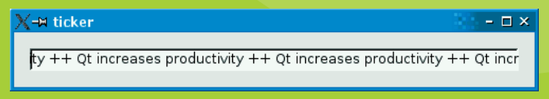
HOW PRODUCTIVE DO YOU WANT TO BE TODAY?

Qt is a strategic technology for cross-platform C++ development. It enables the development of high-performance desktop and server applications that natively target multiple platforms from a single code-base. Through its intuitive API and integrated development tools, Qt increases developer productivity, reduces time to market and delivers real competitive advantage.

HOW MUCH CODE DOES IT TAKE TO BUILD THIS WIDGET?



The answer to this question is – it depends. Perhaps a better question is: which framework is the most efficient for building native, high-end user interfaces? This document presents one answer to this question in terms that any developer or project manager can understand: lines of code. Take a look at the code examples, read the summary below, and draw your own conclusions.

SUMMATION

To build the simple widget shown above, you must write:

With Motif:	224 lines of code
With MFC:	164 lines of code
With Qt:	57 lines of code

To put it another way, Qt reduces the amout of code required by between 64 and 74 percent – for a single platform. This figure, however, takes no account of the cross-platform benefits Qt delivers. Targeting various versions of Windows®, Mac® OS X and X11 requires you to develop and maintain at least three separate source trees. Using Qt, you write and maintain one source tree to target multiple platforms with little more than a simple recompile.

To learn more about how Qt can help increase your productivity, or to download a free 30-day evaluation of Qt, visit www.trolltech.com today.

TROLLTECH®

Oslo, Norway +47 21 60 48 00
Palo Alto, California +1 650 813 1676
Beijing, China +86 10 6588 8836

