Development Systems: A Review

Curt Hill
Department of Mathematics and Computer Science
Valley City State University
Valley City, ND 58072
Curt.Hill@vcsu.edu

Abstract

Considerable attention has been given to the choice of programming language in the introductory programming class. In general, learning to program is difficult task, learning a particular language is also difficult, but the mastery of a subset of features of a development environment is the gateway to programming. This is a task that is often overlooked by instructors, yet it poses an obstacle to students. This paper reviews the evolution of development environments over the years.

1 Introduction

Professional developers as well as students spend much of their careers involved in a development environment. This is where they code and where they debug the source for which they are paid or graded. These development environments run the gamut from very simple to very complicated. Like the programming language, the use of the development environment is one of the things that must be mastered. The development environment may be a collection of separate tools that are used individually or integrated into a single system with a single interface. These tools include the language processor, whether a compiler or interpreter, the text editor, and any debugging support or performance analysis. Other tools could play into this as well, but these three are the basics that must be learned.

This paper intends to review the evolution of this environment through the years and examine some of the current offerings.

2 Pre-Interactive

Although it seems hopelessly primitive by today's standards, a brief mention of WATFOR, the Waterloo FORTRAN compiler is required. The systems of that time were batch oriented, the input media was 80 column punch cards and the editor was an IBM Model 29 card punch along with two hands to manipulate the card deck. A common introductory textbook was by Blatt[1], but also see Waterloo[2].

The WATFOR compiler was student-oriented in several senses. It tried to give more understandable syntax error messages than the comparable production compilers. It also performed a number of runtime checking, such as for subscript range errors, uninitialized variables and other common novice mistakes.

3 Interactive Development Environments (IDEs)

Development environments using time-share systems or personal computers added an interactive capability that we now take for granted. Interactive LISP and BASIC systems are among the earliest implementers of these capabilities. These usually featured a built in editor that allowed for creation and manipulation of the program code. Editors such as VI[3], EMACS[4] become common in the 1970s.

3.1 BASIC

Although LISP was developed earlier, BASIC was more mainstream and thus more important in the advancement of development systems. Both were interpreted, but as an imperative, rather than functional language, BASIC placed more emphasis on line numbers and variable values. It was therefore more important to implement break points, single stepping and the display of current variable values. These three features become the standards that every debugger must implement as a minimum.

The BASIC of those days left much to be desired as a language. However, what it lacked as a language it made up for in other ways.

BASIC was well positioned at the advent of the personal computer age. The language could be implemented in the very small amount of memory that the early computers typically possessed. Tiny BASIC[5] could run in as little as three kilobytes of RAM. Although these machines were slow even by the standards of those days, a BASIC program running alone on a micro-computer compared reasonably to that of time-sharing machine. Both the Apple II and the original IBM PC had BASIC in ROM as a standard feature.

Another advantage that BASIC possessed, was a powerful friend. The first product of Bill Gates and Paul Allen was BASIC for the Altair[6]. This collaboration become Microsoft, a company that continues to maintain the language in their Visual Basic line, which will be discussed later in this paper.

3.2 Turbo Pascal

Personal computers became faster and possessed larger amounts of memory. Several other languages became available. Pascal was represented by UCSD Pascal as well as a version by Microsoft. Yet the product that caught the industry by storm was Turbo Pascal by Borland. It had a very fast integrated compiler, a full screen editor and ran on early systems with typically 64K of memory. The editor was sensitive to the language syntax, which was an uncommon feature of the time. The debugger integrated with the editor, so that runtime error would generally send the user to the line of code in question, as would the encountering of a break point.

Various versions of Turbo Pascal executed under CP/M, CP/M 86, DOS and early versions of Windows. The product line continued for many years. Today it is supplanted by the Delphi line from Embarcadero, which acquired the programming tools products from Borland.

3.3 Visual Basic

Turbo Pascal and contemporary products provided considerable power to develop console style programs, but did little to help the developer to produce a Graphical User Interface (GUI) for the program. In general, it was as difficult to generate the graphical user interface as the normal logic of the project. This changed dramatically with the development of the Tripod prototype by Alan Cooper[8,9]. This prototype was eventually sold to Microsoft, which released a later version as the original Visual Basic.

