                     {
160                         p= controlBuffer[i][j].style;
161                         output << SET_ATTR(ATTR_RESET);
162                         output << SET_ATTR(static_cast<int>(p.GetFgColor()
                                ));
163                         output << SET_ATTR(static_cast<int>(p.GetBgColor()
                                ));
164                         if ((p.GetFgStyle()==Fg::Bright) !=0)
165                             output << SET_ATTR(ATTR_BRIGHT);
166
167                     }
168
169                     // print character itself only if it is printable
170                     if (controlBuffer[i][j].c == 0x7f)
```

```
171                         EncodeUTF8(output,0x2421);// D E L
172                     else if (controlBuffer[i][j].c >= ' ')
173                         EncodeUTF8(output,controlBuffer[i][j].c);
174                     else
175                         EncodeUTF8(output,0x2400+controlBuffer[i][j].c);
176                     // Unicode characters representing teletype mnemonics
                            for
177                     // first 31 ASCII characters
178
179                     J++;
180                 }
181             }// controlBuffer[i][j].c == 0  -- do not print anything
182             else// but mention, that last character was 2-column CJK.
183             {
184                 if (j!=0 && j==(J+1))
185                     J++;
186             }
187
188         }
189     // display cursor if requested
190     if (cursorFlags&cursorVisible)
191     {
192         if (cursorFlags&cursorForced)
193             RealGotoYX(cursorPosition);
194         output << SHOW_CURSOR;
195     }
196
197     output<<flush;
198     if (!output.good())
199         THROW(ConnectionError);
200     copyBuffer=controlBuffer;
201 }
202
203 void Scr::VT100Compatible::Resize(Uint rows, Uint cols)
204     throw()
205 {
206     GenericScreen::Resize(rows,cols);
207     copyBuffer.Resize(controlBuffer.GetHeight(),
208               controlBuffer.GetWidth());
209     copyBuffer.Fill(ScreenCharacter(0,DisplayStyle()));
210 }
211
212 void Scr::VT100Compatible::CleanUp()  throw(ConnectionError)
213 {
214     output << SHOW_CURSOR;
215     output<<SET_ATTR(ATTR_RESET)<<ERASE_SCREEN  << CURSOR_HOME;
216 }
217
218 Scr::VT100Compatible::~VT100Compatible()throw()
219 {
220     RexIOLog(LogLevelModerate)<<"~VT100Compatible()"<<endl;
221 }
```

4.22 lib/screen/src/subscreen/subscreen.c++

```
1 #include <iostream>
2 #include"screen.h++"
```

```
 3 #include"screenbuffer.h++"
 4 #include"subscreen.h++"
 5 #include"throw.h++"
 6 #include"utf8.h++"
 7 using namespace std;
 8 using namespace Scr;
 9
10 Scr::SubScreen::SubScreen(GenericScreen & _parent, Uint _y_offset,
11                 Uint _x_offset, Uint _h,
12                 Uint _w) throw()
13     : ScreenBase(),
14       parent(_parent),
15       offset(_y_offset,_x_offset),
16       s(_h,_w)
17 {
18     ;//nothing
19 }
20
21 void Scr::SubScreen::Clear()throw()
22 {
23     parent.GotoYX(offset.row,offset.col);
24     parent.Rectangle(static_cast<wchar_t>(' '),s);
25 }
26
27 void Scr::SubScreen::SetBgColor(Bg::Color col)throw()
28 {
29     parent.SetBgColor(col);
30 }
31
32 void Scr::SubScreen::SetFgColor(Fg::Color col)throw()
33 {
34     parent.SetFgColor(col);
35 }
36
37 void Scr::SubScreen::SetFgStyle(Fg::Style s)throw()
38 {
39     parent.SetFgStyle(s);
40 }
41
42 void Scr::SubScreen::GotoYX(Uint y, Uint x)
43     throw(GotoOutOfRange)
44 {
45     aPoint.row=y;
46     aPoint.col=x;
47 }
48
49 inline void Scr::SubScreen::ParentGotoYXForPrinting()throw(PrintOutOfRange
       )
50 {
51     try
52     {
53         parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
54     }
55     catch (GotoOutOfRange)
56     {
57         THROW(PrintOutOfRange);
58     }
59 }
60
61 void Scr::SubScreen::AddText(const char * text)throw(PrintOutOfRange,
62                              IllegalCharacter)
63 {
```

```
64      ParentGotoYXForPrinting();
65      parent.AddSubscreenText(text,s.width-aPoint.col);
66      //let any exception from parent function propagate to caller.
67      aPoint.col=parent.GetX()-offset.col;
68      //set position after successful execution (assume strong exception
            safety
69      //guarantee, that is "if exception is thrown, state remains unchanged
            ")
70  }
71
72  void Scr::SubScreen::AddText(const std::string & text)
73      throw(PrintOutOfRange, IllegalCharacter)
74  {
75      AddText(text.c_str());
76  }
77
78  void Scr::SubScreen::AddText(const wchar_t * text)throw(PrintOutOfRange,
79                               IllegalCharacter)
80  {
81      ParentGotoYXForPrinting();
82      parent.AddSubscreenText(text,s.width-aPoint.col);
83      aPoint.col=parent.GetX()-offset.col;
84  }
85
86  void Scr::SubScreen::AddText(const std::wstring & text)
87      throw(PrintOutOfRange,
88            IllegalCharacter)
89  {
90      AddText(text.c_str());
91  }
92
93  Uint Scr::SubScreen::AddTextCols(const wchar_t * text, Uint limitcols)
94      throw(PrintOutOfRange, IllegalCharacter)
95  {
96      try
97      {
98          parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
99      }
100     catch (GotoOutOfRange)
101     {
102         THROW(PrintOutOfRange);
103     }
104     if ((aPoint.col+limitcols)>s.width)
105         THROW(PrintOutOfRange);
106
107     Uint printedlen = parent.AddTextCols(text, limitcols);
108     aPoint.col += printedlen;
109
110 //    aPoint.col+=len;
111     return printedlen;
112 }
113
114 Uint SubScreen::AddTextCols(const std::wstring & text, Uint limitcols)
115     throw(PrintOutOfRange, IllegalCharacter)
116 {
117     return AddTextCols(text.c_str(), limitcols);
118 }
119
120 void Scr::SubScreen::HorizontalLine(char c, Uint n)
121     throw(PrintOutOfRange, IllegalCharacter)
122 {
123     parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
```

```
124     parent.HorizontalLine(c,n);
125     aPoint.col+=n;
126 }
127
128 void Scr::SubScreen::HorizontalLine(wchar_t c, Uint n)
129     throw(PrintOutOfRange, IllegalCharacter)
130 {
131     parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
132     parent.HorizontalLine(c,n);
133     aPoint.col+=n;
134 }
135
136 void Scr::SubScreen::VerticalLine(char c, Uint n)
137     throw(PrintOutOfRange, IllegalCharacter)
138 {
139     parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
140     parent.VerticalLine(c,n);
141     aPoint.row+=n;
142 }
143
144 void Scr::SubScreen::VerticalLine(wchar_t c, Uint n)
145     throw(PrintOutOfRange, IllegalCharacter)
146 {
147     parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
148     parent.VerticalLine(c,n);
149     aPoint.row+=n;
150 }
151
152 void Scr::SubScreen::Rectangle(wchar_t c, const Size & s)
153     throw(PrintOutOfRange, IllegalCharacter)
154 {
155     parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
156     parent.Rectangle(c,s);
157     aPoint+=s;
158 }
159
160 void Scr::SubScreen::Rectangle(char c, const Size & s)
161     throw(PrintOutOfRange, IllegalCharacter)
162 {
163     parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
164     parent.Rectangle(c,s);
165     aPoint+=s;
166 }
167
168 void Scr::SubScreen::AddCharacter(char c)throw(PrintOutOfRange)
169 {
170     try
171     {
172         parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
173     }
174     catch(Scr::Screen::GotoOutOfRange & e)
175     {
176         throw(PrintOutOfRange(string(e.what())+"\n " __WHERE_AM_I__));
177     }
178     parent.AddCharacter(c);
179     aPoint.col++;
180 }
181
182 void Scr::SubScreen::AddCharacter(wchar_t  c)throw(PrintOutOfRange,
183                                   IllegalCharacter)
184 {
185     try
```

```
186     {
187     parent.GotoYX(aPoint.row+offset.row,aPoint.col+offset.col);
188     }
189     catch(Scr::Screen::GotoOutOfRange & e)
190     {
191         throw(PrintOutOfRange(string(e.what())+"\n " __WHERE_AM_I__));
192     }
193     parent.AddCharacter(c);
194     aPoint.col++;
195 }
196
197 void Scr::SubScreen::ForceCursorPosition(Position p)throw(RangeError)
198 {
199     if (p.col>=GetWidth()) THROW(RangeError);
200     if (p.row>=GetHeight()) THROW(RangeError);
201     parent.ForceCursorPosition(p + offset);
202 }
203
204 void Scr::SubScreen::HideCursor()throw(CursorVisibilityNotSupported)
205 {
206     parent.HideCursor();
207 }
208
209 void Scr::SubScreen::ShowCursor()throw(CursorVisibilityNotSupported)
210 {
211     parent.ShowCursor();
212 }
213 void Scr::SubScreen::Refresh()
214     throw(ConnectionError) // parent object may throw this
215                         // exception, and then calling
216                         // function will have to catch it.
217
218 {
219     parent.Refresh();
220 }
221
222 void Scr::SubScreen::Resize(Uint rows, Uint cols)
223     throw(SubscreenResize)
224 {
225     THROW(SubscreenResize);
226 }
227
228 const char * Scr::SubScreen::GetType() const throw(TerminalTypeUnknown)
229 {
230     return parent.GetType();
231 }
232
233 Scr::Uint Scr::SubScreen::GetHeight() const throw()
234 {
235     return s.height;
236 }
237
238 Scr::Uint Scr::SubScreen::GetWidth() const throw()
239 {
240     return s.width;
241 }
242
243 Scr::Screen * Scr::SubScreen::
244 CreateSubScreen(Uint _y_offset, Uint _x_offset, Uint _h,
245                 Uint _w)throw(SubscreenOutOfRange)
246 {
247     SubScreenRangeCheck();
```

```
248     // if no exceptional conditions, just create and return new
249     // subscreen
250
251     // no difference whenever it will be another subscreen of parent
252     // or subscreen of subscreen... but less recursive calls needed to
253     // refresh this subscreen or print anything.
254     return new SubScreen(parent,
255                          offset.row+_y_offset,
256                          offset.col+_x_offset,
257                          _h, _w);
258 }
259
260 bool Scr::SubScreen::GetCursorVisibility() const throw()
261 {
262     return parent.GetCursorVisibility();
263 }
264
265 Scr::SubScreen::~SubScreen()throw()
266 {
267     ;//nothing
268 }
```

4.23 lib/screen/src/terminfo/terminfocore.c++

```
 1 #include "terminfodatabase.h++"
 2 #include "terminfo.h++"
 3 #include "throw.h++"
 4 #include "commons.h++"
 5
 6 using namespace Scr::TI;
 7
 8  boost::mutex GlobalTIMTX;
 9
10 /*!
11   initialized after first call to GetTermInfo
12  */
13 static TerminfoCore * GlobalInstance = 0;
14
15 static TerminfoDatabase * db = 0;
16
17 static Scr::Uint  NumberOfEntries = 0;
18
19 TerminfoCore::TerminfoCore() throw()
20     :entries()//global entries resource.
21 {
22     db = new TerminfoDatabase();
23 }
24
25 static int ctr = 0;
26 TerminfoCore::~TerminfoCore() throw()//global entries resource.
27 {
28     ctr++;
29     using namespace std;
30     RexIOLog(LogLevelModerate) << "Deleting Terminfo database" << endl;
31     delete db;
32     db = 0;
33     GlobalInstance = 0;
```

```
34      for (Dictionary<TerminfoEntry*>::iterator it =entries.begin();
35           it!=entries.end(); ++it)
36          delete *it;
37  }
38
39  const TerminfoEntry & TerminfoCore::__GetTerminfo(const char * name)
40      throw(NotSupportedTerminalType)
41  {
42      boost::mutex::scoped_lock lock (GlobalTIMTX);
43
44      Dictionary<TerminfoEntry*>::iterator it =
45              entries.find (name);
46
47
48
49      if (it == entries.end())// if such TerminfoEntry isn't yet retrieved
50      {                       //  create new one
51          boost::shared_ptr<std::ifstream> fp;
52          try
53          {
54              fp= (db->OpenFile(name));
55          }
56          catch (NotSupportedTerminalType)
57          {
58              throw;
59          }
60          catch (DatabaseException)
61          {
62              THROWP(FatalException,"Database fault");
63              // something bad must have happened
64          }
65          catch(...)
66          {
67              THROW(FatalException);
68          }
69          TerminfoEntry * e = new TerminfoEntry(*(fp));
70          entries.insert(name,e);
71          NumberOfEntries++;
72          return * e;
73      }
74      else
75      {// return existing one otherwise.
76          NumberOfEntries++;
77          return ** it;
78      }
79
80  }
81
82  void TerminfoCore::Initialize()throw(FailedToOpenDatabase)
83  {
84
85      boost::mutex::scoped_lock lock (GlobalTIMTX);
86      if (!GlobalInstance)
87          GlobalInstance= new TerminfoCore();
88
89      EASSERT(db->GetDatabaseStatus(),FailedToOpenDatabase);
90  }
91
92  bool TerminfoCore::GetDatabaseStatus()throw(DatabaseNotOpen)
93  {
94      if (db)
95          return db->GetDatabaseStatus();
```