info@trolltech.com / www.trolltech.com

MOTIF 224 LINES OF CODE

```
/* header */
#include <X11/Intrinsic.h>
extern WidgetClass tickerWidgetClass;
typedef struct _TickerClassRec *TickerWidgetClass;
typedef struct _TickerRec *TickerWidget;
#define XtNtext "text"
#define XtNspeed "speed"
#define XtCText "text"
#define XtCSpeed "speed"
#define XtNbasecolor "basecolor"
#define XtCBasecolor "basecolor"
#define XtNtextcolor "textcolor"
#define XtCTextcolor "textcolor"

/* implementation */
#include <stdio.h>
#include <math.h>
#include <X11/IntrinsicP.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/PrimitiveP.h>
#define MAXSEGMENTS 200
typedef struct _TickerClassPart{
    int ignore;
} TickerClassPart;
typedef struct _TickerClassRec{
    CoreClassPart     core_class;
    XmPrimitiveClassPart  primitive_class;
    TickerClassPart  ticker_class;
} TickerClassRec;
extern TickerClassRec tickerClassRec;
typedef struct _TickerPart {
    char* text;
    int speed, offset;
    Pixel textcolor, basecolor;
    GC text_gc, scroll_gc;
    XFontStruct* font;
    XtIntervalId timer;
} TickerPart;
typedef struct _TickerRec {
    CorePart        core;
    XmPrimitivePart  primitive;
    TickerPart       ticker;
} TickerRec;
static void Initialize();
static void Redisplay();
static void Destroy();
static Boolean SetValues();
static XtResource resources[] = {
  {XtNtext, XtCText, XtRString, sizeof (unsigned char),
   XtOffset(TickerWidget, ticker.text), XtRString, "Hello" },
  {XtNspeed, XtCSpeed, XtRInt, sizeof (int),
   XtOffset(TickerWidget, ticker.speed), XtRString, "20"   },
  {XtNtextcolor, XtCTextcolor, XtRPixel, sizeof (Pixel),
   XtOffset(TickerWidget, ticker.textcolor), XtRString, "Black" },
  {XtNbasecolor, XtCBasecolor, XtRPixel, sizeof (Pixel),
   XtOffset(TickerWidget, ticker.basecolor), XtRString, "White" },
  {XtNfont, XtCFont, XtRFontStruct, sizeof (XFontStruct*),
   XtOffset(TickerWidget, ticker.font), XtRString, "Fixed" }
};
TickerClassRec  tickerClassRec = {
  {    /* CoreClassPart */
    (WidgetClass) &xmPrimitiveClassRec, /* superclass                */
    "Ticker",                   /* class_name                */
    sizeof(TickerRec),          /* widget_size               */
    NULL,                       /* class_initialize          */
    NULL,                       /* class_part_initialize     */
    FALSE,                      /* class_inited              */
    Initialize,                 /* initialize                */
    NULL,                       /* initialize_hook           */
    XtInheritRealize,           /* realize                   */
    NULL,                       /* actions                   */
    0,                          /* num_actions               */
    resources,                  /* resources                 */
    XtNumber(resources),        /* num_resources             */
    NULLQUARK,                  /* xrm_class                 */
    TRUE,                       /* compress_motion           */
    TRUE,                       /* compress_exposure         */
    TRUE,                       /* compress_enterleave       */
    TRUE,                       /* visible_interest          */
    Destroy,                    /* destroy                   */
    NULL,                       /* resize                    */
    Redisplay,                  /* expose                    */
    SetValues,                  /* set_values                */
    NULL,                       /* set_values_hook           */
    XtInheritSetValuesAlmost,   /* set_values_almost         */
    NULL,                       /* get_values_hook           */
    NULL,                       /* accept_focus              */
    XtVersion,                  /* version                   */
    NULL,                       /* callback private          */
    NULL,                       /* tm_table                  */
    NULL,                       /* query_geometry            */
    NULL,                       /* display_accelerator       */
    NULL,                       /* extension                 */
    },
    {0,},     /* Primitive class fields */
    {0,}      /* Ticker class fields */
};
WidgetClass tickerWidgetClass = (WidgetClass) &tickerClassRec;
static void Progress( XtPointer data, XtIntervalId* timer) {
    TickerWidget w = data;
    if ( --w->ticker.offset < -XTextWidth( w->ticker.font, w->ticker.text, strlen(w->ticker.text) ) )
                                w->ticker.offset = 0;
    XCopyArea( XtDisplay(w), XtWindow(w), XtWindow(w), w->ticker.scroll_gc,
                                w->primitive.shadow_thickness+1, w->primitive.shadow_thickness,
                                w->core.width - 2* w->primitive.shadow_thickness - 1,
                                w->core.height - 2*w->primitive.shadow_thickness,