VB, as it is often called, was a not an instant success, but it changed the way GUI programs would be developed. Instead of arranging the widgets on the screen with numerous complicated API calls, the programmer would drag a type of widget from a palette. Sizing and positioning were completed via the drag and drop. Other characteristics could be filled in by forms without touching the actual code. The process was so intuitive that it inspired a whole set of products with drag and drop GUI building capabilities.

In addition to the drag and drop approach that made the GUI design easy, VB introduced a form of event driven programming that is also necessary. As a button caption may be filled in rather quickly, so also the name of the event handler could also be entered. The system then generated the proper subroutine header and connected it to the button. The user then filled in the logic of the subroutine.

Most of the components, such as windows, buttons and text boxes were encapsulated as objects. Things like button captions, sizes and positions were object properties. Methods were used for manipulation and event handling.

4 IDEs of Interest

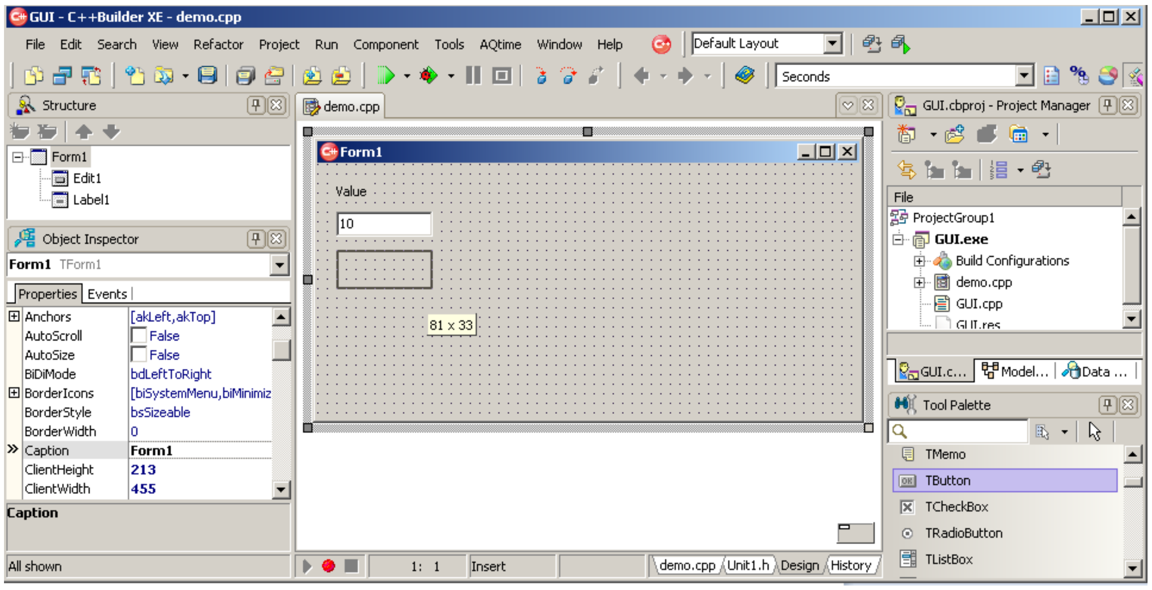
As a classroom instructor of introductory and more advanced programming classes, the author has become familiar with several common IDEs. Some of these are considered now.

4.1 Embarcadero's C++Builder

As the demand for Turbo Pascal and its successors dwindled the project maintainers at Borland realized that they needed to change directions. They introduced the first version of Delphi which used Object Pascal and a drag and drop approach to GUI construction similar to Visual Basic. The C++ group within Borland saw the potential and shortly thereafter created C++Builder. The JBuilder line then moved into the Java area with the same drag and drop approach. Later Borland determined to exit the developer tools market and spun off a company with these products called CodeGear. In turn, CodeGear was acquired by Embarcadero where these product lines continue.

The author[10] has been using various versions of this product in CS1 since 1997. What follows is a short description of how to populate a small window with a few widgets and initialize the event handler.

