

# Galaxy Replacement Investigation

---

**Kenny Gruchalla and Brian Eyster**

## 1.0 Introduction

Galaxy is the graphical user interface API currently used by RiverWare. Galaxy provides all the user interface components (e.g., dialogs, buttons) used to interact with a RiverWare model. Galaxy is a commercial product and its parent company, Visix, declared bankruptcy in the late 1990s. Galaxy was purchased by another company, but development and support of the toolkit has basically ceased. As a result, Galaxy's user base is rapidly shrinking. CADSWES believes Galaxy is becoming a liability to RiverWare, as it is conceivable that in the near future Galaxy will not be supported for new operating system upgrades. In an attempt to be proactive, CADSWES intends on incrementally phasing out and removing Galaxy from Riverware. This document discusses the selection criteria, replacement candidates, and reasoning for the final selection of the Qt graphical user interface API as the replacement for Galaxy.

## 2.0 Selection Criteria

There were several selection criteria that were considered when evaluating new GUI toolkits for RiverWare.

- The toolkit **must** support our target platforms.  
At a minimum Solaris and Windows must be supported. Ideally, the GUI toolkit should be well-supported on all Unix platforms (including Linux), and Windows.
- The toolkit **must** peacefully coexist with Galaxy.  
Galaxy will have to be replaced gradually over time. The chosen toolkit will have to coexist with Galaxy during the replacement process. The toolkit must be capable of being integrated with Galaxy, or the toolkit must provide an IPC (Interprocess Communication) model to coexist outside of the current framework.
- The toolkit **must** have a substantial user-base.  
The size and the breadth of the user-base is important as this gives us an indication of the toolkit's longevity.
- The toolkit's cost and licensing **must** be non-restrictive.  
The cost and licensing of the toolkit needs to be addressed. Whether the licensed distribution includes the source code, should also be considered. Open source development models are preferred as they are not tied to the fate of a single company or developer.
- The toolkit **must** have solid documentation.  
The quality and quantity of the toolkit's documentation is paramount. Toolkits with published third party documentation (e.g., O'Reilly) will be preferred.
- The toolkit **must** have GUI builder support.
- The toolkit **should** be Object-Oriented.  
An object-oriented toolkit is preferred over a procedural API.
- The toolkit **should** have advanced widget support (e.g., plotting, spreadsheet capabilities, workspace)
- The toolkit **should** support the development of custom widgets.

---

## 3.0 Candidates

An exhaustive search and review was performed for all GUI toolkits that would support both Windows and Solaris platforms. All the candidates had strengths and weaknesses. The four most viable candidates were Java, Qt, Tk, and wxWindows.

### 3.1 JAVA (AWT/Swing)

Toolkit Name	Java (AWT/Swing)
Homepage	java.sun.com
License	Sun Microsystems BCLA (Free)
Language	Java
Platforms	Most Platforms
Userbase	Ubiquitous
Object-Oriented	Yes
Documentation	Mountainous (Thousands of published books, several journals, online docs, ...)
GUI Builder	Several
Widget Set	Extremely Rich (with many OpenSource and Commercial extensions)

Java is a full-featured high-level programming language. The Java language is both compiled and interpreted. The compiler translates a program into an intermediate language called Java bytecodes. Java bytecodes are platform-independent and can be interpreted by any platform with a Java Virtual Machine (Java VM). For the purposes of RiverWare, we are only interested in Java's GUI front end (i.e., Swing and AWT). Of all the current major alternatives to Galaxy, Java is one of the more difficult to evaluate. This is largely due to clouds of distracting hype. One thing can be said with confidence, Java is here to stay. The two major issues with Java as RiverWare's front end are Java's speed and the integration of Java with our current code.

Java is an interpreted language and cannot be expected to run as quickly as compiled languages with similar capabilities (e.g., C++). Indeed a common complaint among commercial Java developers is Java's speed. However, we are only interested in using Java as a front-end to RiverWare. A functionally thin GUI front-end should not exhibit any substantial speed problems.

Integrating Java with our simulation code is the larger of our two issues. Java can be integrated directly with C/C++ (or other languages) using the Java Native Interface (JNI). JNI allows java code to call C/C++ code and vice versa. This mechanism could be used to integrate a Java GUI framework into RiverWare, however JNI is tedious (read tricky and kludgy). The general advice in the Java community is to avoid JNI altogether and opt for an IPC solution. The IPC alternatives require building an additional encapsulation layer, but the solution would be much cleaner and more sophisticated.

