Porting X/Motif Applications to Qt®

Scenarios and Advice for a Smooth Migration

Integrated Computer
Solutions Incorporated

The User Interface Company™

54 B Middlesex Turnpike
Bedford, MA 01730
617.621.0060
info@ics.com

www.ics.com

Porting X/Motif Applications to Qt®

Scenarios and Advice for a Smooth Migration

Table of Contents

Introduction 4
  Reasons for porting 4
  Evaluating the state of your application 4
Anatomy of an application 5
  Hello Motif, Hello Qt 6
Toolkit Comparison 7
  Widget-set comparison 7
    Mapping of common UI objects 7
    Menus and Options 8
    Dialogs 8
    Complex widgets 8
  Layout Management 10
  Event Handling 11
  Look and Feel 12
  Development environment 13
    Language, compilers, libraries 13
    Distribution, Installation and Documentation 13
  Interclient communication 14
    Selection and Clipboard 14
    Drag and Drop 14
  Automation: GUI builder 15
    Scripting: QSA, PyQt, Tq 15
    Testing and Inspection 15
  I18N, Session Management and Help 16
    Internationalization 16
    Session Management 16
    Help 17
  Extras 17
Porting your application 17
  Port or Redesign? 18
  Gradual migration using the Qt Motif Extension 18
  Porting Strategies for Applications 19
    Hand-written GUI 19
    GUI-builder generated 19
    C-based 19
    Graphics heavy 19

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

Custom widgets 21
Get Help 22
Qt Training, Advanced Qt Components and Testing Tools 23
QicsTable™ 23
GraphPak™ 23
KD Executor™ 23
KD Tools™ 23
KD Gantt™ 24
Motif to Qt Migration Assessment 24
About ICS 25

Copyright © 2004-2006 Integrated Computer Solutions, Inc. All rights reserved. This document may not be reproduced without written consent from Integrated Computer Solutions, Inc. Qt is a registered trademark of Trolltech AS. QicsTable and GraphPak are trademarks of Integrated Computer Solutions, Inc. All other trademarks are property of their owners.

Porting X/Motif Applications to Qt®

Scenarios and Advice for a Smooth Migration

Introduction

X/Motif-based applications in general have been limited to Unix™ based platforms. Implementation inconsistencies and vendor-specific "enhancements" made porting the same application on variants of UNIX a time consuming and costly task. The complexities of development environments, programmer skills set, and deployment for a GUI-heavy large application coupled with Motif's heavy dependence on X/Xt, prohibits many organizations from expanding their user base to alternate platforms.

The Qt application development framework from Trolltech is gaining popularity among developers as a full-featured, multiplatform, C++-based toolkit. It is easy to use, has a rich set of widgets and is well documented and supported.

In this article, we will examine some common scenarios encountered when attempting to port applications from X/Motif to Qt and offer some advice on how to make the process smoother. We begin by identifying some reasons and possible benefits of porting to Qt. The next chapter takes a look at the anatomy of applications built using the two toolkits and compares the features of the two toolkits. The third chapter offers advice and conversion strategies for different types of applications. We conclude by pointing to resources for help.

Reasons for porting

The primary reason organizations decide to port their application to Qt is to make their applications available on multiple platforms. Other possible reasons cited are:

- You are getting around to porting to Motif 2.1 and have a lot of older custom widgets that will require a rewrite anyway.
- You want to write cleaner, type-safe, object-oriented code in C++ without using static callback functions that use the typeless void *.
- Want to write custom widgets without having to go through a large manual and writing 10000 to 20000 lines of code per widget.
- You want professional support, quick fixes and frequent releases for the toolkit.
- You are saving cost by moving to lower cost Linux boxes and have some resources available to port to Qt and hence other platforms to expand your user/market base.

Evaluating the state of your application

There are several costs and concerns associated with porting to a new toolkit. Available resources need to be approximated and the projected costs need to be analyzed in conjunction with potential benefits and gains to make a successful and justifiable transition from one toolkit to the other. If you are just starting out, you should get an evaluation license for Qt and get familiar with the API by experimenting with some

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

sample code and tutorials. Evaluate your existing applications and future needs based on the following factors:

- Complexity of the existing application: Consider architectural complexity, line and file count. Motif applications tend to be “verbose”.
- Data complexity: Sources of data. Drivers for data sources.
- Unusual device dependency: Tablets, dials and button boxes.
- Handwritten GUI code vs. Builder-generated UI code: Qt comes prepackaged with a GUI builder but easier translation may depend on some support from your existing Motif GUI builder.
- Available time and resources: Do you have sufficient time available for the porting effort? Keep in mind that the current Motif application will continue to need support and fixes during the porting. Do you need to hire additional programmers with C++ expertise? You may need machines to build and test on other platforms.
- User expectations: You need to communicate to the users the future directions that you are taking. Keep them in the loop and solicit feedback from them for needed enhancements. Give them some value added features and not just a simple conversion. It is easier to keep your existing customers than to acquire new ones.

After this initial evaluation, you should have a pretty good idea on the number of licenses and platforms for which you need to develop. In a later chapter we will discuss strategies best suited to specific categories of applications. First, let us look at general differences between the two toolkits.

Anatomy of an application

Motif and Qt both have their roots in the X Window System and use the same event-driven application model as X. This commonality, combined with similar basic widget sets with hierarchical parent-child relationships, makes the porting viable.

The applications execute code in response to an event. The event can be caused by a device (operated by the user), by a timer or internally from within the code. The event is intercepted by the window manager and dispatched to the appropriate widget via the application. The widget responds by executing some procedure and the whole cycle repeats in an event-loop until the application is terminated.

In Motif, the application can define *callback procedures* and add them to the widget’s list of procedures. These procedures are executed when the right event is triggered. Action routines and event handlers are some of the other methods in Motif/Xt to allow finer control of event processing.