Initially a new project is created. Widgets from the palette are clicked and then they are sized and positioned with a drag on the form. Once an item is set into the form, additional properties may be set using the object inspector. Figures 1-3 show this process.

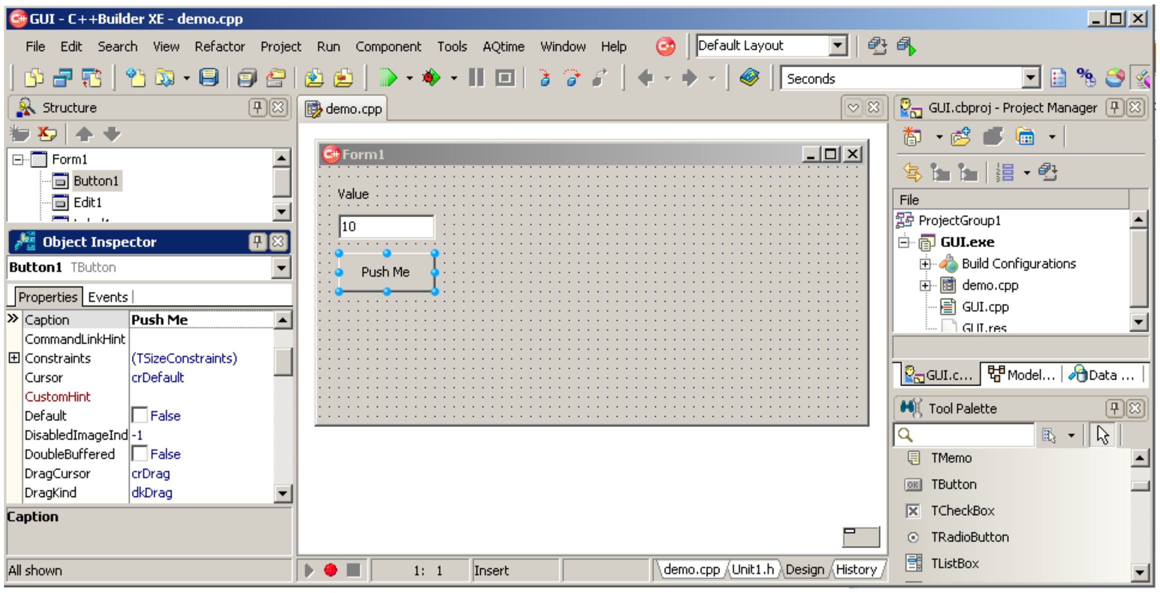


Anchors	[akLeft,akTop]
AutoScroll	False
AutoSize	False
BiDiMode	bdLeftToRight
BorderIcons	[biSystemMenu,biMinimiz
BorderStyle	bsSizeable
BorderWidth	0
Caption	Form1
ClientHeight	213
ClientWidth	455

Figure 1: Dragging a button onto the form.

Consider Figure 1. The lower right hand part of the pane is the tool palette and it may be observed that the TButton is highlighted. In the center pane the button is being sized and positioned. After releasing the mouse and ending the drag, the system supplies a default caption for the button, which is typically ButtonN where N is a sequential count of buttons used.