```
 96     else
 97         THROW(DatabaseNotOpen);
 98 }
 99
100 const TerminfoEntry & TerminfoCore::GetTerminfo(const char * name)
101     throw(NotSupportedTerminalType,FailedToOpenDatabase)
102 {
103     Initialize();
104     return GlobalInstance->__GetTerminfo(name);
105 }
106
107 void TerminfoCore::CleanUp()throw()
108 {
109     delete GlobalInstance;
110     GlobalInstance=0;
111 }
112
113 void TerminfoCore::FreeTerminfoEntry()throw()
114 {
115     using namespace std;
116     boost::mutex::scoped_lock lock (GlobalTIMTX);
117     NumberOfEntries--;
118     RexIOLog(LogLevelModerate) << "FreeTerminfoEntry " << NumberOfEntries
            << endl;
119     if (NumberOfEntries == 0)
120     {
121         if (GlobalInstance)
122             GlobalInstance->CleanUp();
123         else
124             THROW(LogicError);
125     }
126 }
```

4.24 lib/screen/src/terminfo/terminfodatabase.c++

```
 1 #include "terminfodatabase.h++"
 2 #include <sys/types.h>
 3 #include <sys/stat.h>
 4 #include <unistd.h>
 5 #include <fstream>
 6 #include <iostream>
 7 #include"throw.h++"
 8 #include"trace.h++"
 9
10 const char * terminfoPath[]=
11 {
12     // paths, where terminfo database may reside
13
14     "/usr/share/terminfo",
15     "/lib/terminfo",
16     "/etc/terminfo",
17     0
18 };
19
20 using namespace Scr::TI;
21
22 TerminfoDatabase::TerminfoDatabase()throw()
```

```
23 {
24     /*look for terminfo directory*/
25     struct stat statbuf;
26     for (int i = 0 ; terminfoPath[i] ; i++)
27     {
28         if (stat(terminfoPath[i],&statbuf))
29             /* if stat failed*/
30             continue;
31 
32         if (!S_ISDIR(statbuf.st_mode))
33             /* file found, but it is not a directory */
34             continue;
35 
36         if (!(statbuf.st_mode & S_IROTH))
37             /* do we have read permission? */
38             continue;
39 
40         /*proper initialization*/
41         {
42             status = true;
43             path.assign(terminfoPath[i]);
44             RexIOLog(LogLevelModerate) << "Created link to terminfo
                   database in path "
45                     << path << std::endl;;
46             return;
47         }
48     }
49 
50     status = false;
51 }
52 
53 boost::shared_ptr<std::ifstream>
54 TerminfoDatabase::OpenFile(const char * name)
55     throw(FailedToOpenDatabase,
56           NotSupportedTerminalType,
57           FailedToLoadDatabaseEntry)
58 
59 {
60     // do not do anything if failed to open DB
61     EASSERT(status, FailedToOpenDatabase);
62 
63 
64     std::string fullPath = path;
65     fullPath.push_back('/');
66     fullPath.push_back(name[0]);
67     fullPath.push_back('/');
68     fullPath.append(name);
69     RexIOLog(LogLevelModerate) << "Looking for terminfo entry for " <<
           name
70               << " in file " << fullPath << std::endl;
71     struct stat statbuf;
72 
73     if (stat(fullPath.c_str(),&statbuf))
74         /* if stat failed, then probably file does not exist*/
75         THROW(NotSupportedTerminalType);
76 
77     boost::shared_ptr<std::ifstream>
78         result ( new
79                  std::ifstream( fullPath.c_str(),
80                                 std::ifstream::in | std::ifstream::binary
                                       ) );
81 
```

```
82     if (result->good())
83         return result;
84     else
85         THROW(FailedToLoadDatabaseEntry);
86
87 }// OpenFile
88
89 bool TerminfoDatabase::GetDatabaseStatus()throw()
90 {
91     return status;
92 }
```

4.25 lib/screen/src/terminfo/terminfoentry.c++

```
 1 #include <stack>
 2 #include <sstream>
 3 #include "terminfo.h++"
 4 #include "terminfokeymap.h++"
 5 #include "screen.h++"
 6 #include "throw.h++"
 7 #include "capabilities.h++"
 8
 9 using namespace Scr::TI;
10 using namespace std;
11
12 /*refer to man term(5) for more details*/
13 static const unsigned short MagicNumber = 0432;
14
15 TerminfoEntry::TerminfoEntry (std::ifstream & ifile)throw()
16 {
17     RexIOLog(LogLevelLow) << "TerminfoEntry::TerminfoEntry" <<endl;
18     short magic;
19     // verify magic number
20     ifile.read (reinterpret_cast<char*>(&magic), 2);
21     EASSERTP (magic==MagicNumber, FatalException, "Invalid terminfo file")
           ;
22
23     // read basic data
24     ifile.read (reinterpret_cast<char*>(&Meta), sizeof(Meta));
25
26 #define annotate(x) RexIOLog(LogLevelModerate) << #x<< "==" << Meta.x <<
       endl;
27     annotate(namesSize);
28     annotate (numBooleans);
29     annotate (numIntegers);
30     annotate (numOffsets);
31     annotate (stringTableSize);
32
33     //booleans section + '\0' + Integers section + '\0' + Offsets
34     //section  + String table itself
35
36     size_t text_sz
37             =Meta.namesSize+Meta.numBooleans+2*Meta.numIntegers
38             +2*Meta.numOffsets+2+Meta.stringTableSize
39             +((Meta.namesSize+Meta.numBooleans+1)&1);
40
41     text = new char[text_sz];
```

```
42
43      ifile.read (text, text_sz);
44
45      char * tmptxt = text;
46      Data.names=tmptxt;
47      tmptxt+=Meta.namesSize; // '\0'
48      Data.booleans=tmptxt;
49      tmptxt+=Meta.numBooleans;
50
51      // numbers section is 2-byte-aligned
52      tmptxt+=((Meta.namesSize+Meta.numBooleans)&1) ;
53
54      Data.numbers=
55              reinterpret_cast<short*>(tmptxt);
56
57      Data.strings = new char*[Meta.numOffsets];
58      tmptxt+=2*Meta.numIntegers;
59
60      for ( int i = 0 ; i!=Meta.numOffsets-1 ; i++)
61      {
62          if ((reinterpret_cast<short*>(tmptxt))[i]==-1)
63          {
64              Data.strings[i] = 0;
65              RexIOLog(LogLevelVerbose) << "feature not supported" << endl;
66          }
67          else
68          {
69              Data.strings[i]=tmptxt+2*Meta.numOffsets
70                      + (reinterpret_cast<short*>(tmptxt))[i];
71              RexIOLog(LogLevelVerbose) << i << ". " << Data.strings[i] <<
                    endl;
72          }
73      }
74      // Terminfo database retrieved. create keymap.
75
76      keymap = new Keymap(*this);
77  }
78
79  Keymap & TerminfoEntry::GetKeymap() const
80  {
81      return *keymap;
82  }
83
84  bool TerminfoEntry::GetBoolean (int i) const throw()
85  {
86      if (i > Meta.numBooleans)
87          return false;
88      else
89          return Data.booleans[i]==1;
90  }
91
92  short TerminfoEntry::GetInteger (int i) const throw()
93  {
94      if (i > Meta.numIntegers)
95          return  -1;
96      else
97          return Data.numbers[i];
98  }
99
100 const char * TerminfoEntry::GetString (int i) const throw()
101 {
102     RexIOLog(LogLevelVerbose) << "GetString " << Data.strings[i] << endl;
```

```
103     if (i > Meta.numOffsets)
104         return 0;
105     else
106         return Data.strings[i];
107 }
108
109 std::string TerminfoEntry::ParseString (int i, Uint * param)
110         const throw(CapabilityNotSupported, ParseError)
111 {
112     boost::mutex::scoped_lock lock (textmod_mtx);
113     const char * s=GetString (i);
114     EASSERT (s, CapabilityNotSupported); // refuse to process not
115     // supported capabilities
116     RexIOLog(LogLevelModerate) << "Processing String " << s <<  '/' <<
            param[0]<< endl;
117
118     char * ps=const_cast<char*>(s); // safe const cast, as each change is
119                                     // reverted
120     std::string result;
121     stack<int> params;
122     while (*ps)
123     {
124         if (*ps!='%')
125         {
126             result.push_back (*ps);
127             ps++;
128         }
129         else //parse % sequence
130         {
131             ps++;
132             if (*ps=='%')
133             {
134                 result.push_back ('%');
135                 continue;
136             }
137             else if (*ps=='i')// add 1 to first two params
138             {
139                 param[0]++;
140                 param[1]++;
141             }
142             else if (*ps=='p')// push parameter
143             {
144                 ps++;
145                 params.push (param[*ps-'1']);
146             }
147             else if (*ps=='{')// push integer constant (support only
                    positive)
148             {
149                 Uint i = 0;
150                 while (*ps!='}')
151                 {
152                     i*=10;
153                     i+=*(++ps)-'0';
154                 }
155                 params.push (i);
156             }
157             else //  %[[:]flags][width[.precision]][doxXs]
158             {
159                 //according to manual: "as in printf"... so we'll use
160                 //printf itself:D
161
162                 // first extract format string
```

```
163                 char * s = ps -1; // starting from %
164
165                 while (*ps!='d' && *ps!='o' && *ps!='x' && *ps!='X' && *ps
                        !='s')
166                 {
167                     *ps++;
168                 }
169                 char t=ps[1];
170
171                 char tmp[80];
172                 {// mtx scope
173                     ps[1]=0; // format string must be null terminated
174
175                     RexIOLog(LogLevelModerate) << "Processing int " << s
                            << ' ' << params.top () << endl;
176                     sprintf (tmp, s, params.top ());
177                     ps[1]=t;// restore format string (RELEASE MUTEX HERE)
178                 }
179                 result.append (tmp);
180                 params.pop ();
181             }
182             ps++;
183         }
184     }
185     RexIOLog(LogLevelModerate) << "Terminfo processing result: " << result
           << endl ;
186     return result;
187
188 }
189
190 const std::string
191 TerminfoEntry::GotoYX (const Scr::Position & newPosition) const throw(
        CapabilityNotSupported)
192 {
193     Position tmp (newPosition);
194     return ParseString (CursorAddress, & (tmp.row));
195 }
196
197 const std::string TerminfoEntry::GotoYX (const Scr::Position & newPosition
        ,
198         const Scr::Position & oldPosition)
199 const throw(CapabilityNotSupported)// there is field for optimization w/
        relative movements, but don't
200             //  implement it yet
201 {
202     return GotoYX (newPosition);
203 }
204
205 const std::string TerminfoEntry::SetDisplayStyle (const Scr::DisplayStyle
        s)
206 const throw(CapabilityNotSupported)
207 {
208 // stringstream ss;
209 //      ss<< "\x1b["
210 //              << ((s.GetFgStyle ()==Scr::Fg::Bright)?"1;":"0;")// set
        bright if
211 //              // needed
212 //              << s.GetFgColor () << ';' << s.GetBgColor ()
213 //              <<'m';
214 //  string result("\x1b[");
215     int i=0;
216     char result[10];
```

```
217     result[i++]=0x1b;
218     result[i++]='[';
219     result[i++]='0'+(s.GetFgStyle ()==Scr::Fg::Bright);
220     if (s.GetFgColor ()>=Scr::Fg::Black)
221     {
222         result[i++]=';';
223         result[i++]='0'+Scr::Fg::Black/10;
224         result[i++]='0'+s.GetFgColor ()-Scr::Fg::Black;
225     }
226     if (s.GetBgColor ()>=Scr::Bg::Black)
227     {
228         result[i++]=';';
229         result[i++]='0'+Scr::Bg::Black/10;
230         result[i++]='0'+s.GetBgColor ()-Scr::Bg::Black;
231     }
232     result[i++]='m';
233     return string(result,i);
234 }
235
236 const std::string TerminfoEntry::SetDisplayStyle (const Scr::DisplayStyle
        s,
237                                                    const Scr::DisplayStyle
                                                           os)
238 const throw(CapabilityNotSupported)
239 {
240     return SetDisplayStyle(s);
241     int i=0;
242     char result[10];
243     result[i++]=0x1b;
244     result[i++]='[';
245     result[i++]='0'+(s.GetFgStyle ()==Scr::Fg::Bright);
246     if (s.GetFgColor ()!=os.GetFgColor()
247         and s.GetFgColor ()>=Scr::Fg::Black)
248     {
249         result[i++]=';';
250         result[i++]='0'+Scr::Fg::Black/10;
251         result[i++]='0'+s.GetFgColor ()-Scr::Fg::Black;
252     }
253     if (s.GetBgColor ()!=os.GetBgColor()
254         and s.GetBgColor ()>=Scr::Bg::Black)
255     {
256         result[i++]=';';
257         result[i++]='0'+Scr::Bg::Black/10;
258         result[i++]='0'+s.GetBgColor ()-Scr::Bg::Black;
259     }
260     result[i++]='m';
261     return string(result,i);
262 }
263
264 const std::string TerminfoEntry::ShowCursor () const throw(
        CapabilityNotSupported)
265 {
266     const char * s = GetString (CursorVisible);
267     if (s not_eq NULL)
268         return string (s);
269     else
270     {
271         s = GetString (CursorNormal);
272         if (s not_eq NULL)
273             return string (s);
274         else
275             THROW(CapabilityNotSupported);
```