                                w->primitive.shadow_thickness, w->primitive.shadow_thickness );

    w->ticker.timer = XtAddTimeOut( w->ticker.speed, Progress, w );
    XClearArea( XtDisplay(w), XtWindow(w),
                                        w->core.width - w->primitive.shadow_thickness - 2, w->primitive.shadow_thickness,
                                        2, w->core.height - 2* w->primitive.shadow_thickness, True );
}
static void MapNotifyHandler(Widget w, XtPointer client_data,  XEvent *event, Boolean *cont) {
    TickerWidget ticker = (TickerWidget) w;
    if ( event->type = MapNotify  && ticker->ticker.timer == 0 ) {
                        XSetWindowBackground( XtDisplay(w), XtWindow(w), ticker->ticker.basecolor );
                        ticker->ticker.timer = XtAddTimeOut( ticker->ticker.speed, Progress, ticker );
    } else if ( event->type = UnmapNotify && ticker->ticker.timer == 0 ) {
                        XtRemoveTimeOut( ticker->ticker.timer );
                        ticker->ticker.timer = 0;
    }
}
static void Initialize ( TickerWidget request, TickerWidget new ) {
  XGCValues values;
  XtGCMask  valueMask;
  if (request->core.width == 0)
      new->core.width = 40*XTextWidth( new->ticker.font, "x", 1 );
  if (request->core.height == 0)
      new->core.height = new->ticker.font->ascent + new->ticker.font->descent
                                                  + 2*new->primitive.shadow_thickness;
  valueMask = GCFont|GCForeground|GCBackground;
  values.font = new->ticker.font->fid;
  values.foreground = new->ticker.textcolor;
  values.background = new->ticker.basecolor;
  new->ticker.text_gc = XtGetGC ((Widget)new, valueMask, &values);
  valueMask = GCGraphicsExposures;
  values.graphics_exposures = True;
  new->ticker.scroll_gc = XtGetGC ((Widget)new, valueMask, &values);
  new->ticker.offset = 0;
  new->ticker.timer = 0;
  XtAddEventHandler ( (Widget)new, StructureNotifyMask, False, MapNotifyHandler, NULL );
}
static void Destroy (TickerWidget w) {
    XtReleaseGC ((Widget)w, w->ticker.text_gc );
    XtReleaseGC ((Widget)w, w->ticker.scroll_gc );
    if ( w->ticker.timer )
                        XtRemoveTimeOut( w->ticker.timer );
}
static void Redisplay ( TickerWidget w, XEvent *event, Region region) {
    int x;
    XSetRegion( XtDisplay(w), w->ticker.text_gc, region );
    XSetRegion( XtDisplay(w), w->primitive.top_shadow_GC, region );
    XSetRegion( XtDisplay(w), w->primitive.bottom_shadow_GC, region );
    for ( x = 0; x < w->core.width-w->primitive.shadow_thickness - w->ticker.offset;
                                x += XTextWidth( w->ticker.font, w->ticker.text, strlen(w->ticker.text) ) )
                        XDrawString( XtDisplay(w), XtWindow(w), w->ticker.text_gc,
                                                2+x+w->ticker.offset,
                                                (w->core.height + w->ticker.font->ascent+2*w->primitive.shadow_thickness)/2,
                                                w->ticker.text, strlen( w->ticker.text ));
    XmeDrawShadows (XtDisplay (w), XtWindow (w), w->primitive.top_shadow_GC,
                                                w->primitive.bottom_shadow_GC, 0, 0,
                                                w->core.width, w->core.height,
                                                w->primitive.shadow_thickness, XmSHADOW_IN  );
}
static Boolean SetValues (TickerWidget current, TickerWidget request, TickerWidget new,
                                                ArgList args,  Cardinal *num_args) {
  XGCValues values;
  XtGCMask valueMask;
  Boolean redraw = False;
  if( new->ticker.textcolor != current->ticker.textcolor  ||
      new->ticker.font != current->ticker.font) {
      valueMask = GCFont|GCForeground|GCBackground;
      values.font = new->ticker.font->fid;
      values.foreground = new->ticker.textcolor;
      values.background = new->ticker.basecolor;
      XtReleaseGC((Widget)new, new->ticker.text_gc );
      new->ticker.text_gc = XtGetGC ((Widget)new, valueMask, &values);
      redraw = True;
  }
  if( new->ticker.basecolor != current->ticker.basecolor ) {
    if ( XtIsRealized((Widget)new) )
                        XSetWindowBackground( XtDisplay((Widget)new), XtWindow((Widget)new), new->ticker.basecolor );
    redraw = True;
  }
  if ( new->ticker.text != current->ticker.text ) {
      new->ticker.text = strdup( new->ticker.text );
      redraw = True;
  }
  return (redraw);
}

/* usage */
int main( int argc, char* argv[] ) {
  Widget toplevel, ticker;
  Arg wargs[1];
  toplevel = XtInitialize(argv[0], "Ticker", NULL, 0, &argc, argv);
  ticker = XtCreateManagedWidget("ticker", tickerWidgetClass, toplevel, NULL, 0);
  XtSetArg(wargs[0], XtNtext, "This is Motif in all its glory ++ " );
  XtSetValues( ticker, wargs, 1 );
  XtRealizeWidget(toplevel);
  XtMainLoop();
  return(0);
}
```