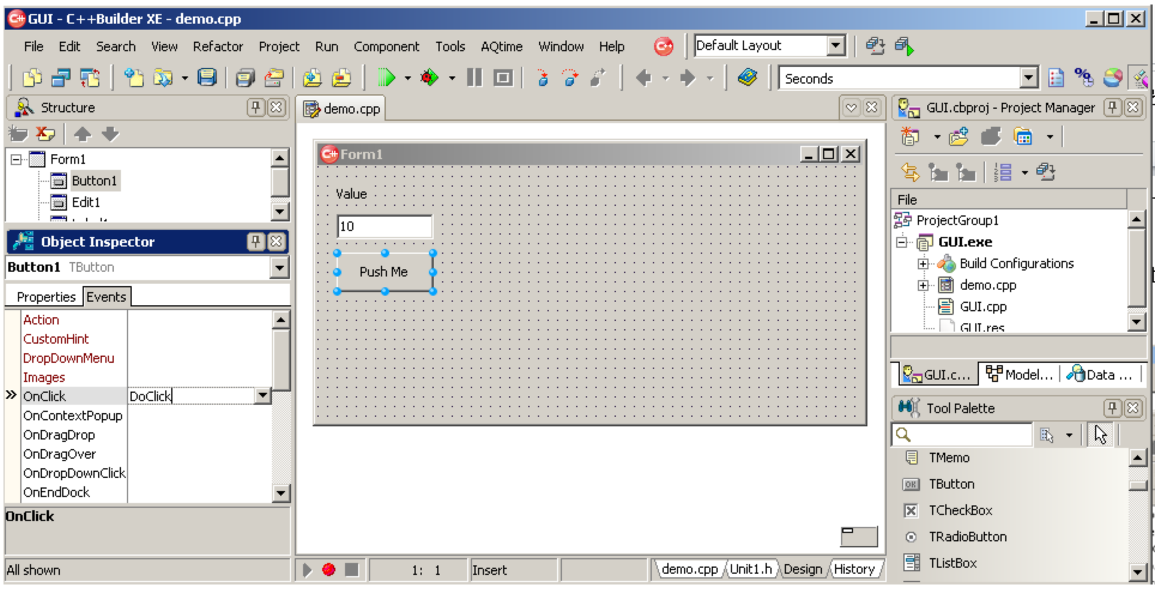
In Figure 2, the caption is being filled in the object inspector on the lower left pane. The display of the button is updated at the same time.



Caption	Push Me
CommandLinkHint	
Constraints	(TSizeConstraints)
Cursor	crDefault
CustomHint	
Default	False
DisabledImageInd	-1
DoubleBuffered	False
DragCursor	crDrag
DragKind	dkDrag

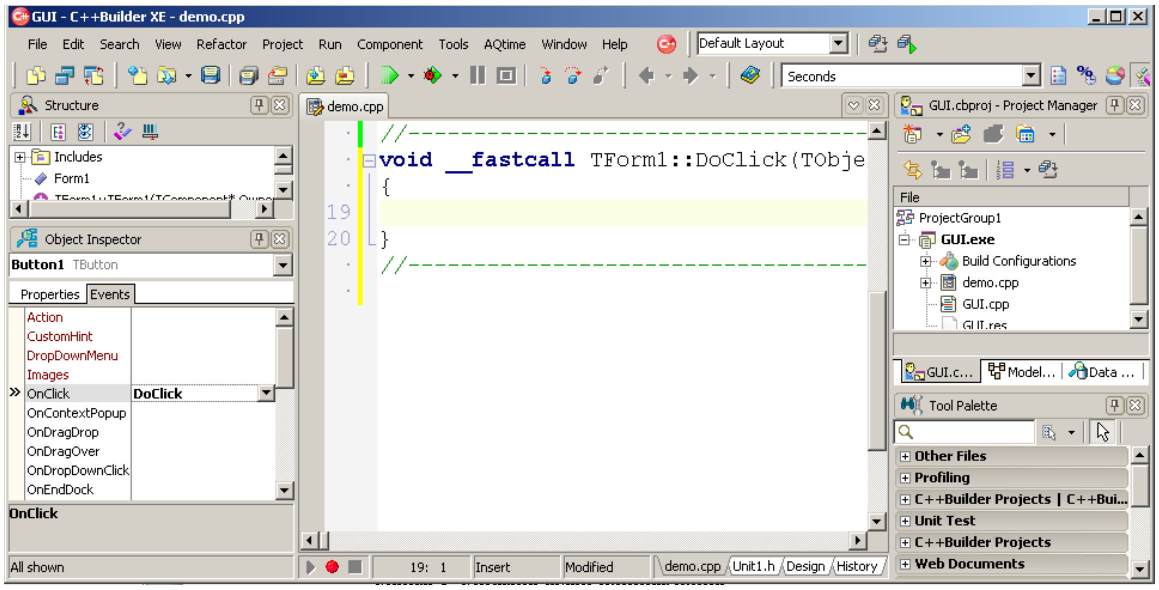
Figure 2: Typing the button caption.