```
276     }
277 }
278
279 const std::string TerminfoEntry::HideCursor () const throw(
        CapabilityNotSupported)
280 {
281     const char * s = GetString (CursorInvisible);
282     if (s)
283     {
284         return string (s);
285     }
286     else
287     {
288         THROW(CapabilityNotSupported);
289     }
290 }
291
292 const std::string TerminfoEntry::CursorHome () const throw(
        CapabilityNotSupported)
293 {
294     const char * s = GetString (TI::CursorHome);
295     if (s)
296     {
297         return string (s);
298     }
299     else
300     {
301         THROW(CapabilityNotSupported);
302     }
303 }
304 TerminfoEntry::~TerminfoEntry ()throw()
305 {
306     using namespace std;
307     RexIOLog(LogLevelLow) << "freing TI entry for" << Data.names << endl;
308     delete[] text;
309     delete[] Data.strings;
310     delete keymap;
311 }
```

4.26 lib/screen/src/terminfo/terminfokeymap.c++

```
 1 #include "terminfokeymap.h++"
 2 #include "capabilities.h++"
 3
 4 using namespace Scr::TI;
 5
 6 Keymap::Keymap (const TerminfoEntry & te)throw()
 7 {
 8     InitializeKeymap (te);
 9 }
10
11 //const char * s
12 #define map_key_and_capability(cap,key)                                  \
13 {                                                                        \
14     s = te.GetString(cap);                                               \
15     if (s != NULL and s[0])/*string exists and is nonempty*/             \
16     {                                                                    \
```

```
17          keyboard.insert(s,static_cast<Scr::Key>(key));          \
18      }                                                           \
19 }
20 #define map_key(k) map_key_and_capability(Scr::TI::Key ## k,Scr::Key::k)
21 void Keymap::InitializeKeymap (const TerminfoEntry & te)throw()
22 {
23     const char * s;
24 //  map_capability(TI::KeyF1,Key::F1);
25
26     keyboard.insert("\x1b[H",Scr::Key::Home);// default code for Home
27     //(if terminfo specifies another it's meaning, it will be overriden)
28     keyboard.insert("\x1b[F",Scr::Key::End  );
29     keyboard.insert("\x1b[A",Scr::Key::Up   );
30     keyboard.insert("\x1b[B",Scr::Key::Down );
31     keyboard.insert("\x1b[C",Scr::Key::Right);
32     keyboard.insert("\x1b[D",Scr::Key::Left );
33
34     keyboard.insert("\x1b[1;5A",Scr::Key::CtrlUp    );
35     keyboard.insert("\x1b[1;5B",Scr::Key::CtrlDown  );
36     keyboard.insert("\x1b[1;5C",Scr::Key::CtrlRight);
37     keyboard.insert("\x1b[1;5D",Scr::Key::CtrlLeft  );
38     map_key (F1);
39     map_key (F2);
40     map_key (F3);
41     map_key (F4);
42     map_key (F5);
43     map_key (F6);
44     map_key (F7);
45     map_key (F8);
46     map_key (F9);
47     map_key (F10);
48     map_key (F11);
49     map_key (F12);
50     map_key (F13);
51     map_key (F14);
52     map_key (F15);
53     map_key (F16);
54     map_key (F17);
55     map_key (F18);
56     map_key (F19);
57     map_key (F20);
58     map_key (F21);
59     map_key (F22);
60     map_key (F23);
61     map_key (F24);
62     map_key (F25);
63     map_key (F26);
64     map_key (F27);
65     map_key (F28);
66     map_key (F29);
67     map_key (F30);
68     map_key (F31);
69     map_key (F32);
70     map_key (F33);
71     map_key (F34);
72     map_key (F35);
73     map_key (F36);
74
75     map_key_and_capability(Scr::TI::KeyBtab,Scr::Key::BackTab);
76     map_key_and_capability(Scr::TI::KeyDc,Scr::Key::Delete);
77     map_key_and_capability(Scr::TI::KeyPpage,Scr::Key::PageUp);
78     map_key_and_capability(Scr::TI::KeyNpage,Scr::Key::PageDown);
```

```
 79     map_key_and_capability(Scr::TI::Tab,Scr::Key::Tab);
 80     map_key (Home);
 81     map_key (End);
 82     map_key (Enter);
 83     map_key (Left);
 84     map_key (Right);
 85     map_key (Up);
 86     map_key (Down);
 87
 88 }
 89
 90 Keymap::validity
 91         Keymap::TestCode (const char * code)throw()
 92 {
 93     Dictionary<Scr::Key>::iterator it;
 94     it = keyboard.partial_find(code);
 95     return it.validity_test();
 96 }
 97
 98 Scr::Key Keymap::GetCode (const char * code)throw(Connection::
       UnsupportedKey)
 99 {
100     Dictionary<Scr::Key>::iterator it;
101     it = keyboard.find(code);
102     if (it.valid())
103         return *it;
104     else
105         THROW(Connection::UnsupportedKey);
106 }
```

4.27 lib/toolkit/src/activewidget.c++

```
 1 #include "activewidget.h++"
 2
 3 using namespace Scr;
 4 using namespace Scr::Tk;
 5
 6 ActiveWidget::ActiveWidget(Uint _height, Uint _width,
 7                            const DisplayStyle& _style,
 8                            const DisplayStyle& _activeStyle)throw() :
 9     Widget(_height, _width, _style), activeStyle(_activeStyle), active(
           false)
10 {;}
11
12 ActiveWidget::ActiveWidget(const DisplayStyle& _style,
13                            const DisplayStyle& _activeStyle)throw() :
14     Widget(_style), activeStyle(_activeStyle), active(false)
15 {;}
16
17 void ActiveWidget::OnFocus(FocusPolicy focustype)throw()
18 {
19     if(GetActive()) {
20         GetParent().PassFocusRequest(focustype);
21         return;
22     }
23     SetActive(true);
24 }
```

```
25
26 void ActiveWidget::OnUnFocus(FocusPolicy focustype)throw()
27 {
28     SetActive(false);
29 }
30
31 void ActiveWidget::OnKeyDown(Key key)throw()
32 {
33     if(key == Key::Enter)
34         OnAction();
35 }
36
37 void ActiveWidget::OnAction()throw()
38 {
39     RexIOLog(LogLevelModerate) << "ACTION?" << std::endl;
40 }
41
42 void ActiveWidget::SetActive(bool _active)throw()
43 {
44     active = _active;
45 }
46
47 bool ActiveWidget::GetActive()throw()
48 {
49     return active;
50 }
51
52 DisplayStyle& ActiveWidget::GetActiveStyle()throw()
53 {
54     return activeStyle;
55 }
56
57 void ActiveWidget::SetActiveStyle(const DisplayStyle& _activeStyle)throw()
58 {
59     activeStyle = _activeStyle;
60 }
61
62 ActiveWidget::~ActiveWidget()throw()
63 {
64     ;
65 }
```

4.28 lib/toolkit/src/boxgroup.c++

```
 1 #include "boxgroup.h++"
 2
 3 using namespace Scr;
 4 using namespace Scr::Tk;
 5
 6 // TODO: with error accumulator
 7 #define PRECISION 1000
 8
 9 BoxGroup::BoxGroup(Uint _height, Uint _width,
10                    const DisplayStyle & _style)throw()
11     : WidgetGroup(_height, _width, _style),
12       alignPolicy(Begin)
13 {
```

```
14     ;
15 }
16
17 BoxGroup::BoxGroup(const WidgetGroup & base)throw()
18     : WidgetGroup(base),
19       alignPolicy(Begin)
20 {
21     ;
22 }
23
24 BoxGroup::~BoxGroup()throw(){;}
25
26 void BoxGroup::SwapWidgets(Widget& widget1, Widget& widget2)throw()
27 {
28     elements.swap(&widget1, &widget2);
29     OnResize();
30
31 }
32
33 void BoxGroup::AddWidget(Widget& widget)throw()
34 {
35     elementsLayout[&widget] = LayoutData(1);
36     Window::AddWidget(widget);
37     ArrangeContents();
38 }
39
40 void BoxGroup::AddWidget(Widget& widget, Uint stretchFactor)throw()
41 {
42     elementsLayout[&widget] = LayoutData(stretchFactor);
43     WidgetGroup::AddWidget(widget);
44     ArrangeContents();
45 }
46
47 void BoxGroup::DelWidget(Widget& widget)throw()
48 {
49     elementsLayout.erase(&widget);
50     WidgetGroup::DelWidget(widget);
51     ArrangeContents();
52 }
53
54 void BoxGroup::OnStart()throw()
55 {
56 //  ArrangeContents();
57     WidgetGroup::OnStart();
58 }
59
60 void BoxGroup::OnResize()throw()
61 {
62     ArrangeContents();
63     WidgetGroup::OnResize();
64 }
65
66 void BoxGroup::SetAlignPolicy(AlignPolicy _alignPolicy)throw()
67 {
68     if(alignPolicy != _alignPolicy)
69         WidgetGroup::OnResize();
70     alignPolicy = _alignPolicy;
71 }
72
73 BoxGroup::AlignPolicy BoxGroup::GetAlignPolicy()throw()
74 {
75     return alignPolicy;
```

```
76 }
```

4.29 lib/toolkit/src/button.c++

```
 1 #include "button.h++"
 2
 3 using namespace Scr;
 4 using namespace Scr::Tk;
 5
 6 Button::Button(Uint _height, Uint _width, const std::string& _label,
 7                const DisplayStyle& _style,
 8                const DisplayStyle& _activeStyle)throw() :
 9     ActiveWidget(_height, _width, _style, _activeStyle), label(_label)
10 {
11     SetMinSize(1, label.length());
12 }
13
14 Button::Button(const std::string& _label,
15                const DisplayStyle& _style,
16                const DisplayStyle& _activeStyle)throw() :
17     ActiveWidget(_style, _activeStyle), label(_label)
18 {
19     SetMinSize(1, label.length());
20 }
21
22 void Button::OnRedraw(Screen& screen)throw()
23 {
24     screen << (GetActive() ? GetActiveStyle() : GetStyle()) << Control::
           Clear
25            << Control::GotoYX(GetHeight()/2, (GetWidth() - label.length())
                /2)
26            << GetLabel();
27     if (GetActive())
28         screen.ForceCursorPosition(Position(0,0));
29 }
30
31 void Button::SetLabel(const std::string& _label)throw()
32 {
33     label = _label;
34 }
35
36 std::string& Button::GetLabel()throw()
37 {
38     return label;
39 }
40
41 Button::~Button()throw()
42 {
43     ;
44 }
```

4.30 lib/toolkit/src/checkbox.c++

```
 1 #include "checkbox.h++"
 2
 3 using namespace Scr;
 4 using namespace Scr::Tk;
 5
 6 Checkbox::Checkbox(Uint _height, Uint _width, const Label& _label,
 7                    const DisplayStyle& _style,
 8                    const DisplayStyle& _activeStyle)throw() :
 9     ActiveWidget(_height, _width, _style, _activeStyle), label(_label)
10 {
11     SetMinSize(1, label.GetMinWidth() + 4); // +4 because of the actual
           checkbox
12 }
13
14 Checkbox::Checkbox(const Label& _label,
15                    const DisplayStyle& _style,
16                    const DisplayStyle& _activeStyle)throw() :
17     ActiveWidget(_style, _activeStyle), label(_label)
18 {
19     SetMinSize(1, label.GetMinWidth() + 4);
20 }
21
22 void Checkbox::OnRedraw(Screen& screen)throw()
23 {
24     screen << Control::GotoYX(0,0)
25            << (GetActive() ? GetActiveStyle() : GetStyle()) << '['
26            << (GetState() ? 'X':' ') << ']'
27            << GetParent().GetStyle() << ' ' << label.GetStyle() << label;
28
29     if (GetActive())
30         screen.ForceCursorPosition(Position(0,3));
31 }
32
33 void Checkbox::OnAction()throw()
34 {
35     SetState(!GetState());
36 }
37
38 void Checkbox::SetLabel(const Label& _label)throw()
39 {
40     label = _label;
41 }
42
43 const Label& Checkbox::GetLabel()throw()
44 {
45     return label;
46 }
47
48 void Checkbox::SetState(bool _state)throw()
49 {
50     try {
51         if(state != _state)
52             GetParent().RedrawRequest(*this);
53     } catch(ParentNotDefined) {
54         ;
55     }
56     state = _state;
57 }
58
59 bool Checkbox::GetState()throw()
60 {
61     return state;
```

