

Creating an SDK: Writing on the Edge

By JOHN T. SARR, *Senior Member, Willamette Valley Chapter*

Many of us who are technical writing professionals pride ourselves on our ability to communicate technical concepts in everyday terms or to provide instructions on how to use a new technology. Our audiences, for the most part, comprise end users who range from novices to power users. We enjoy learning a new technology, describing what it does and looks like—the familiar GUI (graphical user interface) part that we can play with—and organizing basic instructional and descriptive material for the product.

Creating documentation for a software development kit (SDK) has a similar intent and approach. An SDK provides developers with information and coding examples to enable them to develop applications that will work with a newly developed technology.

The problem, of course, that many writers face is that there is no “there” there in an SDK. The product is basically just code. There is no need to write instructions on how to use a dialog box or include a helpful screen shot (which, for some of us, is our usual fare). We can’t see all the different ways the code will be used—in this sense, there’s no product to touch and feel. Yet we must grasp the functions of the code. SDKs also present a challenge organizationally—we can’t avoid listing lines of code and method descriptions, yet we must organize and systematize this material for easy access.

If the idea of having to document an *evanescent* product isn’t enough to scare us off, there are still other reasons many shy away from SDK writing:

- The audience for an SDK is usually small and restricted to developers, many of whom boast that they never read any product documentation.
- There is little or no GUI involved, so there are no nifty screen shots with callouts for the right-brained crowd, nor any creative visuals with which to break up the blocks of text.

- There is a need to document code, which is seldom commented and little understood by writers. Developers often think the writer can look at the code and decipher what it is for and what to say about it. This is rarely the case, much to the chagrin of developers. As a result, writers can find themselves constantly at the mercy of developers in gathering needed code descriptions, which can often be a tedious process.

With all that said, documenting SDKs can still be interesting and rewarding. The purpose of an SDK, after all, is similar to that of end-user documentation: Explain what the product is, how it works, and how to use it. On that level there should be no problem. SDK writers succeed by twisting their brains around the workings of the product and the intricate code descriptions—which can come from long one-on-one whiteboard presentations, background documents, external architecture specifications, and project requirements documentation—and then organizing this information.

Guidelines

When I took up the task of creating an SDK, I was surprised to find that there were no real guidelines available for writers. Perhaps there is no one set of rules for all SDKs, I concluded, after I had looked at those accompanying established and not-so-established products. Despite this frustration, there did seem to be some strands of similarity among them. This finding, along with my own experience in working on an SDK, led me to create the following guidelines:

1. Establish the purpose and needs of the documentation and yourself.

When starting out, the development team sometimes asks writers what information *they* as writers need to write the SDK. The questions the writer must ask at the beginning are “What will the SDK

allow a developer to do?” and “What does a developer who knows nothing about the current technology need to know to write code?” Developers are sometimes surprised that this information needs to go in the SDK documentation; still, they are the best people to provide it. Sometimes a lead developer, or a group of developers, can verbalize this information for you.

Once these first details of the documentation are established, make clear your purpose: You are there to collect data, write, edit, and format. You are *not* there to read code and write about its interpretation (unless of course you are skilled in the code and enjoy interpreting it for other codeheads). Let your motto be “I don’t do windows [that’s probably Windows® for some!], and I don’t do code.”

2. Create an outline of content.

With the purpose and need of the SDK and yourself made clear, you should list the items that will be included in the documentation. Once again, the developers can help compose, order, and fine-tune this list. It basically boils down to an introduction, a narrative on how to implement the code, some sample code, and application program interface (API) references. An API is a basic library of program coding classes and related methods for building a software application.

3. Write an introduction.

As with any documentation set, you need an introduction, one that describes the SDK and its contents and what a developer can do with it.

4. Create narratives with sample code and examples.

The introduction is followed by a brief overview of what the code does or how the code fits together. This narrative section can be quite technical and very daunting to non-code-savvy writers. Sometimes you can get one of the developers

to sketch out, if not write, most of the details. If code references are used throughout and you are not a code person, then taking dictation from a developer or having the developer write out most of the text may be the only solution. These documents can be pure text, or text interspersed with code showing sample usage. Sometimes code samples are provided as separate documents elsewhere in the SDK.

5. List API references and link them to narrative.

Finally, there are the API references, the backbone of the SDK, which list classes, methods, and their descriptions. Figure out which code files are distributed with the SDK and need to be referenced in the documentation. The class name is basically the header code file name. Sometimes a developer can go through a few header files with you and point out what needs to be included in the descriptions. (A header file is the file called at the beginning of a program that contains definitions of data types and variables to be used.) This is certainly doable if the code is commented. Sometimes you can provide a template (see Figure 1) for developers to fill in.

6. Add supporting documents.

Not required but nice to have are what some call “comfort” documents, which add value to the documentation set. They include detailed background information, white papers, and notes that list hints, tips, and tricks in using the code.

7. Don’t be afraid not to be in control.

The usual procedure for writers is to gain a full understanding of a product’s technology, its user interface, and how best to accomplish tasks using the product. Writers are used to being in full command of the subject before writing about it. At times, however, in writing an SDK, certain technology concepts and underlying code are beyond a writer’s ability to

Trusting in “the Force” of the development team is usually the most difficult task in the project, one that makes or breaks SDK writers.

Interface Description

This section provides a high-level description of the interface.

When to Use

This section describes the clients of the interface as well as how the interface is used.

Methods

This section is a summary of the methods in the interface.

<i>Interface Methods</i>	<i>Description</i>
Method1	High-level description of method 1
Method2	High-level description of method 2

Interface::Method1(X param1, y* param2)

Low-level description of method 1.

Parameters

<i>param1</i>	description of param1
<i>param2</i>	description of param2

Return Values: List all possible return values and their meaning.

Exceptions Thrown: List all exceptions that can be thrown by the method.

Comments: This section provides any additional information needed to explain the behavior of the method.

Interface::Method2(X param1, y* param2)

Low-level description of method 2.

Parameters

<i>param1</i>	description of param1
<i>param2</i>	description of param2

Return Values: List all possible return values and their meaning.

Exceptions Thrown: List all exceptions that can be thrown by the method.

Comments: This section provides any additional information needed to explain the behavior of the method.

words together and having the developers indicate whether they are correct, even though you don't quite understand it all, is OK. In fact, this trial-and-error procedure, where the writer takes a stab at describing something and the developer says yea or nay, is almost standard operating procedure, and often the only way to get information from developers. Often it seems that developers cannot focus on a blank form and fill it out, but find it easy to correct a completed page of details.

8. Don't fret if you find you are doing very little creative writing.

The last thing to keep in mind is that SDK writing seldom provides an outlet for true creative writing. It is a fairly cut-and-dried listing of code with some explanations, a process that on the whole is mechanical and technical. Perhaps because of this, SDK writers are much in demand.

SDK writing does offer other means to be creative, however. Devising ways to obtain information from developers and placing it in a form understandable and easily accessible by other developers often requires a very creative process. And when it comes down to it, isn't this why they call us technical communicators? **i**

John Sarr is a senior technical writer for Ace Communications. He worked at Intel Corporation on a 3-D graphics development team in the Intel Architecture Lab. He can be reached at jsarr@teleport.com.

Figure 1. Sample API template

describe. In such cases, the writer must rely on the subject matter experts to provide many, if not most, of the words. Often, SDK writers must put together words that may not hold a lot of meaning for them, but that offer the best way to

describe something—and make perfect sense—to another developer.

Letting go and trusting in “the Force” of the development team is usually the most difficult task in the project, one that makes or breaks SDK writers. Putting