Figure 3 shows the insertion of an event handler. The object inspector's event tab has been clicked. Typing in a valid C name into the OnClick slot defines the name of the event handler. When the name is complete and an enter key is clicked, the IDE shows the code page with the empty method name. This is visible in Figure 4.



OnClick DoClick

Figure 3: Entering event handler name.



OnClick DoClick

```
//-----------------------------------
void __fastcall TForm1::DoClick(TObje
{

}
//-----------------------------------
```

Figure 4: The empty event handler.

There is little overhead for developing GUI based rather than console based programs. The current crop of students is much more familiar with the GUI paradigm than the console paradigm. Thus there is little incentive to teach a full semester of console based programming.

4.2 Microsoft's Visual C++

The procedure for building a GUI in Visual C++ differs in the details, but is insufficiently more or less complicated than what has just been described. The author has actually taught CS1 to different institutions using both Visual C++ and C++Builder. It complicates things slightly.

What makes Visual C++ more interesting is how the language itself has been warped. Since it is part of the Microsoft's .Net initiative is uses a language system with managed classes. These are similar to Java classes with a garbage collector rather than the normal C/C++ memory management. All the widgets are managed classes as well as the common string object. Never the less, for the purposes of this paper the Visual C++ and C++Builder IDEs are equivalent in the common features one might encounter in a CS1 class.

4.3 Borland's JBuilder

JBuilder was developed as a similar product to Delphi and C++Builder but for the programming language Java. It started out as a separate IDE, built completely in Java and generating standard Java. It had the GUI building features of its predecessors as well as the full-featured debugger.

What is unusual about this product is its evolution. In eventually became a product plugin for Eclipse. This shows both the market acceptance of Eclipse as well as its ability to be customized by plugins.

4.4 Eclipse for Java

The Eclipse IDE was originally developed by IBM [11] in the late 1990s. The Eclipse consortium was formed in 2001 as the IDE became open source. This consortium consisted of several companies other than IBM. These included Rational Software and Borland, among others. It has since become the dominant IDE for Java.

The IDE shows all the signs of a modern IDE, seamless integration of a syntax sensitive editor, compiler, debugger as well as a plugin facility that facilitates extensibility. The editor would show lines with syntax errors before compilation. Another nice facility is the generation of blank classes with class header, possibly with an extends and an optional main method header. What it lacked was a free plugin for drag and drop GUI development. However that was not such an omission in the Java language.

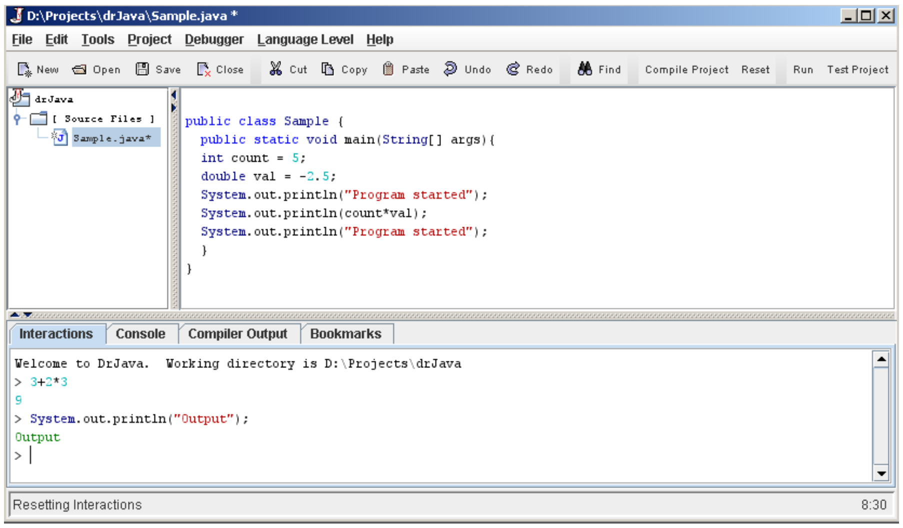
In Visual C++ and C++Builder the GUI information is in a separate file. This file typically compiles into the .res file for Windows. Although Java runs on Windows among other systems, it is fundamentally its own platform. Moreover, the addition and arrangement of widgets is much simpler in Java.

