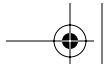


Foreword

I enjoy programming user interfaces in Java, but users don't need to know that—they shouldn't be able to spot that I was using Java from the surface of my application. SWT solves this problem by providing an efficient portable native widget kit. The widget kit is rich enough to build full-fledged applications that look and feel like a native application developed for a particular target platform.

I first used the Standard Widget Toolkit (SWT) several years ago. I was part of the team with the mission to build a Java based integrated development environment for embedded applications that was shipped as the IBM VisualAge/MicroEdition. We used Swing/AWT as the underlying UI technology. We enjoyed being on the bleeding edge of the brand new Swing toolkit. After several months we could finally start to “eat our own dog food” and develop with the IDE we were building. We felt pretty good about what we had achieved! However, our early adopters didn't feel as good as we did... they complained about the performance and most importantly about the fact that the IDE didn't look, feel and respond like a native Windows application. Some of the performance problems were our fault and some of them could be attributed to Swing. The performance problems didn't bother us that much; they could be engineered away over time. What worried us more was the non-native criticism. While we could implement a cool application in Swing that runs on Windows, we couldn't build a true Windows application. Fixing this problem required more drastic measures. Fortunately, there was another team with a lot of experience in portable UI libraries leveraging native widgets for Smalltalk. The next chapter of the story was obvious: transfer these native widget skills and technology to Java and SWT was born. A downside of this decision was that we had to move back several steps and reimplement the user interface of our development environment once more with SWT. This was an intense time with sometimes heated but valuable discussions with the SWT





team. Our reward was that the IDE not only became faster but it also started to look and feel like a native application. In particular, users didn't notice anymore that the application was implemented in Java. The final chapter is public history; based on the lessons learned from the IBM VisualAge/MicroEdition effort, we have moved on to Eclipse and built it on top of SWT.

Since our initial steps with SWT in VisualAge/MicroEdition, SWT has evolved a lot. It now covers more native features with its API and even more importantly, it supports almost all platforms that are popular today. The missing piece was an in-depth book about SWT. I'm glad to see that this gap is now filled. This book comes right from the source—Steve and Mike are the original SWT designers and implementers. They bring amazing depth and broad experience with native widgets to the table. When I hear them telling a horror story about an exotic feature of some native widget, I'm grateful to be on the other side of the API.

This book will give you insights into how the various platform realities shaped particular SWT solutions. You'll find explanations and snippets that illustrate how to program SWT that will support both developers building an eclipse contribution and developers using SWT as a standalone application. Steve and Mike have opened a window into SWT that will profit both beginners and experienced developers!

—Erich Gamma
May 2004

