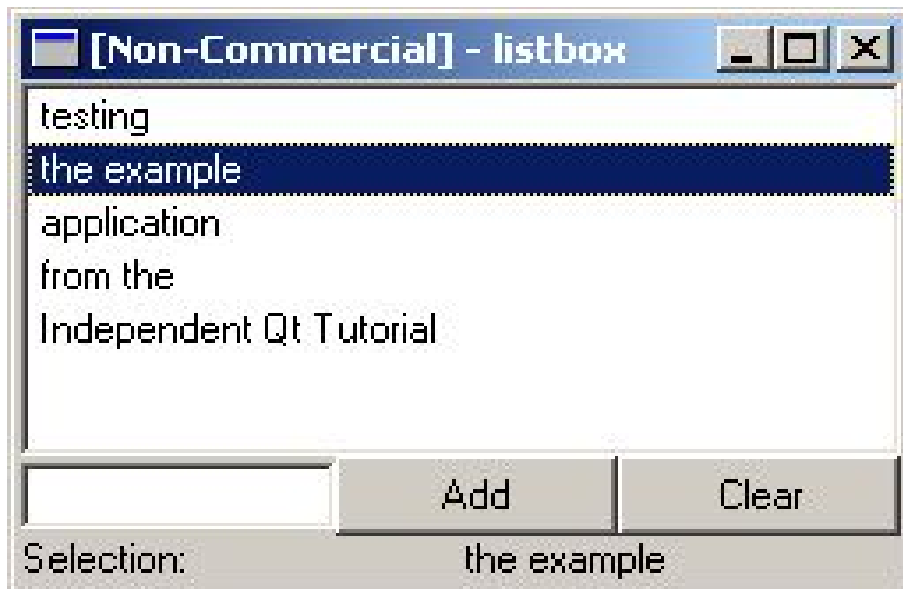


Lists, Trees and Tables

Qt technologie poskytuje zobrazit data jako jednoduchý seznam (list box), jako složitější seznam v podobě zobrazení hodnot ve sloupcích (list view), jako stromček, kde jsou hodnoty zobrazeny hierarchicky (list view) a pomocí tabulky (table).

QListBox

Objekt QListBox poskytuje zobrazení dat v nejjednodušší formě – na jednom řádku jeden údaj.



Konstrukce objektu je popsána v „QT Reference document“ nebo je nejlepší si objekt navrhnout v QR Designeru.

Pokud je konstrukce v pořádku provedena, k vyprázdnění seznamu lze použít metodu clear, pro vkládání je možné použít metody insertItem nebo insertStringList.

```
ListBoxExample::ListBoxExample( QWidget *parent, char *name ) : QVBox( parent, name )
{
    m_listBox = new QListBox( this );

    QHBoxLayout *hb = new QHBoxLayout( this );
    QLineEdit *m_lineEdit = new QLineEdit( hb );
    QPushButton *pbAdd = new QPushButton( "Add", hb );
    QPushButton *pbClear = new QPushButton( "Clear", hb );

    connect( pbAdd, SIGNAL(clicked()), this, SLOT(addItem()) );
    connect( pbClear, SIGNAL(clicked()), m_listBox, SLOT(clear()) );
}

void ListBoxExample::addItem()
{
    m_listBox->insertItem( m_lineEdit->text() );
    m_lineEdit->setText( "" );
}
```

Příklad názorně ukazuje použití metody insertItem. V konstruktoru dochází ke konstrukci objektů a v metodě addItem je přidán text z editačního pole (line edit) do seznamu (list box).

K rozšíření vlastnosti seznamu lze použít objekt `QListBoxItem`. Příkladem je objekt `QListBoxPixmap` a `QjListBoxPixmap`, pomocí kterých můžete tvořit seznam s texty a obrázky.

```
QListBox *lb = new QListBox();
QPixmap pm( 12, 12 );

pm.fill( Qt::red );
new QListBoxPixmap( lb, pm, "Red" );
pm.fill( Qt::yellow );
new QListBoxPixmap( lb, pm, "Yellow" );
pm.fill( Qt::green );
new QListBoxPixmap( lb, pm, "Green" );
pm.fill( Qt::cyan );
new QListBoxPixmap( lb, pm, "Cyan" );
pm.fill( Qt::blue );
new QListBoxPixmap( lb, pm, "Blue" );
pm.fill( Qt::magenta );
new QListBoxPixmap( lb, pm, "Magenta" );
```

Na příkladu je znázorněna konstrukce list boxu a objektu `QPixmap` (výstupní definované zařízení pro kreslení – bitová mapa). Konstrukce `QListBoxPixmap` definuje vlastníka tohoto objektu (vlastníkem je list box), bitovou mapu s obrázkem a doprovodný text. Po konstrukci je automaticky definována položka seznamu (viz. výsledný obrazek seznamu).