```
62 }
63
64 Checkbox::~Checkbox()throw()
65 {
66     ;
67 }
```

4.31 lib/toolkit/src/framedwindow.c++

```
 1 #include "framedwindow.h++"
 2
 3 using namespace Scr;
 4 using namespace Scr::Tk;
 5
 6 FramedWindow::FramedWindow(Uint _height, Uint _width,
 7                            const DisplayStyle& _style,
 8                            const FrameStyle& _frameStyle)throw()
 9     : FramedWindowBase<Window>(_height, _width, _style,
10                                _frameStyle)
11 {
12     ;
13 }
```

4.32 lib/toolkit/src/horizontalgroup.c++

```
 1 #include <iostream>
 2 #include <cmath>
 3 #include "horizontalgroup.h++"
 4
 5 using namespace Scr;
 6 using namespace Scr::Tk;
 7
 8 void HorizontalGroup::ArrangeContents()throw()
 9 {
10     Uint maxheight = GetHeight();
11     Uint maxwidth = GetWidth();
12
13     float totalmax = 0;
14     Uint totalmin = 0;
15     Uint coefsum = 0;
16
17     Uint freestep = 0;
18
19     bool stretchmax = false;
20
21     Uint visible_elems =  elements.size();
22     for(WidgetList::iterator i = elements.begin();
23         i != elements.end(); i++) {
24
25         if((*i)->IsHidden()) {
26             visible_elems--;
27             continue;
```

```
28          }
29
30          if((*i)->GetMaxWidth() == UintMax) // means, stretch to max
31              stretchmax = true;
32          else
33              totalmax += (*i)->GetMaxWidth();
34          // sum of coefficients will give some hint in dividing free
35          // space
36          coefsum += elementsLayout[*i].stretchFactor;
37
38          totalmin += (*i)->GetMinWidth();
39      }
40      bool usecoef = false;
41      if(totalmax > maxwidth || stretchmax) {
42          totalmax = maxwidth; // use whole available space
43          usecoef = true;
44      }
45
46      Uint addpoint = 0; // point at from which the first widget will be
            drawn
47      switch(alignPolicy) {
48      case Distribute:
49      case Begin:
50          addpoint = 0;
51          break;
52      case Center:
53          /* NOTE: (GetHeight() - totalmax)/2 can result in overflow first
54             during substraction and the division would divide the overflown
55             value. More desired behaviour is achieved by the below
                   operation.
56          */
57          addpoint = GetWidth()/2 - totalmax/2;
58          break;
59      case End:
60          addpoint = GetWidth() - totalmax;
61          break;
62      }
63
64      if(usecoef) { // base on coefficients space division
65
66          totalmax = maxwidth;
67
68          if(totalmin <= totalmax)
69              totalmax -= totalmin; // totalmax will be now a height that
70          // has to be distributed among elements
71          else
72              totalmax = 0;
73
74          for(WidgetList::iterator i = elements.begin();
75              i != elements.end(); i++) {
76
77              if((*i)->IsHidden())
78                  continue;
79
80              Widget &w = **i;
81              Uint coef = elementsLayout[&w].stretchFactor;
82
83              Uint distspace = roundf(((float)coef/(float)coefsum) *
                    totalmax);
84
85              if(w.GetMinWidth() >= distspace)
86                  distspace = 0;
```

```
87              else
88                  distspace -= w.GetMinWidth();
89
90              if(distspace + w.GetMinWidth() > w.GetMaxWidth())
91                  distspace = w.GetMaxWidth() - w.GetMinWidth();
92
93              // height = distributed space + minimal height of this widget
94              w.SetSize(maxheight, distspace + w.GetMinWidth());
95
96              totalmax -= distspace;
97              coefsum -= coef;
98          }
99          if(totalmax && visible_elems) { // distribute anything left to the
100             // first visible element that can aquire any more size
101             Widget *w;
102             for(WidgetList::iterator i = elements.begin();
103                 i != elements.end(); i++) {
104                 if((*i)->IsHidden() ||
105                     w->GetWidth() + totalmax > w->GetMaxHeight())
106                     continue;
107                 w = *i;
108                 break;
109             }
110             w->SetSize(w->GetHeight(), w->GetHeight() + totalmax);
111         }
112     }
113     else { // base on align model
114         if(alignPolicy == Distribute)
115             freestep = (maxwidth - totalmax)/elements.size();
116
117         Uint tmp = addpoint;
118         for(WidgetList::iterator i = elements.begin();
119             i != elements.end(); i++) {
120             if((*i)->IsHidden())
121                 continue;
122             Widget &w = **i;
123
124             tmp += w.GetMaxWidth();
125             w.SetSize(maxheight, w.GetMaxWidth());
126         }
127     }
128
129     // finally position all the elements,stacking the heights
130     for(WidgetList::iterator i = elements.begin();
131         i != elements.end(); i++) {
132         if((*i)->IsHidden())
133             continue;
134         Widget &w = **i;
135
136         w.SetPosition(0, addpoint);
137         addpoint += w.GetWidth() + freestep/*from the distributed model*/;
138     }
139
140 }
141
142 HorizontalGroup::HorizontalGroup(Uint _height,
143                                  Uint _width,
144                                  const DisplayStyle &  _style)throw()
145     :BoxGroup(_height, _width, _style)
146 {;}
147
148 HorizontalGroup::HorizontalGroup(const WidgetGroup & base)throw()
```

```
149     :BoxGroup(base)
150 {;}
151
152 HorizontalGroup::~HorizontalGroup()throw(){;}
```

4.33 lib/toolkit/src/inputbox.c++

```
  1 #include "inputbox.h++"
  2 #include "throw.h++"
  3
  4 using namespace Scr;
  5 using namespace Scr::Tk;
  6
  7 #define fillend() \
  8 if(textOffset + curChars  < text.length()) { \
  9     Uint w = GlyphWidth::Get(text[textOffset + curChars]); \
 10     while(curCols + w <= GetWidth() &&                     \
 11           textOffset + curChars < text.length()) { \
 12         curCols += w; \
 13         curChars++; \
 14         w = GlyphWidth::Get(text[textOffset + curChars]); \
 15     } \
 16 }
 17
 18 #define incoffset() { \
 19 Uint w = GlyphWidth::Get(text[textOffset]); \
 20 curCols -= w; \
 21 cursorPos -= w; \
 22 curChars--; \
 23 charPos--; \
 24 textOffset++; \
 25 }
 26
 27 Inputbox::Inputbox(Uint _width, const std::wstring& _text,
 28                    const DisplayStyle& _style,
 29                    const DisplayStyle& _activeStyle,
 30                    const InputboxStyle& _inputboxStyle)
 31     throw() :
 32     ActiveWidget(1, _width, _style, _activeStyle),
 33     cursorPos(0), charPos(0), curCols(0), curChars(0), textOffset(0),
 34     inputboxStyle(_inputboxStyle),maxLength(0)
 35 {
 36     SetText(_text);
 37     SetMinHeight(1);
 38     SetMaxHeight(1);
 39     SetMinWidth(3); // at least one char and a cursor
 40 }
 41
 42 Inputbox::Inputbox(const std::wstring& _text,
 43                    const DisplayStyle& _style,
 44                    const DisplayStyle& _activeStyle,
 45                    const InputboxStyle& _inputboxStyle
 46     )throw() :
 47     ActiveWidget(_style, _activeStyle),
 48     cursorPos(0), charPos(0), curCols(0), curChars(0), textOffset(0),
 49     inputboxStyle(_inputboxStyle),maxLength(0)
 50 {
```

```
51     SetText(_text);
52     SetMaxHeight(1);
53     SetMinHeight(1);
54     SetMinWidth(3); // at least one char and a cursor
55 }
56
57 void Inputbox::OnRedraw(Screen& screen)throw()
58 {
59     screen << (active?activeStyle:style) << Control::Clear
60             << Control::GotoYX (0, 0);
61
62     if(active)
63     {
64         if(inputboxStyle.cursorStyle.GetFgColor () == Fg::System &&
65                 inputboxStyle.cursorStyle.GetBgColor () == Bg::System)
66         {
67
68             screen.ShowCursor ();
69
70             screen.ForceCursorPosition (Position (0, cursorPos));
71
72             std::wstring temp = text.substr (textOffset, curChars);
73             screen << temp;
74
75         }
76         else
77         { // software cursor
78             std::wstring temp;
79             if(charPos)
80             {
81                 temp = text.substr (textOffset, charPos);
82                 screen << temp;
83             }
84
85             screen << inputboxStyle.cursorStyle;
86             screen << Control::GotoYX (0, cursorPos); // should it be
                    needed?
87             if(textOffset + charPos < text.length ())
88             {
89                 screen << text[textOffset + charPos];
90                 screen <<
91                 Control::GotoYX (0, cursorPos +
92                                  GlyphWidth::Get(text[textOffset + charPos
                                         ]));
93                 // the above width calculatation shouldn't be needed!!
94             }
95             else
96                 screen << ' ';
97             screen << activeStyle;
98
99             if(curChars > charPos + 1)
100            {
101                temp = text.substr (textOffset + charPos+1, curChars - (
                       charPos + 1));
102                screen << temp;
103            }
104        }
105    }
106    else
107    {
108        std::wstring temp = text.substr (textOffset, curChars);
109        screen << temp;
```

```
110     }
111 }
112
113 void Inputbox::SetOffset(Uint _textOffset)throw(OffsetOutOfRange)
114 {
115     if(_textOffset >= text.length())
116         THROW(OffsetOutOfRange);
117
118     textOffset = _textOffset;
119     curChars = 0;
120     curCols = 0;
121     charPos = 0;
122     cursorPos = 0;
123     fillend();
124 }
125
126 Uint Inputbox::GetOffset()throw()
127 {
128     return textOffset;
129 }
130
131 void Inputbox::SetText(const std::wstring& _text)throw()
132 {
133     textOffset = 0;
134     curChars = 0;
135     curCols = 0;
136     charPos = 0;
137     cursorPos = 0;
138     text = _text;
139     fillend();
140 }
141
142 const std::wstring& Inputbox::GetText()throw()
143 {
144     return text;
145 }
146
147 void Inputbox::SetMaxLength(Uint _maxLength)throw()
148 {
149     maxLength = _maxLength;
150 }
151
152 Uint Inputbox::GetMaxLength()throw()
153 {
154     return maxLength;
155 }
156
157 void Inputbox::OnKeyDown(Key key)throw()
158 {
159     Uint w;
160     if(key.IsASpecialKey ())
161     {
162         switch(key.GetSpecialKey ())
163         {
164             case Key::Home:
165                 SetOffset (0);
166                 break;
167             case Key::End:
168                 // if the input ending is on the sight
169                 if(textOffset + curChars <= text.length () &&
170                         curCols != GetWidth ())
171                 {
```

```
172                 charPos = curChars;
173                 cursorPos = curCols;
174                 break;
175             }
176
177             // otherwise put as much characters as can fit, counting
178             // from the end
179             {
180                 Uint totalw = 0, i = text.length ();
181                 curChars = 0;
182                 while(i--)
183                 {
184                     totalw += GlyphWidth::Get(text[i]);
185                     curChars++;
186                     if(totalw > GetWidth () - 1)
187                     {
188                         totalw -= GlyphWidth::Get(text[i]);
189                         curChars--;
190                         textOffset = i + 1;
191                         curCols = totalw;
192                         cursorPos = curCols;
193                         charPos = curChars;
194                         break;
195                     }
196                 }
197             }
198             break;
199         case Key::Backspace:
200             RexIOLog (LogLevelModerate) << "AA Backspace ";
201
202             if(textOffset || charPos) // not at the beginning
203             {
204                 Uint w = GlyphWidth::Get(text[textOffset + charPos -
                        1]);
205                 text.erase (textOffset + charPos - 1, 1);
206
207                 if(charPos)
208                 {
209                     cursorPos -= w;
210                     charPos--;
211                     curCols -= w;
212                     curChars--;
213
214                     fillend ()
215                 }
216                 else
217                     textOffset--;
218             }
219             break;
220         case Key::Delete:
221             if(textOffset + charPos != text.length ()) // not at the
                    end
222             {
223                 Uint w = GlyphWidth::Get(text[textOffset + charPos]);
224                 text.erase (textOffset + charPos, 1);
225
226                 curCols -= w;
227                 curChars--;
228
229                 if(charPos == curChars)
230                 { // last character was deleted
231
```