Fundamental to the Java GUI experience is a layout manager. The layout manager controls the placement and sizing of widgets and handles resizing of the window. In particular, this is **not** a feature of the underlying APIs of the system. Thus building the GUI requires adding the widget to the layout manager, possibly with the addition of constraint information. The process typically takes one statement to add the widget and one more to name the event handler. This is much easier than the API approach of native Operating Systems. Thus Eclipse does reasonably well without the drag and drop GUI builder.

4.5 DrJava

Most of the IDEs mentioned in this section are of professional quality or are targeted at professional developers. There are advantages to such an approach. Such environments certainly have all the features that could be use and one does not need to teach all aspects of the environment. However, professional IDEs may also be intimidating. This parallels the argument for introductory language: simplified or marketable?

One of several student IDEs is DrJava[12]. It has the usual elements, a syntax highlighting editor, seamless integration of compiler and full-featured debugging. DrJava’s novel feature is the interactions pane. In this pane, expressions or fragments of programs may be entered and evaluated. Figure 5 shows this facility, in the lower pane.


D:\Projects\drJava\Sample.java *
File Edit Tools Project Debugger Language Level Help
New Open Save Close Cut Copy Paste Undo Redo Find Compile Project Reset Run Test Project
drJava
[Source Files]
Sample.java*
Interactions Console Compiler Output Bookmarks
Resetting Interactions
8:30

```
public class Sample {
  public static void main(String[] args){
  int count = 5;
  double val = -2.5;
  System.out.println("Program started");
  System.out.println(count*val);
  System.out.println("Program started");
  }
}
```

```
Welcome to DrJava.  Working directory is D:\Projects\drJava
> 3+2*3
9
> System.out.println("Output");
Output
> 
```

Figure 5: DrJava.

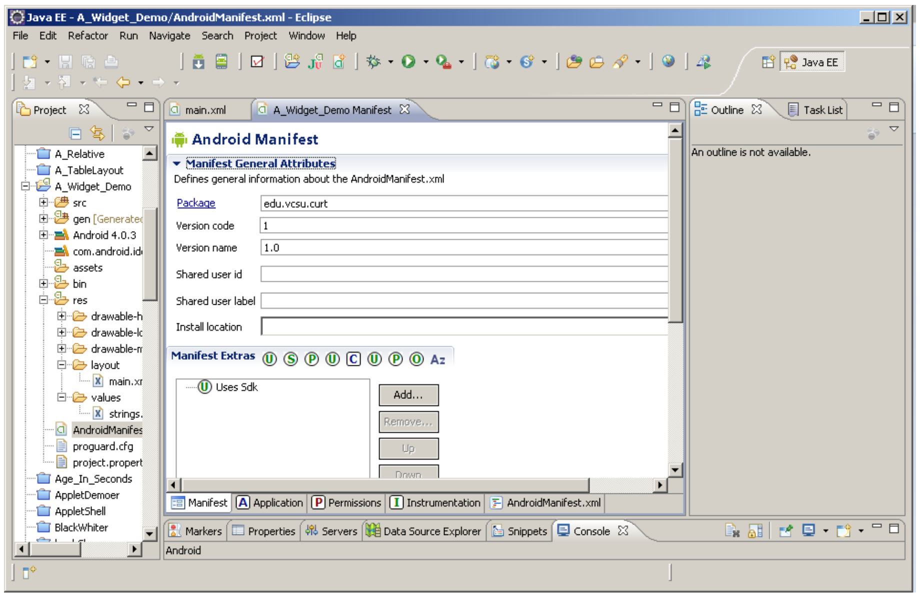
4.6 Eclipse with the Android plugin

The Android system is based upon Linux and targets the comparatively small and weak computers embedded in cell phones and tablets. Such devices are less than ideal developer machines, they often lack real keyboards, have small amounts of memory and slower CPUs. Hence, all development typically occurs on full-featured computers using a variety of common systems. There are several IDEs for Android development but the most common one is Eclipse.