Qt provides an alternative to the callback procedures using a *signals* and *slots* methodology. Signals are emitted by widgets in response to an event. Slot procedures are connected to signals in a type-safe manner. Beyond this initial connection, the objects have no knowledge of each other.

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

Another area where the toolkits differ is the layout management of the widgets. In Motif, the manager widgets are used as parents. The manager widgets are responsible for the layout of the child widgets. In Qt, layout is independent of the widgets. This flexibility allows the programmer to create arbitrary layouts although the supported layouts are sufficient for most uses.

The following table highlights the similarities using a very high level comparison of the toolkit methods needed to implement a basic application.

Description	Motif	Qt
Intialize	XtAppInitialize	QApplication()
Create Widgets	XmCreate<WidgetName>()	Q<WidgetName>()
Event Response	XtAddCallback()	QObject::connect()
Show widget	XtRealizeWidget()	QWidget::show()
Enter the Event Loop	XtAppMainLoop()	QApplication::exec()

Hello Motif, Hello Qt

In the following table, minimal but complete programs to compare the basic styles of applications using Motif and Qt are presented.

Simple Motif Program with Callbacks

```
void toggleChangedCB(
           XmToggleButton tb,
           int data,
      XmToggleButtonCallbackStruct *tbInfo)
{
   bool toggleVal =  tbInfo->set;

   cerr << "The button state is: "
        << toggleVal << endl
        << "The app data is: "
        << data << endl;
}

int main(int argc, char **argv)
{
Widget topLevel, tb;
XtAppContext app;

   topLevel = XtAppInitialize(&app,
         "Example", NULL, 0,
         &argc, argv, NULL, NULL, 0);

   tb = XmCreateToggleButton(topLevel,
         "Hello Motif",
         NULL,
         0);

   XtAddCallback(tb,
         XmNvalueChangedCallback,
         (XtCallbackProc)  toggleChangedCB,
         (XtPointer)  42);

   XtRealizeWidget(topLevel);
   XtAppMainLoop(app);
}
```

Simple Qt Program with Signals & Slots

```
class SomeObj : public QObject
{
Q_OBJECT
public:
   SomeObj(int objData)
     : QObject(0, "SomeObject"),
      _objData(objData) ) {};
public slots:
   void toggled(bool toggleVal);
protected:
   int _objData;
};

void SomeObj::toggled(bool toggleVal) {
   cerr << "The button state is: "
        << toggleVal << endl
        << "The app data is: "
        << _objData << endl;
};

int main(int argc, char ** argv) {
   QApplication app( argc, argv );
   QPushButton tb( "Hello Qt" );

   tb.setCheckable(true);
   tb.setChecked(true);
   tb.show();

   SomeObj myObject(42);

   QObject::connect(
      &tb, SIGNAL(toggled(bool)),
      &myObject, SLOT(toggled(bool))
   );

  return app.exec();
}
```

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
•www.ics.com • Phone: 617.621.0060

Toolkit Comparison

Widget-set comparison

The following section, offers a comparison of most of the widgets in Qt. You can use this list as a general guide when porting your application.

Mapping of common UI objects

Functionality	Motif	Qt	Major Differences
Text/Pixmap Label	XmLabel	QLabel	QLabel can also display RichText or a Movie
PushButton, ToggleButton	XmPushButton, XmToggleButton	QPushButton	Same class offers both kinds of functionality. Otherwise similar.
Radio Button, CheckBox	XmToggleButton -XmRowColumn	QCheckBox, QRadioButton (QButtonGroup)	In Motif, the parent RowColumn's resources dictate the behavior
Scrollbar	XmScrollBar	QScrollBar	Very similar
Scale	XmScale	QSlider	Very similar
List of items	XmList, XmContainer	QListView	QListView can do multicolumn, hierarchical lists of arbitrary type. Motif XmList is limited to single column, linear list of XmStrings but XmContainer is as powerful.
Single Line Text Field	XmTextField	QLineEdit	QLineEdit supports RichText, encrypted entry, validator & built in undo/redo
Text	XmText	QTextEdit	Same as above, plus a LogText mode for very large amounts of text.
Horizontal/Vertical Tiled Windows	XmPanedWindow	QSplitter	Very similar
Spin Box	XmSpinBox	QSpinBox	Very similar

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 • • www.ics.com • Phone: 617.621.0060

Menus and Options

Functionality	Motif	Qt	Major Differences
Horizontal Menubar	XmMenuBar	QMenuBar	QMenuBar has several insertItem methods
Pulldown Menu	XmPulldownMenu	QPopupMenu	Qt provides a cleaner abstraction using QPopupMenu as the common object for pulldowns and popup
Popup Menu	XmCreatePopupMenu	QPopupMenu	
Option Menu, Combo Box	XmComboBox, XmOptionMenu	QComboBox	Qt uses ComboBox for both

Dialogs

Both toolkits provide a similar and rich set of built-in dialogs for various tasks. In Motif, dialogs are usually created using one the numerous convenience XmCreate*Dialog functions. Qt provides some complex dialogs that are not available in Motif, these include, QColorDialog (to choose colors), QFontDialog (to choose fonts), QWizard (a framework for a sequence of dialog pages) and QTabDialog (similar to QWizard but the user is free to choose non-sequentially.) The dialogs can be modal or modeless in both toolkits. Dialog comparison is presented in the following table:

Functionality	Motif	Qt
Error	XmCreateErrorDialog	QErrorMessage
Information	XmCreateInformationDialog	QMessageBox
Question	XmCreateQuestionDialog	QMessageBox
Warning	XmCreateWarningDialog	QMessageBox
Progress	XmCreateWorkingDialog	QProgressDialog
Simple Input	XmCreatePromptDialog	QInputDialog
File Chooser	XmCreateFileDialog	QFileDialog
Item Chooser	XmCreateSelectionDialog	Not Available, use QListBox
Command History	XmCreateCommand	Not Available
Generic/Custom	XmCreateMessageDialog	QDialog