```
232                     if(GlyphWidth::Get(text[textOffset + curChars]) >
                            w)
233                     {
234                         // the deleted character was 1-width, but the
235                         // next in line is 2-width, make place for it
236                         incoffset ();
237                     }
238                 }
239                 fillend ();
240             }
241             break;
242         case Key::Right:
243             if(textOffset + charPos != text.length ())
244             {// not at the end
245                 w = GlyphWidth::Get(text[textOffset + charPos]);
246 
247                 cursorPos += w; // move cursor
248                 charPos++;
249 
250                 if(charPos == curChars) // if cursor at the end
251                 {
252                     w = GlyphWidth::Get(text[textOffset + charPos]);
253 
254                     curCols += w; // add char to the end
255                     curChars++;
256 
257                     // trim from the beginning, until fits
258                     while(curCols >= GetWidth () )
259                     {
260                         incoffset ();
261                     }
262 
263                     // possible place for one char left at the end
264                     fillend ();
265                 }
266             }
267             break;
268         case Key::Left:
269             if(charPos)
270             {
271                 cursorPos -= GlyphWidth::Get(text[textOffset + charPos
                        - 1]);
272                 charPos--;
273             }
274             else if(!charPos && textOffset)
275             { // cursor at the beginning
276                 // add to the front
277                 textOffset--;
278                 curChars++;
279                 curCols += GlyphWidth::Get(text[textOffset]);
280                 while(curCols > GetWidth () )
281                 {  // overflow
282                     // trim from the end, until ok
283                     curCols -= GlyphWidth::Get(text[textOffset +
                            curChars - 1]);
284                     curChars--;
285                 }
286             }
287             break;
288         default:
289             break;
290     }
```

```
291         return;
292     }
293     if(maxLength && text.length () == maxLength) // limit reached?
294         return;
295 
296     if (key<' ' or key == 0x7f) //   special ASCII (not a key);
297         return;
298     w = GlyphWidth::Get(key);
299 
300     text.insert (textOffset + charPos, 1, key);
301     cursorPos += w;
302     charPos++;
303     curCols += w;
304     curChars++;
305 
306     if(curChars == charPos)
307     { // cursor is at the end
308         while(curCols > GetWidth () -1)
309         {   // overflow
310             // trim from the beginning, until ok. Also leave one space for
311             // the cursor
312             incoffset ();
313         }
314     }
315     else
316     { // cursor somewhere in the middle, preserve textOffset
317         // current highlighted char's width
318         Uint curw = GlyphWidth::Get(text[textOffset + charPos]);
319 
320         // the cursor is almost at the end
321         // (also considering current highlighted width)
322         while(cursorPos >= GetWidth () + 1 - curw)
323         {
324 
325             // results in the offset change, make place
326             incoffset ();
327         }
328 
329         while(curCols > GetWidth () )
330         {  // overflow
331             // trim from the end, until ok
332 
333             curCols -= GlyphWidth::Get(text[textOffset + curChars - 1]);
334             curChars--;
335         }
336 
337         // there is one possible additional place for a character at the
                end
338         fillend ();
339     }
340 }
341 
342 Inputbox::~Inputbox()throw() {;}
```

4.34 lib/toolkit/src/label.c++

```
1 //#include "toolkit.h++"
```

```
 2 #include "label.h++"
 3
 4 using namespace Scr;
 5 using namespace Scr::Tk;
 6
 7 Label::Label(const DisplayStyle& _style)throw()
 8     :Widget( _style)
 9 {
10     label = "";
11     SetMaxHeight(1);
12     SetMinHeight(1);
13 }
14
15 Label::Label(Uint _width, const std::string& _label,
16          const DisplayStyle& _style)throw()
17     :Widget(1, _width, _style)
18 {
19     label = _label;
20     SetMaxHeight(1);
21     SetMinHeight(1);
22 }
23
24 Label::Label(const std::string& _label,
25              const DisplayStyle& _style)throw()
26     :Widget(1, _label.length(), _style)
27 {
28     label = _label;
29     SetMaxHeight(1);
30     SetMinHeight(1);
31 }
32
33 void Label::OnFocus(FocusPolicy focustype)throw()
34 {
35     GetParent().PassFocusRequest(focustype); // focus another element
36 }
37
38 void Label::OnUnFocus(FocusPolicy focustype)throw()
39 {
40     ;
41 }
42
43 void Label::OnRedraw(Screen& screen)throw()
44 {
45     screen <<((style.GetFgColor() == Fg::Transparent)?GetParent().GetStyle
           ():
46         style) << Control::Clear << Control::GotoYX(0,0);
47     try
48     {
49         screen << label;
50     }
51     catch (Scr::Screen::PrintOutOfRange)
52     {
53         for (Uint i = 0 ; i< screen.GetWidth() ; i++ )
54             screen << '.';
55     }
56 }
57
58 const std::string& Label::GetText() const throw()
59 {
60     return label;
61 }
62
```

```
63 void Label::SetText(std::string _label)throw()
64 {
65     SetMinWidth(label.length());
66     label = _label;
67 }
68
69 Screen& Scr::operator<<(Screen & screen,const Tk::Label& whatto)
70 {
71     return (screen << whatto.GetStyle() << whatto.GetText());
72 }
73
74 Label::~Label()throw() {;}
```

4.35 lib/toolkit/src/rootwindow.c++

```
 1 #include <iostream>
 2 #include<cstring>
 3 #include "rootwindow.h++"
 4 #include "trace.h++"
 5
 6 using namespace Scr::Tk;
 7
 8 void RootWindow::OnStart()throw()
 9 {
10     Window::OnStart();
11     OnRedraw(*screen);
12     *screen<<Control::Refresh;
13
14     RexIOLog(LogLevelLow)<<"RootWindow::OnStart()throw()"<<std::endl;
15 }
16
17 void RootWindow::OnKeyDown(Key key)throw()
18 {
19     if(key == Key::Tab)
20     {
21         OnFocus(TabFocus);
22     }
23     Window::OnKeyDown(key);
24
25     OnRedraw(*screen);
26     //*screen<<Control::Refresh;
27 }
28
29 RootWindow::RootWindow(std::istream & _input, std::ostream & _output,
30                const DisplayStyle & _style)throw():
31     Connection(_input, _output),
32     Window(screen->GetHeight(), screen->GetWidth(), _style)
33 {
34     SetParent(*this);
35 }
36
37 Scr::Screen& RootWindow::GetScreen()throw()
38 {
39     return *screen;
40 }
41
42 Scr::Uint RootWindow::GetAbsoluteColumn()throw(){return 0;}
```

```
 43 Scr::Uint RootWindow::GetAbsoluteRow()throw(){return 0;}
 44
 45 RootWindow& RootWindow::GetRootWindow()throw()
 46 {
 47     return *this;
 48 }
 49
 50 void RootWindow::OnRedraw(Screen& screen)throw()
 51 {
 52     screen.HideCursor() ;
 53     Window::OnRedraw(screen);
 54     screen.Refresh();
 55 }
 56
 57 void RootWindow::OnResize(Uint rows, Uint cols)throw()
 58 {
 59     size = Size(rows,cols);
 60     OnResize();
 61     OnRedraw(*screen);
 62 }
 63
 64 void RootWindow::LoadStylesheet(const char* filename)
 65     throw(FileNotOpened, Stylesheet::ParsingError)
 66 {
 67
 68     std::fstream fs(filename);
 69     if(!fs.is_open())
 70         THROW(FileNotOpened);
 71     SetStylesheet(new Stylesheet(fs));
 72 }
 73
 74 int RootWindow::Start(int argc, char **argv)
 75     throw(StartFailed,Screen::IllegalCharacter)
 76 {
 77     for(int i = 0;i<argc;i++) {
 78         // strlen("-style=") == 7
 79         if(strncmp(argv[i], "-style=", 7) == 0) {
 80             LoadStylesheet(argv[i]+7);
 81         }
 82     }
 83
 84     return Connection::Start(argc, argv);
 85 }
 86
 87 int RootWindow::Start()throw(StartFailed,Screen::IllegalCharacter)
 88 {
 89     return Connection::Start();
 90 }
 91
 92 void RootWindow::ForceRepaint()throw()
 93 {
 94     screen->GotoYX(0,0);
 95     screen->Rectangle((wchar_t)0xE007,size);
 96     screen->Refresh();
 97     OnRedraw(*screen);
 98
 99 }
100 RootWindow::~RootWindow()throw(){;}
```

4.36 lib/toolkit/src/scrollbar.c++

```
1 #include "scrollbar.h++"
2
3 using namespace Scr;
4 using namespace Scr::Tk;
5
6 ScrollbarBase::ScrollbarBase(Uint _width, Uint _height,
7                              const ScrollbarStyle &_scrollbarStyle)throw()
8     :Widget(_width, _height)
9 {
10    scrollbarStyle = _scrollbarStyle;
11 }
12
13 void ScrollbarBase::SetScrollSize(Uint _scrollSize)throw()
14 {
15    scrollSize = _scrollSize;
16 }
17
18 Uint ScrollbarBase::GetScrollSize() const throw()
19 {
20    return scrollSize;
21 }
22
23 void ScrollbarBase::SetScrollOffset(Uint _scrollOffset)throw()
24 {
25    scrollOffset = _scrollOffset;
26 }
27
28 Uint ScrollbarBase::GetScrollOffset() const throw()
29 {
30    return scrollOffset;
31 }
32
33 void ScrollbarBase::SetScrollProgress(float progress)throw()
34 {
35    SetScrollOffset(static_cast<Uint>(progress *
36                                      static_cast<float>(GetScrollSize()))
                                          );
37 }
38
39 float ScrollbarBase::GetScrollProgress() const throw()
40 {
41    return static_cast<float>(GetScrollOffset())/
42        static_cast<float>(GetScrollSize());
43 }
44
45 void ScrollbarBase::SetScrollbarStyle(const ScrollbarStyle&
      _scrollbarStyle)
46    throw()
47 {
48    RedrawRequest();
49    scrollbarStyle = _scrollbarStyle;
50 }
51
52 const ScrollbarStyle& ScrollbarBase::GetScrollbarStyle() const throw()
53 {
54    return scrollbarStyle;
55 }
56
57 HorizontalScrollbar::HorizontalScrollbar(
58    Uint _width, const ScrollbarStyle& _scrollbarStyle)throw()
```

```
59      : ScrollbarBase(1, _width, _scrollbarStyle) { ; }
60
61  void HorizontalScrollbar::OnRedraw(Screen& screen)throw()
62  {
63      screen<< Control::GotoYX(0, 0)
64            << scrollbarStyle.button << scrollbarStyle.buttonLeft
65            << Control::GotoYX(0, 1) << scrollbarStyle.scrollBg;
66
67      screen.HorizontalLine(scrollbarStyle.scrollField, GetWidth() - 2);
68
69      Uint drawOffset = 0;
70      Uint handleSize = 1;
71      if(scrollSize <= GetWidth()) {
72          handleSize = GetWidth() - 2;
73      }
74      else if(scrollSize > GetWidth() && scrollSize <= 2*GetWidth() - 3) {
75          handleSize = 2*GetWidth() - scrollSize - 2;
76          drawOffset = scrollOffset;
77      }
78      else
79          drawOffset = float(scrollOffset) /
80              float(scrollSize - GetWidth()) * (GetWidth() - 2);
81
82      screen << Control::GotoYX(0, 1 + drawOffset) << scrollbarStyle.
            scrollFg;
83      screen.HorizontalLine(scrollbarStyle.scrollHandleH, handleSize);
84
85      screen << Control::GotoYX(0, GetWidth() - 1)
86             << scrollbarStyle.button << scrollbarStyle.buttonRight;
87  }
88
89  VerticalScrollbar::VerticalScrollbar(
90      Uint _height, const ScrollbarStyle& _scrollbarStyle)throw()
91      : ScrollbarBase(_height, 1, _scrollbarStyle) { ; }
92
93  void VerticalScrollbar::OnRedraw(Screen& screen)throw()
94  {
95      screen << Control::GotoYX(0, 0)
96             << scrollbarStyle.button << scrollbarStyle.buttonUp
97             << Control::GotoYX(1, 0) << scrollbarStyle.scrollBg;
98
99      screen.VerticalLine(scrollbarStyle.scrollField, GetHeight() - 2);
100
101     Uint drawOffset = 0;
102     Uint handleSize = 1;
103     if(scrollSize <= GetHeight()) {
104         handleSize = GetHeight() - 2;
105     }
106     else if(scrollSize > GetHeight() && scrollSize <= 2*GetHeight() - 3) {
107         handleSize = 2*GetHeight() - scrollSize - 2;
108         drawOffset = scrollOffset;
109     }
110     else
111         drawOffset = float(scrollOffset) /
112             float(scrollSize - GetHeight()) * (GetHeight() - 2);
113
114     screen << Control::GotoYX(1 + drawOffset, 0) << scrollbarStyle.
            scrollFg;
115     screen.VerticalLine(scrollbarStyle.scrollHandleV, handleSize);
116
117     screen << Control::GotoYX(GetHeight()-1, 0)
118            << scrollbarStyle.button << scrollbarStyle.buttonDown;
```