The strength of Eclipse is again shown by the usefulness of the Google plugin for Eclipse in Android application development. A typical Android project is composed of one or more Java programs, several XML files and perhaps several graphics. In addition a project must have access to one or more Software Development Kits (SDKs) with each SDK representing one of several releases of the Android OS. One of these SDKs is needed for each Android Virtual Device (AVD) that emulates the Android device. Most developers are running on Intel based hardware, while most Android devices are using an ARM based processor. The AVD interprets the ARM machine language, which provides a slower, but more accurate emulation.

The plugin provides several changes to the Eclipse system. It provides menu entries so that the SDK and AVD managers are available for use. It uses the normal Eclipse editor for Java, but has a variety of editors for XML. The XML is compiled into code in a Android install file and each type of XML file has a different look in the editor.

The Android Manifest file is mainly used for describing the files to install as well as the level of the OS, permissions needed and hardware required. Figure 6 shows a display of the manifest. Notice in this display, that there are several tabs that describe different aspects of the contents. All graphical displays of XML files using the plugin have the right-most tab which allows direct editing of the XML.



Android Manifest

Manifest General Attributes

Defines general information about the AndroidManifest.xml

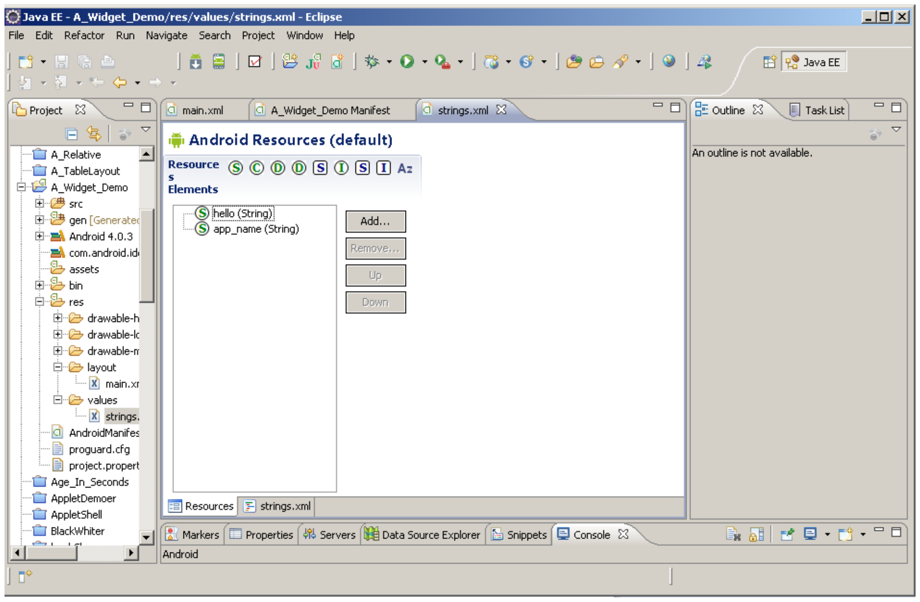
Package	edu.vcsu.curt
Version code	1
Version name	1.0
Shared user id	
Shared user label	
Install location	

Uses Sdk

An outline is not available.

Figure 6: The plugin displaying an Android Manifest file.

Figure 7 shows the string resource XML display. Android projects prefer all strings to be in an XML file in order to facilitate internationalization. Again, the XML may be accessed directly in the right-most tab.



Android Resources (default)

Resource Elements

hello (String)
app_name (String)

An outline is not available.

Figure 7: The string resources XML displayed.

Figure 8 shows a layout XML file, named main. Android programs use a layout manager, like regular Java programs. However, it uses a different set of layout managers and these typically are defined by XML and then compiled into code. The plugin allows a drag and drop approach to build these.