Complex widgets

- **Main Window**: The Motif MainWindow widget provides a predefined layout that most applications should use as suggested by the style guide. The main window consists of a *menu bar* at the top, followed by a scrollable and resizable *work area* (the main interface of the application), an optional *command area* below the work area where the users can enter commands and a *message area* at the bottom to display status messages and to provide feedback to the user. The Qt main window widget (QMainWindow) is similar. In addition to the menu bar, it provides areas

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
•www.ics.com • Phone: 617.621.0060

for multiple dockable toolbars (four dock areas are automatically built). Any Qt widget can be the *work area* (called the *central widget*). A *status bar* at the bottom is also included.

- **OpenGL**: Qt provides a cross platform GUI interface for OpenGL rendering similar to the GLwMDrawingAreaWidgetClass in Motif. The Qt version has higher level functionality in terms of overlay plane rendering, string and pixmap support. A detailed description is presented in a later section.
- **Table**: QTable widget in Qt provides basic spreadsheet like functionality. For the ultimate in Qt based table widget functionality and flexibility in data modeling, ICS has a QicsTable widget. Commercial add-on packages for Motif have been available for some time. Motif 2.0 onwards, the XmContainer widget can be configured for use as a table.
- **Iconview**: QIconView widget in Qt can be used to create interactive graphical file browser or album like applications using movable icons. XmContainer introduced in Motif 2.0 can provide similar functionality when properly configured.
- **Workspace**: The QWorkspace widget provides a Multiple Document Interface (MDI). In such applications, the main window behaves like a mini window manager and may contain several other widgets (usually of the same type), that represent the document being viewed or edited. Automatic focus switching, activation and layout management is also supported. Motif does not have anything similar but a simple open source implementation is available from the X contribution site.
- **Tab Widget**: QTabWidget is similar in functionality to The XmNotebook (introduced in Motif 2.0) widget and provides a widget stack
- **Canvas**: QCanvas in Qt is a module available only with the Enterprise Edition. It provides optimized object-based 2D graphics with support for animation, collision detection and multiple views. The Motif drawing area is normally used to implement such functionality but the Canvas allows you to do much more with less code.
- **Frame**: Although not a complex widget, it causes a lot of confusion to beginners trying to port the usage from Motif where the XmFrame widget is used as a container that hosts one child inside a border drawn by the frame widget. In Qt, the QFrame widget provides a similar frame but it is meant to be used a base class widget. The subclass is responsible for making sure that there is enough area around the QFrame part so that the border is not obscured by the child widget. In Qt, the QGroupBox should be used to provide similar functionality. Alternatively, developers can use widgets like QHBox or QVBox that are already based on QFrame.

There are several other widgets in Qt that don't have a direct counterpart in Motif. A few of these are listed below:

- **QDial**: Similar to a slider/scale widget but round in shape with wrap around ability.
- **QLCDNumber**: For stylized display of numbers.

- **QDateTimeEdit**: These are like the spin box but specialized for date and time editing with built-in validators.
- **QToolBox**: Similar to a tab widget, useful for space saving configuration dialogs.
- **QToolBar, QToolButton**: To create a movable panel of frequently used actions.
- **QDockArea, QDockWindow**: To create dockable, movable panels of widgets for providing end user layout control in applications.
- **QSplashScreen**: This widget is useful for displaying an animatable graphic and progress during application startup.
- **QsvgWidget**: Makes displaying SVG static and animated drawings as easy as a bitmap image.
- **SVG graphics now supported**: SVG (an XML 2D graphics format) rendering can be directed into any QPaintDevice, and can make use of hardware accelerated rendering using Open GL.

Layout Management

In Motif, special widgets, based on XmManager widget, control the resize behavior and the layout of children widgets. They vary in the degree of constraint they impose on the children. In Qt, layouts are not widgets but are C++ objects that determine the size and position of the widgets within a parent container. The children can provide size and alignment hints that the layout can use. A few common Motif layouts along with suggestions for porting them to Qt are listed below:

- **XmBulletinBoard**: This is the simplest manager widget to convert as it does not control the size or position of its children. The absolute positions and sizes can be specified in Qt using the *setGeometry* call. Qt's GUI builder, Qt Designer, can be also used effectively for such layouts.
- **XmScrolledWindow**: Use Qt's QScrollView widget's viewport as the parent and add the child widget to the QScrollView widget using the addChild method.
- **XmRowColumn**: Use QHBoxLayout (organizes widgets in a row) or QVBoxLayout (organizes children in a column) for simple layouts. The QGridLayout object provides more flexibility but is a little more complex to use. These layouts also come as convenience widgets (QHBox, QVBox and QGrid) that include the corresponding layouts. These widgets can be easier to use when transitioning from Motif but you lose some flexibility.
- **XmPanedWindow**: Use the QSplitter widget for almost identical functionality.
- **XmFrame**: Use QGroupBox. Do not use the QFrame widget directly.
- **XmRadioBox**: Use the QButtonGroup widget.
- **XmForm**: The XmForm widget in Motif is an extremely powerful and complex widget that uses a constraint based approach to geometry management. Various resources on parents and children are used to specify the layout and resize behavior of the children.
- Experienced Motif developers know and use the XmForm widget very well and feel crippled and frustrated when they don't see something equivalent in Qt. Qt's layout management is more general and one can use it to create XmForm like behavior but in our opinion, the better technique is to truly understand the layout

behavior of your widgets and implement the overall behavior rather than focus on one-to-one conversion. A combination of correct layouts objects and attributes on widgets like sizePolicy, stretchFactor, minimum/maximumSize/Hint can provide the desired result.