```
119 }
```

4.37 lib/toolkit/src/selectbox.c++

```
  1 #include <rexio/tk/selectbox.h++>
  2 #include <rexio/tk/button.h++>
  3
  4 using namespace Scr;
  5 using namespace Scr::Tk;
  6
  7
  8 Selectbox::_SelectList::_SelectList(Uint _width, Uint _height,
  9                                     const DisplayStyle& _style)throw() :
 10     FramedWindow(_width, _height, _style),
 11     scroll(_height - 2), group(_height - 2, _width - 3, _style)
 12 {
 13     // setup this _SelectList with scroll and group to fit entities (these
          are
 14     // basic building blocks of internal _SelectList of Selectbox.
 15     AddWidget(scroll);
 16     AddWidget(group);
 17     // to trigger further setup
 18     OnResize();
 19 }
 20 void Selectbox::_SelectList::OnResize()throw()
 21 {
 22     FramedWindow::OnResize();
 23     group.SetSize(Size(GetHeight() - 2, GetWidth()-3));
 24
 25     group.SetPosition(0, 0);
 26
 27     scroll.SetPosition(0, GetWidth() - 3);
 28     scroll.SetHeight(GetHeight() - 2);
 29 }
 30 void Selectbox::_SelectList::CloseSelectList()
 31 {
 32     // _SelectList may be shown or hidden while whole Selectbox is active,
 33     // however just deleting object without passing focus to the next one
 34     // may cause SIGSEGV
 35     try {
 36         GetParent().PassFocusRequest(AllFocus);
 37         GetParent().SetActiveWidget(*prevActive);
 38         GetParent().DelWidget(*this);
 39     }
 40     catch(...){}
 41 }
 42 void Selectbox::_SelectList::OnKeyDown(Key key)throw()
 43 {
 44     if (key==Key::Up)
 45     {
 46         if (group.activeWidget!=group.elements.begin())
 47         {
 48             (*group.activeWidget)->OnUnFocus(AllFocus);
 49             --group.activeWidget;
 50             (*group.activeWidget)->OnFocus(AllFocus);
 51         }
 52     }
```

```
53     else if (key==Key::Down)
54     {
55         WidgetList::iterator tmp=group.activeWidget;
56         tmp++;
57         if (tmp!=group.elements.end())// past-the-end really
58         {
59             // mofify focus only if "next element" really exists
60             (*group.activeWidget)->OnUnFocus(AllFocus);
61             ++group.activeWidget;
62             (*group.activeWidget)->OnFocus(AllFocus);
63         }
64     }
65     else if (key==Key::Enter)
66     {
67         CloseSelectList();
68         // just close the list - selection is to be saved elsewhere
69         // and only if deliberately selected (...::Detail::Selector::
               OnAction)
70     }
71     Window::OnKeyDown(key);
72 }
73
74 void Selectbox::_SelectList::OnFocus(FocusPolicy focustype)throw()
75 {
76     if (group.activeWidget==group.elements.end())
77         group.activeWidget=group.elements.begin();
78     if (group.activeWidget!=group.elements.end())
79         (*group.activeWidget)->OnFocus(AllFocus);
80 }
81 void Selectbox::_SelectList::OnUnFocus(FocusPolicy focustype)throw()
82 {
83 }
84
85 Selectbox::_SelectList::~_SelectList()throw()
86 {;}
87
88
89 Selectbox::Selectbox(Uint _width,
90                      const DisplayStyle& _style,
91                      const DisplayStyle& _activeStyle,
92                      const SelectboxStyle& _selectBoxStyle)throw()
93     : ActiveWidget(1, _width, _style, _activeStyle),
94       selectboxStyle(_selectBoxStyle), selectList(4, _width, _style)
95 {
96     SetMaxHeight(1);
97     SetMinHeight(1);
98 }
99
100 Selectbox::Selectbox(const DisplayStyle& _style,
101                      const DisplayStyle& _activeStyle,
102                      const SelectboxStyle& _selectBoxStyle)throw()
103     : ActiveWidget(_style, _activeStyle),
104       selectboxStyle(_selectBoxStyle), selectList(4, 10, _style)
105 {
106     SetMaxHeight(1);
107     SetMinHeight(1);
108 }
109
110 namespace Scr{namespace Tk{namespace Detail
111 {
112 // Selector is part of _SelectList group , that makes Selectbox running.
113 class Selector:public Button
```

```
114 {
115 private:
116     Selectbox & SB;
117 public:
118
119     Selector(Selectbox & _SB,
120              const std::string& _label,
121              const DisplayStyle& _style = BUTTON_DEFAULT_STYLE,
122              const DisplayStyle& _activeStyle
123              = BUTTON_DEFAULT_ACTIVESTYLE)
124         :Button(_label, _style,
125                 _activeStyle), SB(_SB) {}
126     virtual void OnAction()throw()
127         {
128             try
129             {
130                 GetParent().GetRootWindow().DelWidget(SB.selectList);
131                 SB.opened=false;
132             }catch(...){}
133         }
134 };}}}
135
136 Uint Selectbox::AddOption(const std::string& name)throw()
137 {
138     selectList.group.AddWidget(*(new Detail::Selector(*this,name)));
139     return 0;
140 }
141
142 const std::string& Selectbox::GetOption() const throw(NoSuchOption)
143 {
144     try
145     {
146         const std::string& l =
147         dynamic_cast<Button&>
148             (selectList.group.GetActiveWidget()).GetLabel();
149         return l;
150     }
151     catch(Window::WidgetNotPresent)
152     {
153         THROW(NoSuchOption);
154     }
155 }
156
157 void Selectbox::OnAction()throw()
158 {
159     if(!opened) {
160         opened=true;
161         GetParent().GetRootWindow().AddWidget(selectList);
162         selectList.SetWidth(GetWidth());
163         selectList.SetPosition(
164             GetParent().GetAbsoluteRow() + GetRow(),
165             GetParent().GetAbsoluteColumn() + GetCol()
166             );
167         selectList.prevActive =
168             &(GetParent().GetRootWindow().GetActiveWidget());
169         GetParent().GetRootWindow().SetActiveWidget(selectList);
170     }
171 }
172
173 void Selectbox::OnRedraw(Screen& screen)throw()
174 {
175     screen << Control::GotoYX(0, 0) <<
```

```
176         (GetActive()?GetActiveStyle():GetStyle()) <<
177         Control::Clear;
178
179     try
180     {
181         // print out currently active option
182         screen << GetOption();
183     }
184     catch (...) {}
185     screen << Control::GotoYX(0, GetWidth() - 2) <<
186         selectboxStyle.openStyle << selectboxStyle.openButton;
187 }
188
189 void Selectbox::OnFocus(FocusPolicy focusPolicy)throw()
190 {
191     try
192     {
193         GetParent().GetRootWindow().DelWidget(selectList);
194         opened=false;
195     }catch(...){}
196     ActiveWidget::OnFocus(focusPolicy);
197 }
198
199 void Selectbox::OnUnFocus(FocusPolicy focusPolicy)throw()
200 {
201     ActiveWidget::OnUnFocus(focusPolicy);
202 }
```

4.38 lib/toolkit/src/stylesheet.c++

```
 1 #include "stylesheet.h++"
 2 #include "widget.h++"
 3 #include "../../screen/include/utf8.h++"
 4 #include <cstring>
 5 #include <cctype>
 6 #include<cstdlib>
 7 using namespace Scr;
 8 using namespace Scr::Tk;
 9
10 const Stylesheet::Property&
11 Stylesheet::Properties::operator[](const std::string &propertyName)
12     throw(NoSuchProperty)
13 {
14     Property *tmp = properties[propertyName];
15     if(!tmp) {
16         THROW(NoSuchProperty);
17     }
18     return *tmp;
19 }
20
21 void Stylesheet::Properties::SetProperty(const std::string& propertyName,
22                                          const Property& property)throw()
23 {
24     Property *tmp = properties[propertyName];
25     if(!tmp) {
26         tmp = new Property(property);
27         properties[propertyName] = tmp;
```

```
28      }
29      else
30          *tmp = property;
31  }
32
33  Stylesheet::Properties::~Properties() {
34      PropertyMap::iterator i = properties.begin();
35      while(i != properties.end()) {
36          delete (*i).second;
37          i++;
38      }
39  }
40
41  static std::string num2str(Uint num)throw()
42  {
43      std::stringstream strstream;
44      strstream << num;
45      return strstream.str();
46  }
47
48  Stylesheet::Property
49  Stylesheet::ParseValue(const std::string& valuestr)
50      throw(BadValue, Screen::InvalidUTF8)
51  {
52      bool trimwhite = true;
53      for(Uint i = 0;i< valuestr.length();i++) {
54          if(trimwhite && isspace(valuestr[i]))
55              continue;
56          if(valuestr[i] == '"') {
57              for(int j = valuestr.length();j--;) {
58                  if(valuestr[j] == '"') {
59                      if((Uint)j == i)
60                          THROW(BadValue);
61                      return Property(valuestr.substr(i+1, j - i - 1));
62                  }
63              }
64          }
65          if(valuestr[i] == '\'') {
66              i++;
67              const char *str = (valuestr.c_str()) + i;
68              Uint len = CharLengthUTF8(str);
69              if(valuestr[i + len] != '\'')
70                  THROW(BadValue);
71
72              return Property(DecodeUTF8(&str));
73          }
74          if(isdigit(valuestr[i]) || valuestr[i] == '-') {
75              int number;
76              sscanf(valuestr.c_str() + i, "%i", &number);
77              return Property(static_cast<Uint32>(atoi(valuestr.c_str())));
78          }
79          else if(isalpha(valuestr[i])) {
80
81              Fg::Color fg;
82              Fg::Style style;
83              Bg::Color bg;
84
85  #define COLOR(name, target, ns) \
86  (strncasecmp(valuestr.c_str() + i, #name, strlen(#name)) == 0) \
87      target = ns::name, i+=strlen(#name)
88              if COLOR(System, fg, Fg);
89              else if COLOR(Transparent, fg, Fg);
```

```
90              else if COLOR(Black, fg, Fg);
91              else if COLOR(Red, fg, Fg);
92              else if COLOR(Green, fg, Fg);
93              else if COLOR(Yellow, fg, Fg);
94              else if COLOR(Blue, fg, Fg);
95              else if COLOR(Magenta, fg, Fg);
96              else if COLOR(Cyan, fg, Fg);
97              else if COLOR(White, fg, Fg);
98              else THROW(BadValue);
99
100             if(i++ == valuestr.length())
101                THROW(BadValue);
102
103             if COLOR(Bright, style, Fg);
104             else if COLOR(Dark, style, Fg);
105             else THROW(BadValue);
106
107             if(i++ == valuestr.length())
108                 THROW(BadValue);
109
110             if COLOR(System, bg, Bg);
111             else if COLOR(Transparent, bg, Bg);
112             else if COLOR(Black, bg, Bg);
113             else if COLOR(Red, bg, Bg);
114             else if COLOR(Green, bg, Bg);
115             else if COLOR(Yellow, bg, Bg);
116             else if COLOR(Blue, bg, Bg);
117             else if COLOR(Magenta, bg, Bg);
118             else if COLOR(Cyan, bg, Bg);
119             else if COLOR(White, bg, Bg);
120             else THROW(BadValue);
121
122             return Property(DisplayStyle(fg, style, bg));
123         }
124     }
125     THROW(BadValue);
126 }
127
128 Stylesheet::Stylesheet(std::istream &ss)
129     throw(ParsingError, Screen::InvalidUTF8)
130 {
131 //  std::istringstream ss(str);
132
133     Uint linecnt = 0;
134     std::string line;
135
136     typedef enum {Out, ReadClass, WaitBlock, ReadBlock, Comment}
            ParseState;
137     ParseState state = Out;
138     ParseState prevstate;
139     typedef enum {WaitProperty, ReadProperty, WaitValue, ReadValue}
            BlockState;
140     BlockState substate = WaitProperty;
141     BlockState prevsubstate;
142
143
144 #define COMMENT_CHECK \
145 if(line[i] == '/') { \
146     i++; \
147     if(i < line.length()) { \
148         if(line[i] == '/') { \
149             goto endofline; \
```

```
150         } \
151         else if(line[i] == '*') { \
152             i++; \
153             prevstate = state; \
154             prevsubstate = substate; \
155             state = Comment; \
156             goto reread; \
157         } \
158         else \
159             THROWP(UnexpectedCharacter, \
160                    ", " + num2str(linecnt) + " " + num2str(i)); \
161     } \
162     i--; \
163 }
164
165     std::string className;
166     std::string propertyName;
167     std::string propertyValue;
168     bool colonfound = false;
169
170     while (std::getline(ss, line)) {
171         Uint i;
172         linecnt++;
173         for(i = 0;i<line.length();i++) {
174         reread:
175
176             if(state == Out) {
177                 for(;i<line.length();i++) {
178                     if(isspace(line[i]))
179                         continue;
180                     else if(isalnum(line[i]) || line[i] == '_') {
181                         state = ReadClass;
182                         goto reread;
183                     }
184                     else
185                         COMMENT_CHECK;
186                     THROWP(UnexpectedCharacter,
187                     ", " + num2str(linecnt) + ":" + num2str(i)
188                         + " Unexpected character in target specifier.");
189                 }
190             }
191             else if(state == ReadClass) {
192                 for(;i<line.length();i++) {
193                     if(isalnum(line[i]) || line[i] == '_' || line[i] == '#
                        '){
194                         className += line[i];
195                         continue;
196                     }
197                     else if(isspace(line[i])) {
198                         state = WaitBlock;
199                         goto reread;
200                     }
201                     else if(line[i] == '{') {
202                         state = ReadBlock;
203                         i++;
204                         goto reread;
205                     }
206                     else
207                         COMMENT_CHECK;
208                     THROWP(UnexpectedCharacter,
209                            ", " + num2str(linecnt) + ":" + num2str(i)
```