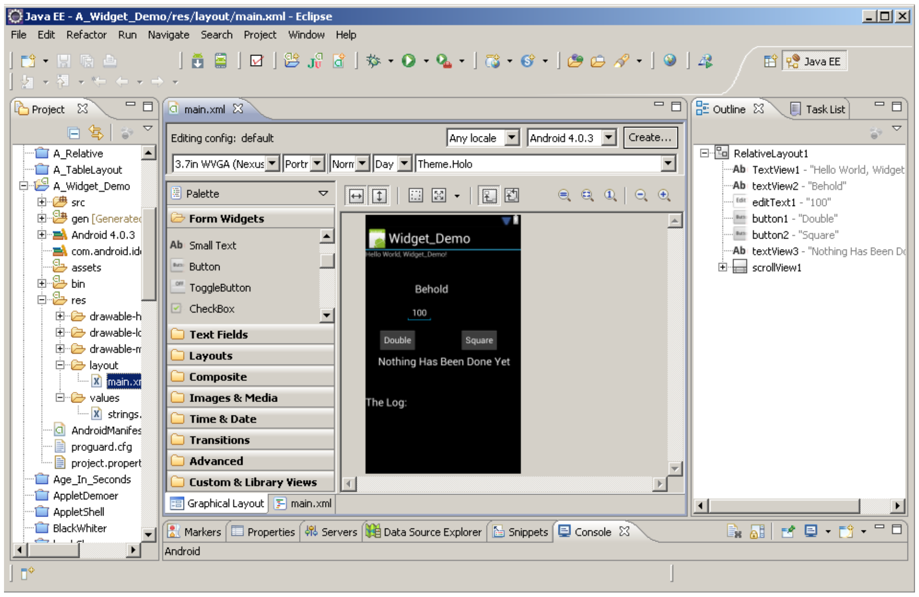


Figure 8: A layout displayed.

Many things in Android projects have a dual nature. Like the layout, it is typically defined in XML. The XML is compiled into Java byte code and then accessed from the Java program. A developer may omit the XML, doing everything from Java, but that is seldom done since it is not as convenient.

Conveniently, the Eclipse debugger has a similar form in Android applications to regular Java programs. The debugger must bridge from the Eclipse environment to the emulator. Yet it provides the features common to most debuggers: break points, single stepping and the display of variable values.

5 Summary

Programming the ENIAC[13] was a matter of plugging patch cords and rotating switches. The improvement in the development environments, like the advances in computation in general, is astonishing. Yet, this is the weak link that is often overlooked in the programming class. It is yet another thing that the student must master.

This paper necessarily could not cover all the important development environments, especially those of platforms such as MacIntosh and Linux/UNIX.

References

[1] Blatt, John M. Introduction to FORTRAN IV Programming. Goodyear Publishing Company. 1971.
[2] University of Waterloo. WATFOR's Silver Anniversary. http://csg.uwaterloo.ca/sdtp/watfor.html Date accessed 15 March 2012.
[3] Wikipedia. vi. http://en.wikipedia.org/wiki/Vi Date accessed 15 March 2012.
[4] Wikipedia. Emacs. http://en.wikipedia.org/wiki/Emacs Date accessed 15 March 2012.
[5] Wikipedia. Tiny BASIC. http://en.wikipedia.org/wiki/Tiny_BASIC Date accessed 15 March 2012.
[6] Wikipedia. Altair BASIC. http://en.wikipedia.org/wiki/Altair_BASIC Date accessed 15 March 2012.
[6] Wikipedia. Turbo Pascal. http://en.wikipedia.org/wiki/Turbo_Pascal Date accessed 15 March 2012.
[7] Embarcadero. Delphi XE2. http://www.embarcadero.com/products/delphi Date accessed 15 March 2012.
[8] Smiley, John. The History of Visual Basic. http://www.johnsmiley.com/visualbasic/vbhistory.htm Date accessed 15 March 2012.
[9] Mack, George. A History of Visual Basic. http://dc37.dawsoncollege.qc.ca/compsci/gmack/info/VBHistory.htm Date accessed 15 March 2012.
[10] Hill, Curt. Experience with GUI programming in lower level classes. MICS 2002, April 5-6, 2002. Cedar Falls, IA.
[11] IBM. A brief history of Eclipse. http://www.ibm.com/developerworks/rational/library/nov05/cernosek/ Date accessed 16 March 2012.
[12] DrJava. About DrJava. http://www.drjava.org/ Date accessed 16 March 2012.
[13] Columbia University. Programming the ENIAC. http://www.columbia.edu/cu/computinghistory/eniac.html Date accessed 16 March 2012.