Some tips for layout management troubleshooting:

- Use the Designer. Experiment with various widgets and layouts and observe how they behave. Avoid setting explicit geometry. Generate the source code to see how you can incorporate something similar in your code.
- Use simple QWidgets with bright disparate background colors or labels for all widgets in a layout. This makes it easy to visually spot problem areas.
- You may have to subclass certain widgets to override the sizeHint and minimumSizeHint values to be able to resize them properly.

Event Handling

Developers have varying views of the event handling mechanisms in Motif depending upon their experience and exposure. This is because of the three layers of toolkits (X11, Xt & Motif) and the levels of abstraction provided by them, which can be used (often in the same application) for event handling. This section describes the common event handling methods in Motif applications and shows how they can be adapted to Qt's event handling.

Overview:

- At the basic level, each widget instance contains a translation table that maps events to procedure names. Another table, the action table, maps procedure names to actual procedures. When an event gets dispatched to a widget, the corresponding action procedure is invoked.
- In Qt, events are delivered to objects derived from the base QObject class using the QObject::event interface.

Do Nothing:

- In Motif, widget instances "inherit" built-in event handlers that take care of the basic behavior (for example, pressing the backspace key in a text widget erases a character).
- In Qt, the C++ inheritance model takes care of the default behavior of widget instances/subclasses.

Behavioral Events:

- Widgets that anticipate behavior augmentation allow the programmer to add procedures and data for certain combination of events. For example, for an XmPushButton, the callbacks in the XmNactivateCallback list are called when the user presses and releases the active mouse button while the pointer is inside the widget. For most applications, use of these callback lists and procedures are sufficient.

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

- Qt has signals and slots. A signal is emitted when an event occurs. A slot is a method that is called in response to a signal. A signal from an object can be connected to a slot in any other object as long as the signature and the type is compatible. Most built-in widgets emit signals on events that are meaningful to their functionality. For example, a QPushButton emits a clicked signal when the user presses and releases the active mouse button while the pointer is inside the widget. Slots that are connected to the signals are used to add the functionality.

Finer Control:
- For finer control over event handling, the translation tables and actions procedures can be changed or added to Motif widgets.
- Additional signals can be easily added in subclassed Qt widgets.

Direct Dispatch:
- A Motif application can register event handler procedures with the Xt event dispatcher for a particular widget. These procedures are called before (and can preempt) the ones in the translation table.
- The QObject::event virtual method can be overridden to bypass or to add event handling for a widget. Most Qt widgets also have several event handling methods that are at a higher level of abstraction and can be overridden without completely changing the behavior of a widget through the lower level QObject::event method. A widget can observe (and usurp) another widget's events using the QObject::eventFilter, QObject::installEventFilter methods.

More Control:
- Some applications also grab the keyboard, pointer, or the entire server for specialized event handling. It is also possible to go down to the level of X11 to peek at, extract, resend events from the event queue.
- In Qt, more control can be exercised using the available methods on the main application object to manipulate the event loop or by installing an application-wide event filter. Creation and dispatch of custom and non-gui events is also possible.

Look and Feel

On the MotifZone page, there is a survey that lists *New Motif Look & Feel* as the most important feature for Motif 2.4. In the past, vendors have tried to give their own unique look to Motif (most notable, the SGI look) but user control has been limited to manipulation of font, color and other resources.

Qt provides control over the look and feel of applications via the QStyle interface. There are reasonable defaults for all major platforms. The QStyle interface lets the developer define alternate visual representation of different primitives, controls and complex widgets. Overall metrics such as thicknesses, widths, and spaces can also be controlled. Within an application, each widget can be given a different style (not recommended), or

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

the style of the whole application replaced by a custom style. Styles can also be installed as plug-ins. These can then be used by applications at run time when properly configured.

Development environment

Language, compilers, libraries

If you are using KDE as your graphical desktop environment, you should know that KDE is built upon the free version of the Qt toolkit. In addition, most distributions of Linux include the free version of the toolkit. If you are a paying customer of Trolltech, it is highly recommended that you get the toolkit in source format and build it for the platforms for which you have valid licenses. You can tailor the build and installation to suit your needs. It is customary to set the QTDIR environment variable to point to the installed location. Do not forget to set the paths properly to avoid picking up the wrong library or headers.

The Qt toolkit includes a build system called qmake. Qmake is a Makefile generator that takes care of managing variations in compiler and platform dependencies. Qmake is used to build the Qt toolkit. It also supports special constructs for meta-object file generation using *moc* and code generation from user-interface files using *uic*.
Large X/Motif applications (and the X/Motif toolkits themselves) have traditionally been built using Imake and its associated configuration files that are used to generate the Makefiles to build the project. The Imake configuration files that are included with X11 are numerous and complex. Imake uses the C preprocessor under the hood and the configuration file constructs do not allow for much processing beyond the macros and #if statements. Despite the complexity and hard to find documentation, Imake has been successfully used to build X11, Motif and other large products on various Unix-based platforms with different compilers. The qmake .pro files are similar in concept to Imakefiles. There are built-in templates for binaries, libraries, hierarchical subdirectories, etc. that are akin to the ones in Imake .rules files. Qmake is much superior to Imake in its processing power achieved via the many operators, functions and support for regular expression matching.

If you don't want the additional burden to convert existing makefiles or if you have other non-gui based products and would like to keep using your current build system, the rules for moc and uic can be incorporated directly into makefiles or as Imake rules.

Qt is C++ based, and works with g++ and the native C++ compiler on most variations of Unix. The KDevelop IDE makes Qt widgets (and many KDE extension widgets) available through its interface. A tighter integration with other IDEs is promised in version 4.0 of Qt.