Na obrázku je vybrán uživatelem programu řádek v seznamu s označením green. QT technologie umožňuje pracovat se třemi módy výběru položek (selected mode):

`QListBox::Single` – lze vybrat pouze jednu položku seznamu.

`QListBox::Multi` – lze vybrat jakékoliv množství položek seznamu myší (drag and drop).

`QListBox::Extended` – lze vybrat jakékoliv množství položek seznamu známým způsobem `shift+rozsah` nebo `ctrl+položka`.

Na následujícím příkladu je ukázka zjištění textu vybrané položky. Jako selected mode je použit mód `QListBox::Single`:

```

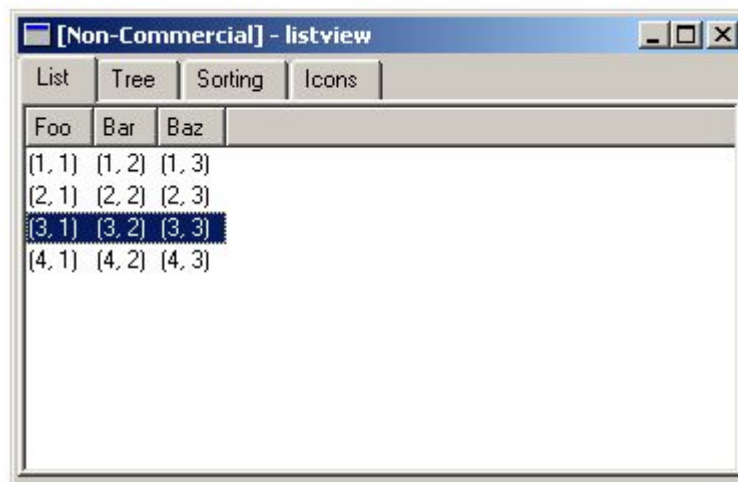
ListBoxExample::ListBoxExample( QWidget *parent, char *name ) : QVBox( parent, name )
{
    ...
    connect( m_listBox, SIGNAL(selectionChanged()), this, SLOT(newSelection()) );
}

void ListBoxExample::newSelection()
{
    if( !m_listBox->selectedItem() )
        m_label->setText( "nothing" );
    else
        m_label->setText( m_listBox->selectedItem()->text() );
}

```

QListView

Pokud potřebujeme zobrazit položku seznamu přehledněji do sloupců, použijeme technologii podporovanou objektem QListView.



Na příkladu je seznam zobrazen ve třech sloupcích. Sloupce lze tvořit návrhem v QT Designeru nebo metodou QListView::addColumn. Konstrukce objektu QListViewItem, určí vlastníka tohoto objektu (vlastník je list view) a texty pro jednotlivé sloupce. Po konstrukci je položka zobrazena v list view (viz. příklad konstrukce).

```

QWidget *ListViewExample::setupListTab()
{
    m_listView = new QListView();

    m_listView->addColumn( "Foo" );
    m_listView->addColumn( "Bar" );
    m_listView->addColumn( "Baz" );

    m_listView->setAllColumnsShowFocus( true );

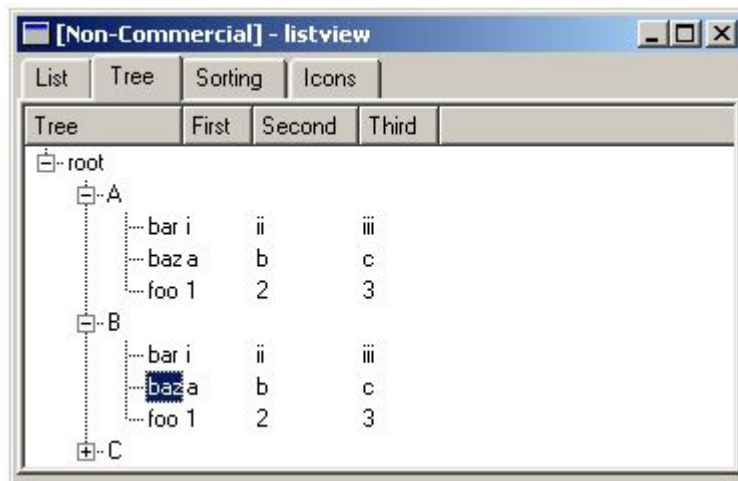
    new QListViewItem( m_listView, "(1, 1)", "(1, 2)", "(1, 3)" );
    new QListViewItem( m_listView, "(2, 1)", "(2, 2)", "(2, 3)" );
    new QListViewItem( m_listView, "(3, 1)", "(3, 2)", "(3, 3)" );
    new QListViewItem( m_listView, "(4, 1)", "(4, 2)", "(4, 3)" );

    return m_listView;
}