MFC 164 LINES OF CODE

```
/* header */
#include "stdafx.h"
#include "resource.h"
class CTickerCtrl : public CWnd {
public:
                    CTickerCtrl();
                    void SetText( CString strText );
                    CString GetText();
                    void SetSpeed( int nSpeed );
                    int GetSpeed();
private:
                    CString m_strText;
                    int m_nSpeed;
                    UINT m_timer;
                    int m_nOffset;
                    void progress();
                    //{{AFX_VIRTUAL(CTickerCtrl)
                    //}}AFX_VIRTUAL
protected:
                    //{{AFX_MSG(CTickerCtrl)
                    afx_msg void OnPaint();
                    afx_msg void OnShowWindow(BOOL bShow, UINT nStatus);
                    afx_msg void OnTimer(UINT nIDEvent);
                    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
                    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                    //}}AFX_MSG
                    DECLARE_MESSAGE_MAP()
};

/* implementation */
CTickerCtrl::CTickerCtrl() : m_nSpeed( 20 ), m_timer( 0 ), m_nOffset( 0 ) {};
BEGIN_MESSAGE_MAP(CTickerCtrl, CWnd)
                    //{{AFX_MSG_MAP(CTickerCtrl)
                    ON_WM_PAINT()
                    ON_WM_SHOWWINDOW()
                    ON_WM_TIMER()
                    ON_WM_ERASEBKGND()
                    ON_WM_CREATE()
                    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
void CTickerCtrl::OnPaint() {
                    CRect rcClient;
                    CPaintDC dc(this);
                    GetClientRect( rcClient );
                    dc.SetBkMode( TRANSPARENT );
                    for( int x = 0; x < rcClient.Width() - m_nOffset; x += dc.GetTextExtent( m_strText ).cx )
                        dc.TextOut( x + m_nOffset, ( rcClient.Height() - dc.GetTextExtent( m_strText ).cy ) >> 1, m_strText );
                    dc.DrawSbRect( rcClient, GetSysColor( COLOR_BTNSHADOW ), GetSysColor( COLOR_BTNHILIGHT ) );
}
void CTickerCtrl::OnShowWindow(BOOL bShow, UINT nStatus) {
                    CWnd::OnShowWindow(bShow, nStatus);
                    if( bShow )
                        m_timer = SetTimer( 1, m_nSpeed, NULL );
                    else {
                        if( m_timer )
                                KillTimer( m_timer );
                        m_timer = 0;
                    }
}
CString CTickerCtrl::GetText() { return m_strText; }
void CTickerCtrl::SetText( CString strText ) { m_strText = strText; Invalidate(); }
int CTickerCtrl::GetSpeed() { return m_nSpeed; }
void CTickerCtrl::SetSpeed( int nSpeed ) {
    m_nSpeed = nSpeed;
    if( m_timer )
                    KillTimer( m_timer );
    m_timer = SetTimer( 1, m_nSpeed, NULL );
}
void CTickerCtrl::OnTimer(UINT nIDEvent) {
                    progress();
                    CWnd::OnTimer(nIDEvent);
}
void CTickerCtrl::progress() {
                    CRect rcScroll;
                    GetClientRect( rcScroll );
                    rcScroll.top += 2;
                    rcScroll.left += 2;
                    rcScroll.bottom -= 2;
                    rcScroll.right -= 2;
                    CDC* pDC = GetDC();
                    if( --m_nOffset < -pDC->GetTextExtent( m_strText ).cx )
                        m_nOffset = 0;
                    ReleaseDC( pDC );
                    ScrollWindow( -1, 0, rcScroll, rcScroll );
}
BOOL CTickerCtrl::OnEraseBkgnd(CDC* pDC) {
                    CRect rcClient;
                    GetClientRect( rcClient );
                    pDC->FillSolidRect( rcClient, GetSysColor( COLOR_WINDOW ) );
                    return CWnd::OnEraseBkgnd(pDC);
}
int CTickerCtrl::OnCreate(LPCREATESTRUCT lpCreateStruct) {
                    if (CWnd::OnCreate(lpCreateStruct) == -1)
                                        return -1;
                    CDC* pDC = GetDC();
                    CSize baseSize = pDC->GetTextExtent( "X" );
                    SetWindowPos( NULL, 0, 0, baseSize.cx * 40, baseSize.cy + 4, SWP_NOMOVE | SWP_NOZORDER );
                    ReleaseDC( pDC );
                    return 0;
}

/* usage */
class CTickerDlg : public CDialog {
public:
                    CTickerDlg(CWnd* pParent = NULL);
                    //{{AFX_DATA(CTickerDlg)
                    enum { IDD = IDD_TICKER_DIALOG };
                    //}}AFX_DATA
                    //{{AFX_VIRTUAL(CTickerDlg)
                    virtual BOOL DestroyWindow();
                    //}}AFX_VIRTUAL
private:
                    CTickerCtrl* m_pTicker;
protected:
                    //{{AFX_MSG(CTickerDlg)
                    virtual BOOL OnInitDialog();
                    afx_msg void OnSize(UINT nType, int cx, int cy);
                    //}}AFX_MSG
                    DECLARE_MESSAGE_MAP()
};
CTickerDlg::CTickerDlg(CWnd* pParent /*=NULL*/) : CDialog( CTickerDlg::IDD, NULL), m_pTicker( NULL ) {}
BEGIN_MESSAGE_MAP(CTickerDlg, CDialog)
                    //{{AFX_MSG_MAP(CTickerDlg)
                    ON_WM_SIZE()
                    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
BOOL CTickerDlg::OnInitDialog() {
                    CRect rcClient, rcWnd, rcCtrl;
                    CDialog::OnInitDialog();
                    GetClientRect( rcClient );
                    GetWindowRect( rcWnd );
                    m_pTicker = new CTickerCtrl;
                    m_pTicker->Create( NULL, NULL, WS_CHILD | WS_VISIBLE , rcClient, this, -1 );
                    m_pTicker->GetClientRect( rcCtrl );
                    SetWindowPos( NULL, 0, 0, rcCtrl.Width() + rcWnd.Width() - rcClient.Width(), rcCtrl.Height() + rcWnd.Height() - rcClient.
Height(), SWP_NOMOVE | SWP_NOZORDER );
                    m_pTicker->SetText( " MFC is the Microsoft Foundation Classes ++ " );
                    return TRUE;
}
BOOL CTickerDlg::DestroyWindow() {
        delete m_pTicker;
                    return CDialog::DestroyWindow();
}
void CTickerDlg::OnSize(UINT nType, int cx, int cy) {
                    CRect rcClient;
                    CDialog::OnSize(nType, cx, cy);
                    GetClientRect( rcClient );
        if( m_pTicker )
                        m_pTicker->SetWindowPos( NULL, 0, 0, rcClient.Width(), rcClient.Height(), SWP_NOMOVE | SWP_NOZORDER );
}
class CTickerApp : public CWinApp {
public:
                    CTickerApp();
                    //{{AFX_VIRTUAL(CTickerApp)
                    public:
                    virtual BOOL InitInstance();
                    //}}AFX_VIRTUAL
                    //{{AFX_MSG(CTickerApp)
                    //}}AFX_MSG
                    DECLARE_MESSAGE_MAP()
};
BEGIN_MESSAGE_MAP(CTickerApp, CWinApp)
                    //{{AFX_MSG_MAP(CTickerApp)
                    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
CTickerApp::CTickerApp(){}
CTickerApp theApp;
BOOL CTickerApp::InitInstance() {
                    Enable3dControls();
                    CTickerDlg dlg;
                    dlg.DoModal();
                    return FALSE;
}
```