Distribution, Installation and Documentation

Trolltech makes the Qt toolkit available in source form. Many products based on Qt use a licensing model similar to Trolltech's and make their source code available to the buyers. By providing the source code and build instructions to the customer, it is easier to support

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

multiple platforms and versions as you don't have to re-build and re-release your product for each small variation. You also get the added benefit of better bug reporting and suggestions for improvements.

If you prefer, you can also distribute just the executables and shared libraries for your product if you are a paying customer for Trolltech.

Doxygen is an open-source documentation package for C++ capable of generating rich hyperlinked documentation in a variety of formats. The documentation is generated automatically from tags and comments embedded in the source code. It understands signals and slots constructs and even has a front-end written in Qt to edit the configuration file. Doxygen is available from http://www.stack.nl/~dimitri/doxygen/

Interclient communication

Selection and Clipboard

X Window supports both a Primary selection and a Clipboard selection. The Primary selection enables availability of data as soon as it is selected. The Clipboard provides more traditional support by making data available that has been explicitly placed in the clipboard using a copy or a cut operation. Qt has built-in support for both kinds of selections using the QClipboard class. The DragObject data model as described below can be used to convert selection data during copy or paste operations.

Drag and Drop

Drag and Drop can be used to transfer data within and between applications. Although similar in principle to copy and paste, the interaction from user's perspective is more direct. It could be as simple as selecting and moving text around in a document or as complex as automatically converting between different representations of data. Motif's support of drag and drop is implemented on top of the selection mechanisms provided by Xt and the X Inter-Client Communications Conventions Manual (ICCCM). Motif uses non-visible widget-like objects (DragContext, DropSite, DropTransfer, DragIcon) to facilitate drag and drop. Drag sources and drop sites use Atoms to specify the supported data formats. The complexity of the drag and drop in Motif is also increased by the need to repeat the data conversion code each time for selection, clipboard and drag & drop transfer. The Uniform Transfer Model (UTM), in Motif 2.0 and onwards, has eased things a bit but drag and drop remains cumbersome to support in Motif and most applications end up just using the default support in Text and Label widgets.

Under X11, Drag and Drop support in Qt is based on the XDND protocol. It can also handle drops made using the Motif drag and drop protocol. XDND is safer in terms of potential data corruption and race conditions. The Qt abstraction provides a DragObject base class that can be used to represent data. The DragObjects are independent of the drop sites and use MIME types instead of Atoms to signify data formats. As with the UTM in Motif, the same mechanism can be used to transfer data to the clipboard or even for automatic translation during file IO. Desktop drag and drop is supported using the QDesktopWidget that spans the virtual desktop or the primary screen. Left mouse button

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

is generally used to initiate drag in Qt applications in keeping with its multiplatform support rather than the Motif style-guide recommended middle mouse button.

Automation: GUI builder

Qt comes with a GUI builder named Qt Designer that allows you to design Qt based applications, widgets, dialogs and other UI elements. It also lets you connect the signals with slots and program the application behavior using a built in syntax-highlighting text editor. The GUI description can be saved in XML format and then converted into C++ header and implementation files via a supplied generator binary (User Interface Compiler [uic]). It is advised that you keep the generated code untouched and use the Generation Gap pattern of subclassing to extend the behavior of the generated classes instead of modifying the generated code. You can also add custom widgets to Qt Designer for reuse in other applications.
Most simple applications (and applications that mostly display data in various dialogs and forms) are much easier to convert using Qt Designer. Using editres, it is possible to write the complete hierarchy of the widgets from a Motif application. Duplicating this hierarchy for simple widgets is relatively straightforward using the comparisons and features of widgets as described earlier in this article.

Scripting: QSA, PyQt, Tq

QSA (Qt Script for Applications) is another product available from Trolltech that can make Qt-based applications scriptable. There is nothing quite like it in the Motif world. tclMotif, Wafe, Winterp etc. don't quite measure up. QSA has more in common with VBA (Visual Basic for Applications) on the Windows platform. The QSA script is based on ECMAScript and can be used to program and interact with a Qt application. The product also comes with a simple IDE and a Dialog Framework to extend any supported application.

PyQt, a third party product provides Python language bindings for the Qt toolkit and allows for scripting of Qt applications. Tq is yet another product that allows Tk and Qt event loops to coexist in a single application.

Testing and Inspection

Various commercial packages have been available for automated testing of X/Motif based applications. These range from event level "capture and playback" to object (widget) level interaction recording and playback (Centerline, Software Research Inc., XRunner). QTestLib, a recently added standard part of Qt, is an application and library unit testing framework. KD Executor from ICS also provides an easy way to perform thorough regression testing. QSA, described earlier, can also be used in a limited way to automate testing.

Editres has long been a favorite tool of Xt/Motif developers due to its ready availability and its seemingly simple yet powerful features. Most developers have used editres as a debugging tool for application resources like color, layout, geometry, fonts etc. and to examine and "learn" from the widget hierarchy of complex applications. When you port

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

your application to Qt, similar functionality can be added with very little extra code by utilizing the object-oriented nature and the natural widget hierarchy of the Qt toolkit. If you are an Enterprise licensee, Trolltech provides a set of custom components and tools (Qt Solutions) free of charge. One of these solutions, Object Inspector provides editres-like functionality. In addition to the current resource (property) values, it also shows the connection graphs for callbacks (signals and slots) and reports their activation as they happen.

I18N, Session Management and Help

Internationalization

Most application writers incorrectly assume that internationalization involves just converting all the strings displayed in the user interface. I18N should also include conversion of currency, time, date and data formats as well as culture and country appropriate icons, bitmaps and cursors. Qt provides I18N support by a combination of the following four mechanisms:

- QString: Uses Unicode (a simple encoding for every character) for text processing.
- QObject::tr(): For ensuring translation for all user visible string.
- Tools:
  - *lupdate*: To extract translatable information from source files.
  - *linguist*, a simple application for translators to convert extracted data from lupdate.
  - *lrelease*: to create compact binary lookup table to be used at application run-time.
- Support for various languages using Qt's text engine for all input (QTextEdit, QLineEdit) and display (QLabel) controls.