```

V popisu objektu QListView a QListViewItem v QT Reference document jsou popsány kompletní metody pro správu list view.

Objekt QListView poskytuje možnost zobrazit seznam i pomocí stromu hierarchicky:



Ukázka kódu přibližuje způsob práce se hierarchickým zobrazením seznamu:

```
QWidget *ListViewExample::setupTreeTab()
{
    m_treeView = new QListView();

    m_treeView->addColumn( "Tree" );
    m_treeView->addColumn( "First" );
    m_treeView->addColumn( "Second" );
    m_treeView->addColumn( "Third" );

    m_treeView->setRootIsDecorated( true );

    QListViewItem *root = new QListViewItem( m_treeView, "root" );

    QListViewItem *a = new QListViewItem( root, "A" );
    QListViewItem *b = new QListViewItem( root, "B" );
    QListViewItem *c = new QListViewItem( root, "C" );

    new QListViewItem( a, "foo", "1", "2", "3" );
    new QListViewItem( a, "bar", "i", "ii", "iii" );
    new QListViewItem( a, "baz", "a", "b", "c" );

    new QListViewItem( b, "foo", "1", "2", "3" );
    new QListViewItem( b, "bar", "i", "ii", "iii" );
    new QListViewItem( b, "baz", "a", "b", "c" );

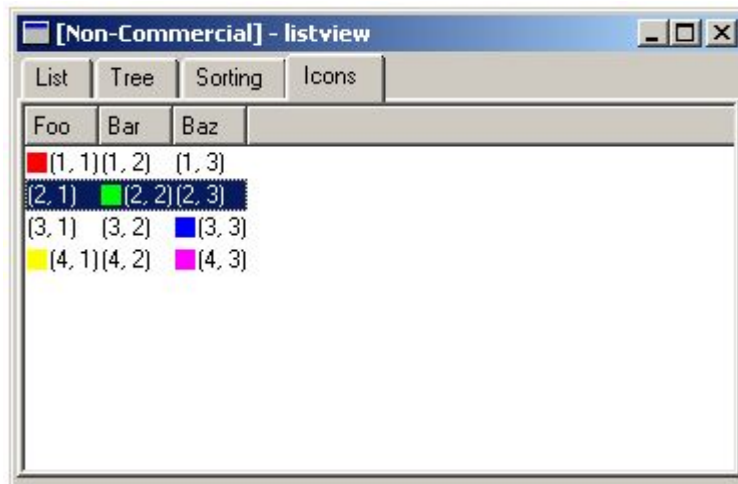
    new QListViewItem( c, "foo", "1", "2", "3" );
    new QListViewItem( c, "bar", "i", "ii", "iii" );
    new QListViewItem( c, "baz", "a", "b", "c" );

    return m_treeView;
}
```

Pomocí metody vytvoříme sloupce seznamu a jejich názvy. Hierarchický vztah je definován vlastníkem konstruovaného objektu QListViewItem. Filozofie vkládání položek do zobrazení je

stejný jako v předchozím použití QListView.