```
210                         + " Unexpected character in target specifier.")
                                ;
211                 }
212             }
213             else if(state == WaitBlock) {
214                 for(;i<line.length();i++) {
215 
216                     if(isspace(line[i]))
217                         continue;
218                     else if(line[i] == '{') {
219                         state = ReadBlock;
220                         i++;
221                         goto reread;
222                     }
223                     else
224                         COMMENT_CHECK;
225                     THROWP(UnexpectedCharacter,
226                            ", " + num2str(linecnt) + ":" + num2str(i)
227                            + " Unexpected character after target specifier
                                .");
228                 }
229             }
230             else if(state == ReadBlock) {
231                 bool inQuote = false;
232 
233                 for(;i<line.length();i++) {
234 
235                     if(!inQuote)
236                         COMMENT_CHECK;
237 
238                     if(line[i] == '}') {
239                         className = "";
240                         i++;
241                         state = Out;
242                         if(substate != WaitProperty)
243                             THROWP(UnexpectedCharacter,
244                                    ", " + num2str(linecnt) + ":" + num2str
                                        (i)
245                                    + " Unexpected end of block(forgot a
                                        semicolon?).");
246                         colonfound = false;
247                         propertyName = "";
248                         propertyValue = "";
249                         goto reread;
250                     }
251                     if(substate == WaitProperty) {
252                         if(isspace(line[i]))
253                             continue;
254                         else if(isalnum(line[i]) || line[i] == '_')
255                             substate = ReadProperty;
256                         else
257                             THROWP(UnexpectedCharacter,
258                                    ", " + num2str(linecnt) + ":" + num2str
                                        (i)
259                                    + " Unexpected character while
                                        expecting property.");
260 
261                     }
262                     if(substate == ReadProperty) {
263                         if(isalnum(line[i]) || line[i] == '_')
264                             propertyName += line[i];
265                         else if(isspace(line[i]) || line[i] == ':') {
```

```
266                         substate = WaitValue;
267                         colonfound = false;
268                     }
269                     else
270                         THROWP(UnexpectedCharacter,
271                                ", " + num2str(linecnt) + ":" + num2str
                                       (i)
272                                + " Unexpected character while reading
                                       property.");
273                 }
274                 if(substate == WaitValue) {
275                     if(isspace(line[i]))
276                         continue;
277                     else if(line[i] == ':' && !colonfound)
278                         colonfound = true;
279                     else if(colonfound &&
280                             isprint(line[i]))
281                         substate = ReadValue;
282                     else
283                         THROWP(UnexpectedCharacter,
284                                ", " + num2str(linecnt) + ":" + num2str
                                       (i)
285                                + " Unexpected character while
                                       expecting value.");
286                 }
287                 if(substate == ReadValue) {
288                     if(line[i] == '"')
289                         inQuote = !inQuote;
290
291                     if(line[i] == ';' && !inQuote) {
292                         substate = WaitProperty;
293
294                         SetProperty(className, propertyName,
295                                     ParseValue(propertyValue));
296
297                         propertyName = "";
298                         propertyValue = "";
299                         colonfound = false;
300                         inQuote = false;
301                     }
302                     else
303                         propertyValue += line[i];
304                 }
305
306             }
307         }
308         else if(state == Comment) {
309             for(;i<line.length();i++) {
310                 if(line[i] == '*') {
311                     i++;
312                     if(i < line.length()) {
313                         if(line[i] == '/') {
314                             state = prevstate;
315                             substate = prevsubstate;
316                             i++;
317                             goto reread;
318                         }
319                     }
320                 }
321             }
322         }
323     }
```

```
324     endofline:;
325     }
326     if(state != Out)
327         THROW(UnexpectedEndOfSheet);
328 }
329
330 const Stylesheet::Property& Stylesheet::GetProperty(const Widget& w,
331                                                     const std::string& property)
332     throw(Properties::NoSuchProperty)
333 {
334     Properties* tmp;
335
336     const Widget::ClassHierarchy &cs = const_cast<Widget &>(w).Hierarchy()
            ;
337
338     // walk through the whole class hierarchy
339     for(Uint i = 0;i < cs.size();i++) {
340         if(w.objectName != "") {
341             tmp = classes[cs[i] + "#" + w.objectName];
342             if(tmp) {
343                 try {
344                     return (*tmp)[property];
345                 }
346                 catch(Properties::NoSuchProperty) {
347                     ;
348                 }
349             }
350         }
351         tmp = classes[cs[i]];
352         if(tmp) {
353             try {
354                 return (*tmp)[property];
355             }
356             catch(Properties::NoSuchProperty) {
357                 continue;
358             }
359         }
360     }
361     THROW(Properties::NoSuchProperty);
362 }
363
364 void Stylesheet::SetProperty(const std::string& className,
365                              const std::string& property,
366                              const Property& value) throw()
367 {
368     Properties *tmp = classes[className];
369     if(!tmp) {
370         tmp = new Properties();
371         classes[className] = tmp;
372
373     }
374     tmp->SetProperty(property, value);
375 }
376
377 Stylesheet::~Stylesheet() {
378     ClassMap::iterator i = classes.begin();
379     while(i != classes.end()) {
380         delete (*i).second;
381         i++;
382     }
383 }
```

4.39 lib/toolkit/src/verticalgroup.c++

```
#include <iostream>
#include <cmath>
#include "verticalgroup.h++"

using namespace Scr;
using namespace Scr::Tk;

void VerticalGroup::ArrangeContents()throw()
{
    Uint maxheight = GetHeight();
    Uint maxwidth = GetWidth();

    float totalmax = 0;
    Uint totalmin = 0;
    Uint coefsum = 0;

    Uint freestep = 0;

    bool stretchmax = false;

    Uint visible_elems =  elements.size();
    for(WidgetList::iterator i = elements.begin();
        i != elements.end(); i++) {

        if((*i)->IsHidden()) {
            visible_elems--;
            continue;
        }

        if((*i)->GetMaxHeight() == UintMax) // means, stretch to max
            stretchmax = true;
        else
            totalmax += (*i)->GetMaxHeight();
        // sum of coefficients will give some hint in dividing free
        // space
        coefsum += elementsLayout[*i].stretchFactor;

        totalmin += (*i)->GetMinHeight();
    }
    bool usecoef = false;
    if(totalmax > maxheight || stretchmax) {
        totalmax = maxheight; // use whole available space
        usecoef = true;
    }

    Uint addpoint = 0; // point at from which the first widget will be
        drawn
    switch(alignPolicy) {
    case Distribute:
    case Begin:
        addpoint = 0;
        break;
    case Center:
        /* NOTE: (GetHeight() - totalmax)/2 can result in overflow first
           during substraction and the division would divide the overflown
           value. More desired behaviour is achieved by the below
               operation.
        */
        addpoint = GetHeight()/2 - totalmax/2;
        break;
```

```
59      case End:
60          addpoint = GetHeight() - totalmax;
61          break;
62      }
63
64      if(usecoef) { // base on coefficients space division
65
66          totalmax = maxheight;
67
68          if(totalmin <= totalmax)
69              totalmax -= totalmin; // totalmax will be now a height that
70          // has to be distributed among elements
71          else
72              totalmax = 0;
73
74          for(WidgetList::iterator i = elements.begin();
75              i != elements.end(); i++) {
76
77              if((*i)->IsHidden())
78                  continue;
79
80              Widget &w = **i;
81              Uint coef = elementsLayout[&w].stretchFactor;
82
83              Uint distspace =roundf(((float)coef/(float)coefsum) * totalmax
                    );
84
85              if(w.GetMinHeight() >= distspace)
86                  distspace = 0;
87              else
88                  distspace -= w.GetMinHeight();
89
90              if(distspace + w.GetMinHeight() > w.GetMaxHeight())
91                  distspace = w.GetMaxHeight() - w.GetMinHeight();
92
93              // height = distributed space + minimal height of this widget
94              w.SetSize(distspace + w.GetMinHeight(), maxwidth);
95
96              totalmax -= distspace;
97              coefsum -= coef;
98          }
99          if(totalmax && visible_elems) { // distribute anything left to the
100             // first visible element that can aquire any more size
101             Widget *w;
102             for(WidgetList::iterator i = elements.begin();
103                 i != elements.end(); i++) {
104                 w = *i;
105                 if(w->IsHidden() ||
106                    w->GetHeight() + totalmax > w->GetMaxHeight())
107                     continue;
108                 break;
109             }
110             w->SetSize(w->GetHeight() + totalmax, w->GetWidth());
111         }
112     }
113     else { // base on align model
114         if(alignPolicy == Distribute)
115             freestep = (maxheight - totalmax)/elements.size();
116
117         Uint tmp = addpoint;
118         for(WidgetList::iterator i = elements.begin();
119             i != elements.end(); i++) {
```

```
120             if((*i)->IsHidden())
121                 continue;
122             Widget &w = **i;
123 
124             tmp += w.GetMaxHeight();
125             w.SetSize(w.GetMaxHeight(), maxwidth);
126         }
127     }
128 
129     // finally position all the elements,stacking the heights
130     for(WidgetList::iterator i = elements.begin();
131         i != elements.end(); i++) {
132             if((*i)->IsHidden())
133                 continue;
134         Widget &w = **i;
135 
136         w.SetPosition(addpoint, 0);
137         addpoint += w.GetHeight() + freestep/*from the distributed model*/
                ;
138     }
139 
140 }
141 
142 VerticalGroup::VerticalGroup(Uint _height,
143                              Uint _width,
144                              const DisplayStyle &  _style)throw()
145     :BoxGroup(_height, _width,_style)
146 {
147     ;
148 }
149 
150 VerticalGroup::VerticalGroup(const WidgetGroup & base)throw()
151     :BoxGroup(base)
152 {;}
153 
154 VerticalGroup::~VerticalGroup()throw() {;}
```

4.40 lib/toolkit/src/widget.c++

```
 1 #include <rexio/screen.h++>
 2 
 3 #include <rexio/tk/widget.h++>
 4 #include <rexio/tk/window.h++>
 5 #include <rexio/throw.h++>
 6 
 7 using namespace Scr;
 8 using namespace Scr::Tk;
 9 
10 Widget::Widget(Uint _height,
11                Uint _width,
12                const DisplayStyle & _style)throw()
13     :parentWindow(NULL),
14      styleSheet(NULL),
15      position(0, 0),
16      size(_height, _width),
17      sizeMax(UintMax, UintMax),
18      sizeMin(0, 0),
```

```
19      style(_style),
20      hidden(false) {;}
21
22 Widget::Widget( const DisplayStyle & _style)throw()
23     :parentWindow(NULL),
24      styleSheet(NULL),
25      position(0, 0),
26      size(0, 0),
27      sizeMax(UintMax, UintMax),
28      sizeMin(0, 0),
29      style(_style),
30      hidden(false) {;}
31
32 void Widget::SetParent(Window& window)throw(ParentAlreadySet)
33 {
34     if(parentWindow)
35         THROW(ParentAlreadySet);
36     parentWindow = &window;
37 }
38
39 Window& Widget::GetParent()throw(ParentNotDefined)
40 {
41     if(!(this->parentWindow))
42         THROW(ParentNotDefined);
43     return *parentWindow;
44 }
45
46 void Widget::ReParent(Window *window)throw()
47 {
48     parentWindow = window;
49 }
50
51 void Widget::SetStylesheet(Stylesheet* _styleSheet)throw()
52 {
53     styleSheet = _styleSheet;
54     __FetchProperty(style, "style");
55 //FIXME when changed to Bg::Black, weird things happen
56 }
57
58 void Widget::OnFocus(FocusPolicy focustype)throw()
59 {
60     // default behaviour is allowing focus?
61     parentWindow->PassFocusRequest(focustype);
62 }//element unfocused
63
64 void Widget::OnUnFocus(FocusPolicy focustype)throw(){;}//element focused
65 void Widget::OnStart()throw() {;}
66 void Widget::OnRedraw(Screen&screen)throw() {;}
67 void Widget::RedrawRequest()throw()
68 {
69     try
70     {
71         GetParent().RedrawRequest(*this);
72     }
73     catch(ParentNotDefined)
74     {
75         ;
76     }
77 }
78
79 void Widget::OnResize()throw() {;}
80 void Widget::OnKeyDown(Key key)throw() {
```