There is also support for encodings other than Unicode. Localization support can be provided by creating keywords for formats and storing them as translations. One nice feature is that Qt Designer created GUIs automatically allow for dynamic localization, so a developer can very easily create applications that will change their displayed language on the fly, at run time.

Session Management

On X Windows, client applications can participate in session management by registering themselves with the Session Manager using the X Session Manager Protocol. Client and Session Manager communicate using the protocol to initiate requests for saving states, terminating, checkpointing etc. Session Manager also provides for grouping of instances of a single application and for uniquely identifying instances. Qt provides full support for the X Session Manager by encapsulating the protocol details in the QSessionManager class. Application state can be saved using QSettings.

In practice, we have found that although the functionality provided by the X Session Manager is powerful, not many end users run the X session manager (or even know about

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
•www.ics.com • Phone: 617.621.0060

it). It is more likely that you will end up developing a simpler session management system that is more localized for your application and is not bound to a system level service. One such mechanism, as employed in the Viewkit toolkit, is to call an okToQuit method on each top level window in the reverse order of their creation on termination. When all the top level windows have returned true (and saved all the state data), the application safely exits.

Help

Help is one area where Motif has been behind all the modern user interface toolkits. There is not much support beyond the cumbersome help callback, the XmTrackingLocate for context sensitive help and the recently added XmtoolTipString resource.

Users have come to expect four kinds of help in an application. The ToolTip help that appears temporarily when the user hovers over a control, the message line help that appears immediately in the status line at the bottom of the application when the mouse is over a control, context sensitive help, usually a longer version of the tool tip when the user presses the F1 key over a control and finally a full fledged index, hyperlinked detailed help with usage guide and a search facility (usually available from the Help menu). Qt provides support for all the above kinds of help using easy to use classes. (QToolTip, QStatusBar and Qt Assistant)

Extras

Qt comes with support for a lot of things that are not necessarily GUI related but they make the task of writing modern applications much easier. There are classes for I/O and networking client and server support for various protocols, image processing and encoding, XML/DOM parsing, date & time, SQL database and even an STL-like template library. Additionally, there are classes for text editors that do syntax highlighting, tree and table viewers that work with a model – view – control (MVC) paradigm to allow multiple different views synchronized to the same data. Qt also provides it own Java style multi-threading module. Finally, Qt has its own resource system that basically allows anything that can be treated like a file to be compiled into memory resident virtual file system.

Porting your application

We recommend that you find the smallest Motif-based application that you have in your organization and attempt to port it first. Take the small application all the way from changes in build environment to a full fledged Qt-based application written in C++. After the conversion, add a few new features. Note down your experiences and stumbling blocks. Ideas and concepts those were difficult to translate for a small application may require you to learn new techniques or seek expert help.

You should try a larger application next. You will learn that there are no hard and fast rules for the conversion process. Methods that work for one application are not

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

necessarily going to work for the other. In some cases you might even end up with a radically different set of functionality or UI for of the application. In all cases, except the simplest, the porting process will not amount to a widget-for-widget swap.

The experimentation will allow the developers to learn faster than they would otherwise learn from a book or from examples. Teams of developers can learn from each other and discover Qt features and capabilities. All this work will prepare you to tackle large applications and curb your urge to dive in and start modifying the code without a plan.

Port or Redesign?

The porting effort can have a large impact on the architecture of a complex application. It is important that such applications be re-architected instead of employing a strategy of widget-by-widget conversion. In a lot of cases this might mean throwing away or rewriting large portions of the code that are not GUI specific. Instead of looking at it as wasted effort, take the opportunity to modernize the architecture of your application. Perhaps you can make use of concepts and features that were not available when the application was originally written. You can now incorporate features requested by customers that you could not provide earlier because of the rigidity of the architecture.

From our experience we have noticed that Qt almost forces you write code in an object-oriented fashion. The signals and slots mechanism frees you from having to use static methods and typeless data transfer. Another design aspect often talked about in GUI development circles but seldom implemented correctly is the Model-View-Controller (MVC) architecture. Separating the visual representation (View) from the data (Model) has numerous advantages. There is more work upfront but gains in productivity and cleanness of design outweigh the effort. ICS's Table widget (qicstable) is a very good example of the use of MVC architecture in a real widget. Qt is incorporating the technique in few of the data-driven widgets to provide multiple views and responsive GUIs using a common data model abstraction for future releases.

Gradual migration using the Qt Motif Extension

Qt provides some extra widgets and glue code to help ease the transition from Motif/Xt based applications to Qt. QMotif initializes Xt and creates an application context. QMotifWidget can be used as a wrapper for creating parent Xt/Motif widgets. QMotifWidget behaves like a QWidget but can be used as parent for other Xt/Motif widgets. Similarly, there is a QMotifDialog class that provides QDialog functionality for Xt/Motif dialogs. Using these classes, you can "wrap" your Motif widgets around Qt widgets thereby allowing you to focus on first getting the application infrastructure converted to Qt and then tackle the widgets.

Another route to follow is to use Qt for all new functionality in the application. This can be achieved by passing the application context of the Motif application to the QMotif constructor. You can then create a QApplication object. Once the QApplication object is available, Qt widgets can be created at will. This technique is good for adding Qt based "plugins" to your Motif application and can also be used with third party applications that

don't support Qt directly. For example, in Maya (a high end 3D application) you can get hold of the main view widget and use the XtWidgetToApplicationContext call to get the application context, allowing you to create richer and more responsive user interfaces than those possible with Maya's scripting language MEL.