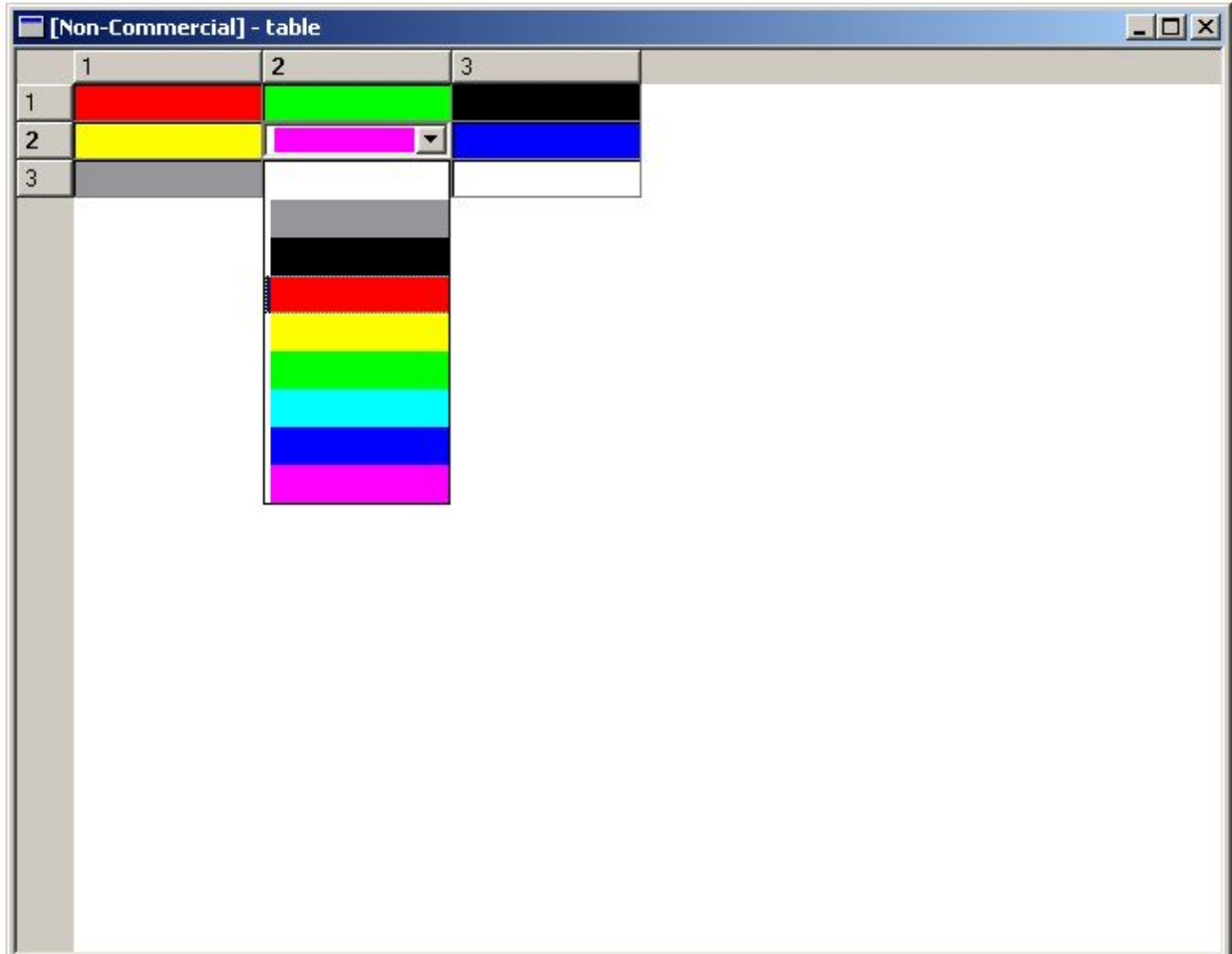
Jednoduché je i zobrazování obrázku k textu na jednotlivých sloupcích.



Ikony lze umísťovat na jakékoliv místo v seznamu (jakýkoliv řádek, jakýkoliv sloupec). Standardně plníme položky seznamu list view. Pomocí objektu QPixmap a metody QListViewItem::setPixmap můžeme vkládat ikony na jakékoliv místo seznamu.

QTable

Použitím widgetu QTable se nám dostává možnost definovat pro jakoukoliv buňku jakýkoliv widget (check box, combo box, obyčejný řetězec, ...). Je jednoduché vytvořit tabulku a pro vložení / definici příslušného widgetu použít setText, setPixmap, setItem funkce.