For the implementation of the RiverWare GUI in Java, the only reasonable choice would be to split RiverWare into two (or more) processes with the computational model server implemented in C++, and the GUI client implemented in Java with an IPC communication layer between the two. A number of different technologies exist for building distributed client/server applications, but for a multi-language, multi-platform application the clear choice is CORBA. CORBA is a specification that defines how distributed objects can interoperate. CORBA objects can be written in any programming language supported by a CORBA software manufacturer such as C, C++, Java, Ada, or Smalltalk. These objects can also exist on any platform that is supported by a CORBA software manufacturer. This

---

means that we could have a Java application running under Windows 95 that dynamically loads and uses C++ objects stored across the Internet on a Unix Web server.

JNI does not provide a foundation for a feasible solution, it is far too complicated. However, a distributed CORBA solution is feasible. CORBA does have broad industry support and provides a scalable elegant solution. The primary disadvantage is the additional development overhead of building a communication layer for all the GUI interaction with the model. This work would be substantial. First a IPC framework would have to be developed for RiverWare. Then messages for all communication between the GUI and simulation process would have to be built and passed along the IPC framework. Integrating Java with RiverWare will require a major software development effort, above and beyond the redesign of the GUI front end.

### 3.2 Qt

Toolkit Name	Qt
Homepage	www.trolltech.com
License	Commerical/Open Source (depending on version and usage)
Language	C++
Platforms	Unix, Windows, MacOS
Userbase	Moderate (open-source and commercial developers)
Object-Oriented	Yes
Documentation	Moderate (several published works, and 2500 pages of "in-house" documentation)
GUI Builder	Yes
Widget Set	Extremely Rich (sophisticated and visually attractive)

Qt is a high quality commercial widget set, in development since 1995. Qt is used as the GUI API for the KDE window manager (widely used on many Linux distributions). KDE users account for the largest percentage of Qt's userbase. Qt also claims to have such commercial customers as: AT&T, IBM, Xerox, and NASA.

In contrast to most other C++ toolkits, Qt is an "emulating" cross-platform toolkit. Qt utilizes only the low-level drawing/event layer of the native platform, and then draws all the widgets themselves. This is a potentially more efficient approach than the "thin wrapper" approach. It also shields Qt applications from changes in the native API. The drawback of this approach is that the look-and-feel of each platform must be implemented by Qt. Fortunately, virtual functions within the style class hierarchy allow users to implement style changes themselves, if needed.

Qt uses a unique licensing strategy. Qt comes in three editions, the Free Edition, the Professional Edition and the Enterprise Edition. The Free Edition is free for developers of free open-source software on Unix/X11 only. Meanwhile, the handsomely priced Professional and Enterprise Editions (which include 3D visualization utilities and some more sophisticated widgets) are required for all Windows developers (commercial and open-source) and developers of commercial applications on Unix/X11. The pricing is approximately \$2100 (\$2960 for enterprise edition) per developer and a \$650/\$950 per developer yearly maintenance fee. Trolltech requires a license only for developers who are actively doing GUI development. So non-GUI developers do not need a license to build the Qt GUI code.

Qt offers the richest and one of the most mature GUI APIs that we evaluated. From a programming perspective Qt offers the easiest and quickest solution. We have proved through a prototype that Qt dialogs and Galaxy dialogs can coexist in the same process. We were able to add a simple Qt dialog to RiverWare and successfully build and run RiverWare with both Galaxy and Qt dialogs running simultaneously. Using this strategy will allow us to incremen-

---

tally phase out and replace Galaxy with Qt, one dialog at a time. It should be noted that Qt could also be integrated with Galaxy via a CORBA solution, though unlike Java this is not the only possible integration strategy.

There are some minor complaints about Qt in the development community. Qt does not use templates or the standard template library, Qt instead supplies custom containers. Qt's unique *signals-slots* event system uses preprocessor macros and an additional level of code-generation, so Qt cannot be viewed as a pure C++ library. But these macros enhance the readability of Qt code and help implement a type-safe callback system. The other common complaint is Qt's lack of namespaces. Beyond these technical limitations, Qt's largest drawback is the question of its longevity.

Qt is developed by Trolltech, a private company based in Norway. Considering the plight of Galaxy, which was developed by a private company, the longevity of Trolltech and Qt is an important concern. Qt differs from Galaxy, however, in that a Qt licensee receives the complete source code to all the Qt libraries and tools. If Trolltech were to go out of business, the source code could continue to be developed by the existing user base.

### 3.3 Tk

Toolkit Name	Tk
Homepage	www.tcltk.com
License	BSD
Language	C API (typically used with scripting languages)
Platforms	Unix, Windows, MacOS
Userbase	Ubiquitous in academic sectors, sparse in commercial sectors
Object-Oriented	No
Documentation	Significant (almost a hundred published books and many online resources)
GUI Builder	Yes (but it is clumsy and difficult to use)
Widget Set	Basic

Tk is a GUI C API. It was introduced in 1990 as a GUI toolkit for Tcl. Tk has since found an audience with many scripting languages (e.g., Tcl, Perl, Python). Tk is primarily used with scripting languages, and has scripting look-and-feel. Although geared toward scripting languages, Tk can be used directly from within C/C++ programs.