Porting Strategies for Applications

Hand-written GUI

Hand written GUI usually means that the layout of widgets is very dynamic. These involve custom layout or constraints that would be hard to implement using a GUI builder. Qt provides a layout engine that is very versatile. In addition to the many standard layouts, custom layouts can be implemented as long as the algorithm can be cleanly described and abstracted. If widgets are created/deleted and mapped/unmapped in the GUI as a result of user interaction, you have to make sure that your widget hierarchy in Qt is correctly maintained and layouts are properly parented (recall that layouts are independent of widgets) for proper destruction of widgets.

GUI-builder generated

These kinds of applications, such as those built using BXPRO from ICS, are easier to convert and your vendor may be able to provide you with special assistance with the conversion process. Most GUI builders save the GUI representation in a simple format that is easy to parse. It is not very difficult to write a script that can parse this description and can convert it to the XML-based format that Qt can use with its GUI builder, Designer. Some GUI builders are capable of generating source code for different kinds of applications from the same description; you may want to contact your vendor to see if they can add Qt as one of the formats.

C-based

If your Motif-based application is written in C, a major part of the effort required to port it to Qt will be expended in converting the application to C++. It is recommended that programmers take a course in object-oriented design using C++ to effectively use C++ as an expression of object-based design rather than using the C++ compiler but retaining C-like code. GUI toolkits are well suited for learning object-oriented principles because of the way widget hierarchies naturally fall into an object oriented design. Event handling and widget drawing use well known patterns from object-oriented design. Qt toolkit comes with all the source code. It is well written and well documented code. Many beginners utilize patterns and best practices found in the Qt source and code examples in their own applications.

Graphics heavy

More and more applications are making heavy use of graphics and colors not just for display of data but for the user interface as well. The increasing power and dropping costs of high quality graphics hardware has encouraged adoption of pleasing and responsive interfaces. At the basic level, Qt supports drawing of text, pixmaps and advanced 2D

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 • • www.ics.com • Phone: 617.621.0060

graphics primitives such as points, lines, arcs, chords, ellipses, cubic Bezier curves and polygons. These methods can be used to draw on any supported device, a widget, a printer, a pixmap or a picture. The functionality provided is similar to that available using the Xlib graphics calls.

For more advanced users, Qt also provides a Canvas module. A QCanvas can contain a number of object-based QCanvasItems that can be drawn, manipulated, animated using a sophisticated 2D transformation coordinated system. The Canvas module is much easier to use than the Motif XmDrawingArea and the supporting Xlib drawing calls.

The OpenGL module that is included in the professional distribution of the Qt toolkit provides support for hardware-accelerated advanced 2D/3D graphics. The OpenGL API, by design, is independent of the underlying windowing system or operating system. The Qt OpenGL module provides the same windowing and operating system independence for the GUI needed to support an OpenGL-based application on all platforms. It provides a wrapper for the GLX extension on the X Window system, WGL on Microsoft Windows, and AGL on the OSX system.

X/Motif users will find this module a godsend. It streamlines the calls to specify OpenGL configurations and visuals and allows Qt pixmaps to be used for off-screen rendering. The Qt OpenGL widget also supports efficient bit-mapped text rendering, automatic management of overlay visuals and texture image generation

Converting a X/Motif-based OpenGL is easier if you have been using the GLwDrawingAreaWidgetClass. The QGLWidget api is somewhat similar as shown in the following table. The OpenGL part obviously remains unchanged.

<table>
<tr><th>Motif GL Widget (GLwDrawingArea)</th><th>Qt GL Widget (QGLWidget)</th></tr>
<tr><td><pre>
             . . .

Arg args[10];
int n;
Widget parent;
Widget glw;
      . . .
n = 0;
XtSetArg(args[n], GLwNrgba, TRUE);
n++;
glw = XtCreateManagedWidget("glw",
     GLwDrawingAreaWidgetClass,
     parent, args, n);
XtAddCallback(glw,
 GLwNexposeCallback, exposeCB,0);
XtAddCallback(glw,
 GLwNresizeCallback, resizeCB, 0);
XtAddCallback(glw,
 GLwNginitCallback, ginitCB, 0);
// input callback
      . . .
</pre></td><td><pre>
class OGLWidget : public QGLWidget
{
  Q_OBJECT
public:
..
..
protected:
  virtual void initializeGL();
  virtual void paintGL();
  virtual void resizeGL(int w, int
h);
..
..
};
</pre>
Note: the input callback and the expose callback functionality is provided by standard QWidget event handling.</td></tr>
</table>

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

If you are using X directly instead of Motif, i.e. using XCreateWindow with visual, depth, Colormap and EventMask information to create the OpenGL window, the design and code can be significantly simplified by using the QGLWidget. Use the QGLFormat class to create custom GL display formats and the three virtual methods shown in the table above to handle all drawing and update. For portability reasons, QGLWidget uses 1.2 version of the GLX api. This means that pbuffers and the newer GLX frame buffer configuration (FBConfig) structures available in version 1.3 are not directly supported in a platform independent way. You will have to allocate them directly and manage sharing and switching contexts properly. The newer graphics cards also support RGB overlay visuals instead of the color indexed overlays. You should override the default overlay format (which is assumed to be color index) by using the static method, QGLFormat::defaultOverlayFormat() in this case.

Custom widgets

Writing custom widgets in Motif has always been very complex. The knowledge, experience and code required to create a custom widget in most cases is almost not worth the effort. The Motif widget writer's guide says, "…Complex widgets may easily require 10,000 to 20,000 lines of source code…We do not recommend that you write all this code from scratch…" In Qt, you use normal C++ inheritance mechanism to subclass and extend the functionality of existing widgets.