Qt® 57 LINES OF CODE

```
/* header */
#include <QtGui>
class Ticker : public QFrame {
    Q_OBJECT
    Q_PROPERTY(QString text READ text WRITE setText)
    Q_PROPERTY(int speed READ speed WRITE setSpeed)
public:
    Ticker(const QString &text, QWidget *parent = 0);
    void setText(const QString &txt) { txt = text; update(); }
    QString text() const { return txt; }
    QSize sizeHint() const;
    void setSpeed(int speed) { timer.setInterval(speed); }
    int speed() const { return timer.interval(); }
protected:
    void showEvent(QShowEvent *) { timer.start(); }
    void hideEvent(QHideEvent *) { timer.stop(); }
    void paintEvent(QPaintEvent *event);
private slots:
    void progress();
private:
    QString txt; QTimer timer; int offset;
};

/* implementation */
#include "ticker.h"
Ticker::Ticker(const QString &text, QWidget *parent)
    : QFrame(parent), txt(text), offset(0) {
    setFrameStyle(QFrame::Panel | Sunken);
    setBackgroundRole(QPalette::Base);
    setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
    timer.setInterval(20);
    connect(&timer, SIGNAL(timeout()), this, SLOT(progress()));
}
QSize Ticker::sizeHint() const {
    return QSize(40 * fontMetrics().width('x'), fontMetrics().lineSpacing() + 2 * frameWidth());
}
void Ticker::paintEvent(QPaintEvent *event) {
    QFrame::paintEvent(event);
    QPainter painter(this);
    for (int x = 0; x < width() - offset; x += fontMetrics().width(txt))
        painter.drawText(x + offset, 0, 65536, height(), Qt::AlignVCenter, txt);
}
void Ticker::progress() {
    if (--offset < -fontMetrics().width(txt)) offset = 0;
    update();
}

/* usage */
#include "ticker.h"
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QWidget window;
    QGridLayout layout(&window);
    layout.setMargin(15);
    Ticker ticker("Qt increases productivity ++ ");
    layout.addWidget(&ticker, 1, 1);
    window.show();
    return app.exec();
}
```

TO COMPARE THE CODE!

Qt BY TROLLTECH®

CODE LESS. CREATE MORE.

© 2005 Trolltech A.S. Trolltech, Qt, Qtopia and respective logos are registered trademarks of Trolltech AS in Norway and/or other countries worldwide. All other trademarks are property of their respective owners. QTCC-P-05/12