Tk can be integrated with a C++ code in one of three ways:

- C++ programs can use Tcl/Tk's C interface by calling the Tcl/Tk library routines directly. The Tcl/Tk C interface is difficult to use. Calling the interpreter from C appears to be a dull chore and is widely discouraged.
- Embedded Tk (ET) allows Tk to be inlined within a C program. The inlined code is then converted to C code with Tk preprocessor, *et2c*. This is the preferred method. However, this is messy, and promises to become completely unmanageable in systems with complex graphical user interfaces (i.e., RiverWare).
- The third option is to manage the GUI as a separate Tcl/Tk process, which communicates with the C/C++ program via IPC. As mentioned in the Java section, building an IPC framework into RiverWare would be a substantial effort.

Tk's major drawbacks are its lack of sophistication and the difficult RiverWare integration path.

---

### 3.4 wxWindows

Toolkit Name	<b>wxWindows</b>
Homepage	www.wxwindows.com
License	Open Source
Language	C++
Platforms	Unix, Windows, MacOS
Userbase	Fledgling (but on the increase)
Object-Oriented	Yes
Documentation	Limited (no published documentation, “inhouse” documentation only)
GUI Builder	Yes (but not very sophisticated)
Widget Set	Complete (spreadsheet widget and basic plotting widget provided)

wxWindows was first developed in 1992 and development uses an open-source model managed through a SourceForge project. The user base is small, but it has been steadily increasing from the early nineties. wxWindows is a set of open source libraries that allows C++ applications to compile and run on several different platforms, with minimal source code changes. wxWindows provides a common abstraction layer above platform specific GUI libraries (i.e., Win32, GTK+, and Motif). wxWindows provides a common API for GUI functionality, as well as functionality for accessing some commonly-used operating system facilities, from copying and deleting files to socket and thread support.

In order to support a large number of compilers, wxWindows does not use templates, nor the standard template library. wxWindows instead provides its own container classes (i.e., strings, arrays, lists, hash tables, ...). A common complaint about wxWindows is its speed. The extra abstraction layer is noticeably slower than using the native libraries.

The sample programs provided with wxWindows are fairly buggy when built on top of Motif. X-resource are regularly violated (e.g., “X Error of failed request: BadDrawable”) leading to core dumps. The sample programs may be poorly written or this may be a reflection on the library itself. Significant differences in appearance were observed between the sample programs built on the Windows platform versus the samples built on the Unix/Motif platform. wxWindows appears to be more stable and mature on the Windows platform.

wxWindows should integrate directly with Galaxy which offers the easiest and quickest solution. There may be some risk integrating wxWindows with Galaxy, as no prototype has yet been built to test the feasibility. wxWindows could also be integrated with Galaxy via a CORBA solution (as discussed in the Java segment).

wxWindows largest drawbacks are its immaturity, small user base, and relative instability on Unix platforms.

### 4.0 Lesser Candidates

Several other toolkits were examined which we were able to quickly dismiss, on the basis of a failure to meet one or more of our major selection criteria.

---

#### 4.1 Coral

Toolkit Name	Coral
Homepage	<a href="http://www.imonk.com/coraldoc.html">www.imonk.com/coraldoc.html</a>
License	Open Source (Artistic License)
Language	C++
Platforms	Unix, Windows
Userbase	Nonexistent
Object-Oriented	Yes
Documentation	Nonexistent
GUI Builder	None
Widget Set	Minimal

Coral is a collection of C++ libraries, including GUI and non-GUI libraries. Coral is very academic in nature with a nearly nonexistent user base, no documentation, and a very minimal widget set.

#### 4.2 FLTK

Toolkit Name	FLTK (The Fast Light Toolkit)
Homepage	<a href="http://www.flTk.org">www.flTk.org</a>
License	Open Source (LGPL)
Language	C
Platforms	Unix, MacOS, Windows
Userbase	Minimal
Object-Oriented	No
Documentation	Nonexistent
GUI Builder	Yes
Widget Set	Basic

FLTK (pronounced "fulltick") is a multiplatform graphical user interface toolkit. FLTK is C library originally designed to interact with the XForms C toolkit. There is no support for networking, printing, tables, or text editing, and the library only provides a few of the most basic widgets. The single event handling function inside each widget is passed an integer value, and all other parameters (e.g. mouse position, key that was pressed) must be requested by calling separate functions. This turns the event parameters into global data, which is totally unnecessary and prevents the event loop from ever being re-entrant.