If you have existing custom Motif widgets that you wish to port to Qt, it is recommended that you focus on the functionality and the design rather than attempt to convert it line by line. A custom widget in Motif normally is coded in three files, the public and private header files and the source file. The public header file roughly corresponds to the public API for the widget (usually the constructor and methods to modify the widget's state and data). The resources of the Motif widget can be converted to attributes in Qt using the Q_PROPERTY macro. The Q_PROPERTY macro, allows one to define a resource, its type, get/set/reset methods and whether the property is stored, usable by a GUI designer or by a scripting engine like QSA.

The rest of the steps normally followed in the writing of a custom Motif widget's private header file: Defining inheritable methods, Defining Widget Class/Instance Part, Declaring the Full Class/Instance Record is basically a way of doing C++ style inheritance with C. You get most of this functionality for free in C++ by just using the appropriate base class.

In Motif, custom widgets derived from Manager widgets have the added burden of handling geometry and event requests from the children. Qt keeps the layout management procedures separate from the widget and any widget can be made a manager widget by adding a layout to manage its children.

The table below gives a breakdown of a typical custom widget implementation in Motif and how Qt can be used to achieve similar functionality.

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

Description	Motif	Qt
Intialize	ClassInitialize, Initialize, Create	widget constructor(s)
Re-render the widget	Redisplay	update(), repaint()
Draw the widget	DrawVisual, DrawShadow	paintEvent, QPainter
Optimal widget size	WidgetSize, VisualSize	sizeHint,
Accommodate widget	Resize	adjustSize
Changes in resource	setValues	public set/get methods
Event Handling	Callbacks/actions/translations	events, signals & slots
Geometry mgmt.	geometry_manager	QLayout
Even propagation	parent_process	automatic, event filter

Get Help

Porting large applications to use a new toolkit can be a daunting task. It is very likely that you will end up making substantial changes to not just the graphical user interface but also to the architecture of your application. Although the primary motivation for switching to Qt is usually to support cross-platform deployment of the application, invariably you will want to take advantage of new features offered by the toolkit and the opportunity to do so with the redesign of the underlying architecture.

If, because of time or resource constraints, you are unable to proceed with the porting effort on you own, ICS is more than happy to provide consulting and application development services. ICS also offers various levels of training courses to bring your engineering staff up to date on their GUI development and Qt skills.

Qt Training, Advanced Qt Components and Testing Tools

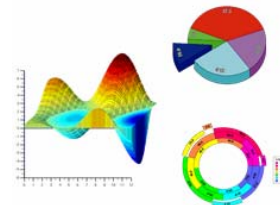
As Trolltech's preferred training partner for North America, ICS provides public and customized on-site training on Qt. See details at
http://www.ics.com/services/training/?cont=QTtraining

ICS also provides a growing selection of advanced components and testing tools for Qt. Some of our products including:

QicsTable™

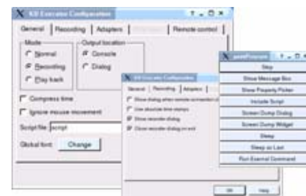
Power to easily manipulate the largest data sets and with all of the display and print functions you need to satisfy even the most demanding end-users, this advanced Table component comes with a comprehensive API that makes programming a snap. MVC architecture assures that you application is easy to build, modify, and maintain.
Free trial at
http://www.ics.com/qt/qicstable/?cont=getqicstable

GraphPak™



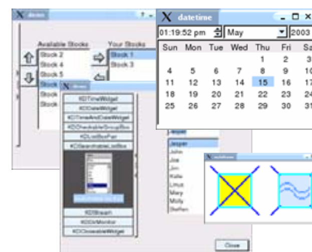
A collection of powerful charts and graphs that make it easy to visually present complex data.
Free trial at
http://www.ics.com/qt/graphpak/?detail=downloadA.html

KD Executor™



A true cross-platform testing harness that makes it easy to fully test your Qt applications.
Free trial at
http://www.ics.com/qt/kdexecutor/?cont=download

KD Tools™



The indispensable library of widgets, containers, drawing objects, and non-GUI classes that speed the creation of world class applications
Free trial at
http://www.ics.com/qt/kdtools/?cont=download

Copyright © 2004-2006 Integrated Computer Solutions, Inc. • 54 B Middlesex Turnpike, Bedford, MA 01730 •
• www.ics.com • Phone: 617.621.0060

KD Gantt™

A graphics library add-on to Qt that eliminates the need to write custom charting code by providing all of the graphics, layout, linkage, and end user facilities needed to create Gantt charts.
Free trial at
http://www.ics.com/qt/kdgantt/?cont=download

Motif to Qt Migration Assessment

Motif→Qt

ICS is also the world's foremost expert in Motif! Let our experts in Motif and Qt guide you through the migration process.
Contact us at sales@ics.com to discuss your needs.

About ICS

Driven by the belief that the success of any software application ultimately depends on the quality of the user interface, Integrated Computer Solutions, Inc., The User Interface Company™, of Bedford, MA, is the world's leading provider of advanced user-interface development tools and services for professional software engineers in the aerospace, petrochemical, transportation, military, communications, entertainment, scientific, and financial industries. Long recognized as the platform of choice for visually developing mission-critical, high-performance Motif applications, ICS' BX series of GUI builders has recently been expanded to provide a complete line of tools that accelerate development of Java™. ICS is the largest independent supplier of add-on products to the Qt multi-platform framework developed by Trolltech. Supporting the software development community since 1987, ICS also provides custom UI development services, custom UI component development, training, consulting, and project porting and implementation services. More information about ICS can be found at http://www.ics.com

Integrated Computer Solutions Incorporated

The User Interface Company™

54 B Middlesex Turnpike
Bedford, MA 01730
617.621.0060
info@ics.com

www.ics.com