```
 81     if(key.IsASpecialKey()) {
 82         if(key == Key::Tab) {
 83             GetParent().PassFocusRequest(TabFocus);
 84         }
 85
 86     }
 87 }
 88
 89 void Widget::OnExit()throw() {;}
 90
 91 void Widget::SetPosition(const Position& _pos)throw(ParentNotDefined)
 92 {
 93     GetParent(); // throws exception
 94     position = _pos;
 95 }
 96
 97 void Widget::SetPosition(Uint _row, Uint _col)throw(ParentNotDefined)
 98 {
 99     SetPosition(Position(_row, _col));
100 }
101
102 Position Widget::GetPosition() const throw(ParentNotDefined)
103 {
104     return position;
105 }
106
107 void Widget::SetRow(Uint _row)throw(ParentNotDefined)
108 {
109     SetPosition(_row, position.col);
110 }
111
112 Uint Widget::GetRow() const throw(ParentNotDefined)
113 {
114     return position.row;
115 }
116
117 void Widget::SetCol(Uint _col)throw(ParentNotDefined)
118 {
119     SetPosition(position.row, _col);
120 }
121
122 Uint Widget::GetCol() const throw(ParentNotDefined)
123 {
124     return position.col;
125 }
126
127 void Widget::SetSize(const Size& _size)throw()
128 {
129     size = _size;
130     OnResize();
131 }
132
133 void Widget::SetSize(Uint _height, Uint _width)throw()
134 {
135     SetSize(Size(_height, _width));
136 }
137
138 const Size& Widget::GetSize() const throw()
139 {
140     return size;
141 }
142
```

```
143 void Widget::SetHeight(Uint _height)throw()
144 {
145     SetSize(_height, size.width);
146 }
147
148 Uint Widget::GetHeight() const throw()
149 {
150     return size.height;
151 }
152
153 void Widget::SetWidth(Uint _width)throw()
154 {
155     SetSize(size.height, _width);
156 }
157
158 Uint Widget::GetWidth() const throw()
159 {
160     return size.width;
161 }
162
163 void Widget::SetGeometry(const Position& _pos, const Size& _size)
164     throw(ParentNotDefined)
165 {
166     SetPosition(_pos);
167     SetSize(_size);
168 }
169
170 void Widget::SetGeometry(Uint _row, Uint _col,
171                          Uint _height, Uint _width)throw(ParentNotDefined)
172 {
173     SetGeometry(Position(_row, _col), Size(_height, _width));
174 }
175
176 void Widget::SetMinSize(const Size& _size)throw()
177 {
178     sizeMin = _size;
179 }
180
181 void Widget::SetMinSize(Uint _height, Uint _width)throw()
182 {
183     SetMinSize(Size(_height, _width));
184 }
185
186 const Size& Widget::GetMinSize() const throw()
187 {
188     return sizeMin;
189 }
190
191 void Widget::SetMinHeight(Uint _height)throw()
192 {
193     SetMinSize(_height, sizeMin.width);
194 }
195
196 Uint Widget::GetMinHeight() const throw()
197 {
198     return sizeMin.height;
199 }
200
201 void Widget::SetMinWidth(Uint _width)throw()
202 {
203     SetMinSize(sizeMin.height, _width);
204 }
```

```
205
206 Uint Widget::GetMinWidth() const throw()
207 {
208     return sizeMin.width;
209 }
210
211 void Widget::SetMaxSize(const Size& _size)throw()
212 {
213     sizeMax = _size;
214 }
215
216 void Widget::SetMaxSize(Uint _height, Uint _width)throw()
217 {
218     SetMaxSize(Size(_height, _width));
219 }
220
221 const Size& Widget::GetMaxSize() const throw()
222 {
223     return sizeMax;
224 }
225
226 void Widget::SetMaxHeight(Uint _height)throw()
227 {
228     SetMaxSize(_height, sizeMax.width);
229 }
230
231 Uint Widget::GetMaxHeight() const throw()
232 {
233     return sizeMax.height;
234 }
235
236 void Widget::SetMaxWidth(Uint _width)throw()
237 {
238     SetMaxSize(sizeMax.height, _width);
239 }
240
241 Uint Widget::GetMaxWidth() const throw()
242 {
243     return sizeMax.width;
244 }
245
246 void Widget::SetFocusPolicy(FocusPolicy _policy)throw()
247 {
248     focusPolicy = _policy;
249 }
250
251 Widget::FocusPolicy Widget::GetFocusPolicy() const throw()
252 {
253     return focusPolicy;
254 }
255
256 void Widget::SetStyle(const DisplayStyle& _style)throw()
257 {
258     style = _style;
259 }
260
261 const DisplayStyle& Widget::GetStyle() const throw()
262 {
263     return style;
264 }
265
266 void Widget::SetHidden(bool _hidden)throw()
```

```
267 {
268     hidden = _hidden;
269     RedrawRequest();
270 }
271
272 bool Widget::IsHidden() const throw()
273 {
274     return hidden;
275 }
276
277 Widget::~Widget()throw() {;}
```

4.41 lib/toolkit/src/widgetgroup.c++

```
 1 #include <iostream>
 2 #include "widgetgroup.h++"
 3
 4 using namespace Scr;
 5 using namespace Scr::Tk;
 6
 7 WidgetGroup::WidgetGroup(Uint _height,
 8                          Uint _width,
 9                          const DisplayStyle &  _style)throw()
10     :Window(_height,_width,_style) {;}
11
12 WidgetGroup::WidgetGroup(const WidgetGroup & base)throw()
13     : Window(base.size.height,base.size.width,base.style)
14 {
15     ;
16 }
17
18 WidgetGroup::~WidgetGroup()throw(){;}
19 void WidgetGroup::SwapWidgets(Widget& widget1, Widget& widget2)throw()
20 {
21     // swap positions - in case of really rumb WidgetGroup :)
22     Position pos = widget1.GetPosition();
23     widget1.SetPosition(widget2.GetPosition());
24     widget2.SetPosition(pos);
25
26     // swap on the list
27     elements.swap(&widget1, &widget2);
28
29     ArrangeContents();
30 }
31
32 void WidgetGroup::ArrangeContents()throw()
33 {
34     ;
35 }
36
37 // fugly, but works :)
38 void WidgetGroup::ShiftFWidget(Widget& widget)throw()
39 {
40     WidgetList::iterator i = elements[&widget];
41     WidgetList::iterator end = elements.end();
42     if(i != (--end))
43         SwapWidgets(widget, **(++i));
```

```
44     else
45         SwapWidgets(widget, **(elements.begin()));
46 }
47
48 // fugly, but works :)
49 void WidgetGroup::ShiftBWidget(Widget& widget)throw()
50 {
51     WidgetList::iterator i = elements[&widget];
52     WidgetList::iterator end = elements.end();
53     if(i != elements.begin())
54         SwapWidgets(widget, **(--i));
55     else
56         SwapWidgets(widget, **(--end));
57 }
```

4.42 lib/toolkit/src/window.c++

```
1 #include "window.h++"
2
3 using namespace Scr;
4 using namespace Scr::Tk;
5
6 Window::Window(Uint _height,
7                Uint _width,
8                const DisplayStyle &  _style)throw()
9     :Widget(_height, _width, _style)
10 {
11     activeWidget = elements.end();
12 }
13
14 Screen& Window::GetScreen()throw(ParentNotDefined)
15 {
16     return GetParent().GetScreen();
17 }
18
19 Uint Window::GetAbsoluteColumn()throw(ParentNotDefined)
20 {
21     return GetParent().GetAbsoluteColumn() + position.col;
22 }
23
24 Uint Window::GetAbsoluteRow()throw(ParentNotDefined)
25 {
26     return GetParent().GetAbsoluteRow() + position.row;
27 }
28
29 void Window::SetStylesheet(Stylesheet* _styleSheet)throw()
30 {
31     Widget::SetStylesheet(_styleSheet);
32     for(WidgetList::iterator i=elements.begin(); i!=elements.end();++i)
33         (*i)->SetStylesheet(_styleSheet);
34 }
35
36 void Window::AddWidget(Widget& widget)
37     throw(ParentAlreadySet, WidgetAlreadyAdded)
38 {
39     if(elements[&widget] != elements.end())
40         THROW(WidgetAlreadyAdded);
```

```
41
42      widget.SetParent(*this);
43      if(styleSheet)
44          widget.SetStylesheet(styleSheet);
45      elements.push_back(&widget);
46      // each widget must recv. OnStart. activeWidget is set after
47      // OnStart, so when it is set, certainly Window::OnStart() was
48      // already called!
49      if(activeWidget != elements.end())
50          widget.OnStart();
51  }
52
53  void Window::DelWidget(Widget& widget)
54      throw(WidgetNotPresent)
55  {
56      if (&widget==*activeWidget)
57          PassFocusRequest(TabFocus);
58      if(elements[&widget] == elements.end())
59          THROW(WidgetNotPresent);
60      elements.remove(&widget);
61      widget.ReParent(NULL);
62  }
63
64  RootWindow& Window::GetRootWindow()throw(ParentNotDefined)
65  {
66      return GetParent().GetRootWindow();
67  }
68
69  void Window::RedrawRequest(Widget& widget)throw()
70  {
71      if(IsHidden())
72          return;
73
74      if (parentWindow != this) // this is not a root win
75      {
76          GetParent().RedrawRequest(*this);
77      }
78      else
79      {
80          Screen & screen=GetScreen();// main screen
81
82
83          WidgetList::iterator i;
84          for (i=elements[&widget]; i!=elements.end ();++i)
85          {
86              if((*i)->IsHidden ())
87                  continue;
88
89              // working subscreen
90              Screen * ss;
91              try
92              {
93                  ss = screen.CreateSubScreen ((*i)->GetRow (), (*i)->GetCol
                        (),
94                             (*i)->GetHeight (), (*i)->GetWidth ());
95              }
96              catch (Scr::Screen::SubscreenOutOfRange)
97              {
98                  continue;
99              }
100             (*i)->OnRedraw (*ss);
101             delete ss;
```

```
102         }
103
104         screen.Refresh();
105
106     }
107 }
108
109 void Window::RedrawRequest()throw()
110 {
111     if (! elements.empty())
112     RedrawRequest(**(elements.begin()));
113 }
114
115 void Window::PassFocusRequest(FocusPolicy focustype)throw()
116 {
117     if (!elements.empty()) {
118         if(activeWidget != elements.end()) {
119             (*activeWidget)->OnUnFocus(focustype);
120         }
121
122         ++activeWidget;
123
124         if (activeWidget == elements.end()) {
125             if(focustype == TabFocus) {
126                 parentWindow->PassFocusRequest(focustype);
127                 return;
128             }
129             activeWidget = elements.end();
130             return;
131         }
132
133         if((*activeWidget)->IsHidden ())
134             PassFocusRequest(focustype);
135         else
136             (*activeWidget)->OnFocus(focustype);
137     }
138 }
139
140 void Window::SetActiveWidget(Widget &w)
141     throw(WidgetNotPresent)
142 {
143     if(activeWidget != elements.end())
144     {
145         if (&w==*activeWidget)
146             return;
147         (*activeWidget)->OnUnFocus(AllFocus);
148     }
149     activeWidget = elements[&w];
150     if(activeWidget == elements.end())
151         THROW(WidgetNotPresent);
152     (*activeWidget)->OnFocus(AllFocus);
153 }
154
155 Widget& Window::GetActiveWidget()const throw(WidgetNotPresent)
156 {
157     if(activeWidget == elements.end())
158        THROW(WidgetNotPresent);
159     return **activeWidget;
160 }
161
162 void Window::OnFocus(FocusPolicy focustype)throw()
163 {
```

```
164     if(activeWidget == elements.end()) {
165         activeWidget = elements.begin();
166         if(activeWidget != elements.end())
167             (*activeWidget)->OnFocus(focustype);
168         else
169             parentWindow->PassFocusRequest(focustype);
170     }
171     else {
172         (*activeWidget)->OnFocus(focustype);
173     }
174 }
175
176 void Window::OnUnFocus(FocusPolicy focustype)throw()
177 {
178     if(activeWidget != elements.end()) {
179         (*activeWidget)->OnUnFocus(focustype);
180     }
181 }
182
183 void Window::OnStart()throw()
184 {
185     WidgetList::iterator i;
186     for (i=elements.begin(); i!=elements.end();++i)
187         (*i)->OnStart();
188     activeWidget=elements.begin();
189 }
190
191 void Window::OnResize()throw()
192 {
193     for(WidgetList::iterator i=elements.begin(); i!=elements.end();++i)
194         (*i)->OnResize();
195 }
196
197 void Window::OnRedraw(Screen& screen)throw()
198 {
199     screen << GetStyle() << Control::Clear;
200
201     WidgetList::iterator i;
202     for (i=elements.begin(); i!=elements.end();++i)
203     {
204         if((*i)->IsHidden())
205             continue;
206
207         Screen * ss;
208         try
209         {
210             ss = screen.CreateSubScreen((*i)->GetRow(), (*i)->GetCol(),
211                                         (*i)->GetHeight(), (*i)->GetWidth
                                                ());
212         }
213         catch (Scr::Screen::SubscreenOutOfRange)
214         {
215             continue;
216         }
217         (*i)->OnRedraw(*ss);
218         delete ss;
219     }
220 }
221
222 void Window::OnKeyDown(Key key)throw()
223 {
224     if (activeWidget!=elements.end()) {
```

```
225         (*activeWidget)  ->OnKeyDown(key);//pass to activeWidget element;
226     }
227 }
228
229 void Window::SetSize(const Size& _size)throw()
230 {
231     if(!parentWindow)
232         return;
233     Widget::SetSize(_size);
234 }
```