FLTK was eliminated due to its non-object-oriented design (C language), nonexistent documentation, and minimal user base.

---

### 4.3 FOX

Toolkit Name	<b>FOX</b>
Homepage	<a href="http://www.fox-toolkit.org">www.fox-toolkit.org</a>
License	Unknown
Language	C++
Platforms	Unix, Windows
Userbase	Small
Object-Oriented	Yes
Documentation	Nonexistent
GUI Builder	None
Widget Set	Basic

FOX is a C++ based Toolkit for developing Graphical User Interfaces. FOX was ruled out due to its minimal user base, nonexistent documentation, and simplistic widget set.

### 4.4 GTK+

Toolkit Name	<b>GTK+ (The GIMP Toolkit)</b>
Homepage	<a href="http://www.gtk.org">www.gtk.org</a>
License	Open Source (LGPL)
Language	C
Platforms	Unix, Windows (Windows port in progress)
Userbase	Large rabid open source community
Object-Oriented	Yes
Documentation	Significant (several published works, rich "in-house" documentation)
GUI Builder	Yes
Widget Set	Sophisticated

GTK+ is a multi-platform toolkit for creating graphical user interfaces. GTK+ is Qt's sister toolkit, used as the backbone of GNU's GNOME desktop. GTK+ is clearly suitable for projects ranging from small projects to complete application suites. Unfortunately, GTK+ does not yet support the windows platform. A windows port is in progress, but it could be quite sometime before GTK+ is fully supported in a Windows environment.

---

#### 4.5 XVT DSC++

Toolkit Name	<b>XVT DSC++</b>
Homepage	www.xvt.com
License	Commerical
Language	C++
Platforms	Unix, Windows, MacOS
Userbase	Small
Object-Oriented	Yes
Documentation	Limited (no published documentation, "inhouse" documentation only)
GUI Builder	Unknown
Widget Set	Minimal

XVT commerical is client/server GUI software toolkit developed by small company, Providence Software Solutions. With the host of open source GUI solutions now available and minimal user base, XVT is considered not likely to survive.

#### 4.6 ZINC

Toolkit Name	<b>ZINC</b>
Homepage	www.windriver.com/zinc
License	Commerical
Language	C++
Platforms	Unix, Windows
Userbase	Nonexistent
Object-Oriented	Yes
Documentation	Limited (no published documentation, "inhouse" documentation only)
GUI Builder	Yes
Widget Set	Minimal

ZINC is a commercial GUI software toolkit developed by small company, WindRiver. Like the XVT toolkit, the survivability of ZINC is highly questionable.

### 5.0 Selection

wxWindows' propaganda made it look very promising. Unfortunately after building several small prototypes, wxWindows' immaturity and instability were apparent. The immaturity and instability of the package were the sole reasons for its dismissal.

Tk is stable, well-documented, and widely used. Tk is primarily used in smaller applications, so there is some apprehension using Tk in a project of RiverWare's scale. Tk also has no distinct advantages over Java or Qt, when comparing Tk's sophistication against these other two, Tk is clearly inferior.

---

Comparing Java and Qt is more difficult. Java's biggest advantage is its promise of longevity. Java's biggest disadvantage is the difficulty of integration with the existing RiverWare code. The integration effort required to use Java as the front-end for RiverWare would be substantial. In addition to rewriting every GUI component in RiverWare, an IPC framework would need to be developed and then every action between the model and the GUI would have to be dealt with in this IPC layer. RiverWare was not designed with a multiple process client/server model in mind, so significant areas of RiverWare would need to be redesigned to effectively build an IPC framework capable of supporting a Java front-end. This would require a substantial effort before any of the Java GUI components could be developed.

Qt is a very sophisticated and visually attractive toolkit. Qt is object-oriented C++, which promises an easy integration with existing RiverWare code. To verify this, a simple test Qt dialog was successfully added to RiverWare. On the Galaxy-user-mailing list several former Galaxy users have shared very positive experiences with converting a Galaxy application to a Qt application using a similar strategy. It is worthwhile to note that Galaxy does not provide a C++ interface, and in the past a significant amount of time and effort has been expended to integrate the object-oriented C++ RiverWare code with the Galaxy C libraries. Moving to an object-oriented C++ toolkit would be a major improvement.

Qt's main disadvantage is the question of its longevity. The availability of the Qt source code combined with a steadily growing user base, however, makes this less of a concern. If Trolltech were to disappear, it seems likely that Qt would continue to be maintained by this base of users.

After weighing the question of Qt's longevity against the enormous task of integrating Java with RiverWare via an added IPC layer, Qt was selected as the replacement for Galaxy as the graphical user interface API for RiverWare.