Na obrázku tabulka je definována objektem `QTable` a jednotlivé buňky objektem `QTableWidgetItem`. Jako buňku musíme definovat třídu odvozenou od `QTableWidgetItem`, která bude určovat buňku jako barvu, kterou určí uživatel programu pomocí combo boxu.

```
class ColorTableWidgetItem : public QTableWidgetItem
{
public:
    ColorTableWidgetItem( QTable *table, const QString &color );

    QWidget *createEditor() const;
    void setContentFromEditor( QWidget *w );

    void paint( QPainter *p, const QColorGroup &cgroup, const QRect &cr, bool selected );
};
```

Funkce `paint` vykreslí pozadí buňky při editaci dle definovaného textu ve funkci `createEditor`. Funkce `createEditor` vytvoří v buňce combo box a definuje položky combo boxu (barvu a text).

```
void ColorTableWidgetItem::paint( QPainter *p, const QColorGroup &cgroup, const QRect &cr,
bool selected )
{
    if( text() == "White" )
        p->setBrush( Qt::white );
    else if( text() == "Gray" )
        p->setBrush( Qt::gray );
    else if( text() == "Black" )
        p->setBrush( Qt::black );
    else if( text() == "Red" )
        p->setBrush( Qt::red );
    else if( text() == "Yellow" )
        p->setBrush( Qt::yellow );
    else if( text() == "Green" )
        p->setBrush( Qt::green );
    else if( text() == "Cyan" )
        p->setBrush( Qt::cyan );
    else if( text() == "Blue" )
        p->setBrush( Qt::blue );
    else if( text() == "Magenta" )
        p->setBrush( Qt::magenta );

    p->drawRect( table()->cellRect( row(), col() ) );
}
```

```

QWidget *ColorTableItem::createEditor() const
{
    QComboBox *cb = new QComboBox( table()->viewport() );
    QObject::connect( cb, SIGNAL( activated( int ) ), table(), SLOT( doValueChanged
() ) );

    QPixmap pm( 100, 20 );

    pm.fill( Qt::white );
    cb->insertItem( pm );
    pm.fill( Qt::gray );
    cb->insertItem( pm );
    pm.fill( Qt::black );
    cb->insertItem( pm );
    pm.fill( Qt::red );
    cb->insertItem( pm );
    pm.fill( Qt::yellow );
    cb->insertItem( pm );
    pm.fill( Qt::green );
    cb->insertItem( pm );
    pm.fill( Qt::cyan );
    cb->insertItem( pm );
    pm.fill( Qt::blue );
    cb->insertItem( pm );
    pm.fill( Qt::magenta );
    cb->insertItem( pm );

    if( text() == "White" )
        cb->setCurrentItem( 0 );
    else if( text() == "Gray" )
        cb->setCurrentItem( 1 );
    else if( text() == "Black" )
        cb->setCurrentItem( 2 );
    else if( text() == "Red" )
        cb->setCurrentItem( 3 );
    else if( text() == "Yellow" )
        cb->setCurrentItem( 4 );
    else if( text() == "Green" )
        cb->setCurrentItem( 5 );
    else if( text() == "Cyan" )
        cb->setCurrentItem( 6 );
    else if( text() == "Blue" )
        cb->setCurrentItem( 7 );
    else if( text() == "Magenta" )
        cb->setCurrentItem( 8 );

    return cb;
}

```

```

void ColorTableWidgetItem::setContentFromEditor( QWidget *w )
{
    if( w->inherits( "QComboBox" ) )
    {
        switch( ((QComboBox*)w)->currentItem() )
        {
            case 0:
                setText( "White" );
                break;
            case 1:
                setText( "Gray" );
                break;
            case 2:
                setText( "Black" );
                break;
            case 3:
                setText( "Red" );
                break;
            case 4:
                setText( "Yellow" );
                break;
            case 5:
                setText( "Green" );
                break;
            case 6:
                setText( "Cyan" );
                break;
            case 7:
                setText( "Blue" );
                break;
            case 8:
                setText( "Magenta" );
                break;
        }
    }
    else
        QTableWidgetItem::setContentFromEditor( w );
}

```

```

QTable t( 3, 3 );

t.setItem( 0, 0, new QTableWidgetItem( &t, "Red" ) );
t.setItem( 0, 1, new QTableWidgetItem( &t, "Green" ) );
t.setItem( 0, 2, new QTableWidgetItem( &t, "Black" ) );
t.setItem( 1, 0, new QTableWidgetItem( &t, "Yellow" ) );
t.setItem( 1, 1, new QTableWidgetItem( &t, "Magenta" ) );
t.setItem( 1, 2, new QTableWidgetItem( &t, "Blue" ) );
t.setItem( 2, 0, new QTableWidgetItem( &t, "Gray" ) );
t.setItem( 2, 1, new QTableWidgetItem( &t, "Cyan" ) );
t.setItem( 2, 2, new QTableWidgetItem( &t, "White" ) );

```