Sergey Konstantinov
The API

Introduction

Chapter 1. On the Structure of This Book

The book you're holding in your hands comprises this Introduction and three large sections.

In Section I we are to talk about deisgning the API as a concept: how to build the architecture properly, from a high-level planning down to final interfaces.

Section II is dedicated to API's lifecycle: how interfaces evolve over time, and how to elaborate the product to match users' needs.

Finally, Section III is more about un-engineering sides of the API, like supporting, marketing and working with a community.

First two sections are much of of interest to engineers, when third section being more relevant to both engineers and product managers. But we insist that this section is the most important for the API software developer. Since API is the product for engineers, you cannot simply pronounce non-engineering team responsible for its product planning

and support. Nobody but you understands more what product features your API is capable of.

Let's start.

Chapter 2. The API Definition

Before start talking about the API design, we need to explicitly define what the API is. Encyclopedia tells us that API is an acronym for ‘Application Program Interface’. This definition is fine, but useless. Much like ‘Man’ definition by Plato: Man stood upright on two legs without feathers. This definition is fine again, but it gives us no understanding what's so important about a Man. (Actually, not ‘fine’ either. Diogenes of Sinope once brought a plucked chicken, saying ‘That's Plato's Man’. And Plato had to add ‘with broad nails’ to his definition.)

What API *means* apart from formal definition?

You're possibly reading this book using a Web browser. To make the browser display this page correctly, a bunch of stuff must work correctly: parsing the URL according to the specification; DNS service; TLS handshake protocol; transmitting the data over HTTP protocol; HTML document parsing; CSS document parsing; correct HTML+CSS rendering.

But those are just a tip of an iceberg. To make HTTP protocol work you need the entire network stack (comprising 4-5 or even more different level protocols)

work correctly. HTML document parsing is being performed according to hundreds of different specifications. Document rendering calls the underlying operating system API, or even directly graphical processor API. And so on: down to contemporary CISC processor commands implemented on top of microcommands API.

In other words, hundreds or even thousands of different APIs must work correctly to make possible basic actions like viewing a webpage. Contemporary internet technologies simply couldn't exist without these tons of API working fine.

The API is an obligation. A formal obligation to connect different programmable contexts.

When I'm asked to provide an example of a good API, I usually show the picture of a Roman viaduct:

- it interconnects two areas;
- backwards compatibility broken zero times in two thousand years.

What differs a Roman viaduct from a goof API is that an API presumes a contract being *programmable*. To connect two areas some *coding* is needed.

The goal og this book is two help you to design an API which serves its purposes as solidly as a Roman viaduct does.

A viaduct also illustrates another problem of the API design: your customers are engineers themselves. You are not supplying water to end-users: suppliers are connectin to you engineering structure by building their pipe system upon it. From one side, you may provide water access to much more people, providing you're not spending time on plugging each individual house to your network. But from other side, you can't control the quality of suppliers' solutions, and you are to blamed on every water problem caused by their incompetence.

That's why designing the API implies a larger area of responsibilities. **API is an amplifier to both your opportunities and mistakes**.

The API Design