

[\[Back to Professional Contributions\]](#)

"Yesterday API was just another acronym; today I have to document one!"

by Susan W. Gallagher

Object-oriented code is modular and portable. As a result, technology sharing is becoming common and more and more companies are packaging up their code and sending it on the road. Suddenly, you're faced with providing programmer docs and you don't speak a word of C++ or Java. This session presents an overview of API (Application Program Interface) documentation and discusses the kind of information you need to provide, where to find some of the information you need, and how to understand it once you find it.

Why me?

If you "signed up" for a technical writing job that involved documenting an end-user application and you're suddenly faced with needing to document an API (Application Programming Interface) or SDK (Software Development Kit), you're probably feeling overwhelmed. What is this thing that needs documentation? Where did it come from? Where and how do you even get started?

Why are more tech writers being asked to document APIs and SDKs?

One of the basic characteristics of object-oriented code is encapsulation. This design principle stipulates that the inner workings of an object-oriented program are hidden from the public eye. OO code need only present an interface to the world -- a list of the behaviors it supports and the data it contains. How the code processes the messages it receives is private.

Additionally, object-oriented code is modular. Code is developed as a collection of classes that are arranged hierarchically. Consequently, separate and distinct functions fall in separate and distinct areas of the program. Modularity facilitates a building-block approach to software development

Encapsulation and modularity are the forces that drive the development of APIs and SDKs. Distinct functions within a program are already within separate modules, so there is little extra work to be done to "package" a function. And the code is hidden, so a company can package its code as an API or SDK without fear of piracy.

And so, the age of modular software that the experts predicted way back in the 80s is upon us; but it's not the end-user that's selecting the modules to be included in the system, it's the developers and the product managers. For example, a current-day teamwork productivity application that offers version control, bug tracking, threaded conversations, and document comparison may comprise modular applications from myriad companies: the conversation module from Company W, the charting function from Company X, the report writer from Company Y, and the comparison utility from Company Z.

Clearly, object-oriented code has increased the number of make-or-buy options for software organizations; and just as clearly, the decision more and more often is to buy and integrate rather than to develop (make). The software companies on the supply side of these technology-sharing arrangements find new markets for their code without needing to develop new products or new marketing campaigns; while the software companies on the consumer side of these arrangements produce feature-rich commercial applications without significantly increasing development costs or time to market. It's a win-win situation.

What are APIs and SDKs?

An Application Program Interface or Software Development Kit is a module of programming code that performs one or more specific functions. The essential difference between the two is that the API comes ready to use; the SDK needs to be compiled or built on the target development platform. Frequently, software for use on a single hardware platform or development environment is bundled as an API while software for use on multiple hardware platforms or development environments is bundled as an SDK.

Physically, the product consists of a group of files. Some of these files may be provided in text form, such as C++ header files; but most of the product consists of binary code or dynamic link libraries (DLLs). Logically, the product consists of specific behavior that can be invoked programmatically.

What's the purpose?

API documentation is the gateway into the provided code. Its job is to document and explain the ways in which it is possible to make the code do what it's supposed to do. For example, the API documentation for a charting package would need to explain how to pass the data to be charted into the module and how to request that the appropriate chart type be

displayed with that specific data plotted.

You may have the advantage of already knowing all of the things the API is meant to do, because you have already documented it as an end-user application as part of your company's primary product. But you will now be looking at it from a different point of view. You'll now be looking at the software without its graphical interface--the program without any clothes on.

The documents you write will provide the purchasing development team with a window into your company's code. They will rely on those docs to integrate the API functionality into their application.

Who's the user?

This is the one time when you can look to your development organization for an end-user model; in this case, your user is very much like you. The developers in your target audience:

- Are familiar with the programming language and its jargon
- Are working under deadline
- Are concerned that the look and feel of your application mesh with theirs as much as possible
- Are *not* familiar with your application's functionality

Trust your development staff to use vocabulary that is appropriate to the documents at hand. Words like instantiate, scope, and mutex may sound foreign to you, but your developer audience will understand them.

Additionally, you may be called upon to provide end-user documentation to the purchasing company's publications department. This segment of the target audience is very much like you and is concerned with providing comprehensive end-user documentation for your product that can be incorporated seamlessly into their own. They have standards and conventions that they will want to enforce, just like you have. The technical writers in your target audience:

- Are familiar with software documentation and its jargon
- Are working under deadline

- Are concerned that the look and feel of your documentation mesh with theirs as much as possible
- Understand their end-users better than you do
- Are *not* familiar with your application's functionality

What are my deliverables?

The list of deliverables for an API or SDK package may include:

- End-user documentation for the publications department
- Installation, configuration, and build instructions
- A user guide
- A reference manual
- Examples, examples, and more examples

End-user documentation for the publications department

The development organization that purchases your company's API will likely be integrating it with other APIs and their own programming code to create a full-featured application. Their publications team will need to produce end-user docs that view the application as a whole, sometimes integrating the documentation from two or three different companies into a single document.

As you prepare documents to be included with the API, picture yourself on the receiving end. If it were you taking myriad documents and assembling them into one, you'd be concerned with altering the look and feel to assure consistency. Deliver the documents in an easily modifiable format such as Word or FrameMaker files. Provide uncompiled help files or a map file that lists all the values for windows and dialog boxes defined within the API.

Do not deliver compiled help files or generated HTML documents. While the customer's publications department may be able to integrate your compiled help file with their own, they won't be able to change "select" to "choose", reorganize the topics to mirror the

organization scheme in the rest of the document set, or change the background color of the help window to match their standard. Generally speaking, HTML files that someone else generated, complete with numeric links and targets, are a nightmare to maintain and a monumental task to reverse-engineer. Do unto other pubs groups as you would have them do unto you!

Installation, configuration, and build instructions

Although you'll need to rely on your developers or testers to supply this information, you can take control of the project by defining what the set-up instructions should include and creating an outline structure for the book.

Set up instructions should include the procedures necessary to install the API on the target system, whether any environment variables need to be set, and whether an addition needs to be made to the system's path variable. For SDKs, you'll need to include instructions for building or compiling the software, too. A list of installed directories and files allows the user to verify that the product was completely installed and become familiar with file locations.

A user guide

Although you won't be able to write the user guide yourself, you can take control of the task by working with the developers in your organization to create an outline for the book before they start writing it. Your advantages are twofold: you already know what the software is supposed to do because you've documented it as part of your company's end-user application and you know how to structure a book to make it task-based. Your challenge is to explain what goes on behind the scenes; to document how to achieve results with the software programmatically rather than by clicking buttons and making menu selections.

Tasks that a development team will need to perform to integrate your API into their application may include:

- Adding a selection to their application's menu that communicates with your API's code
- Adding a button to their application's toolbar that communicates with your API's code
- Passing data from their application into your API

- Specifying the format of the output from your API (for example, specifying whether a charting package produces a pie chart, bar chart, or scatter graph)
- Manipulating output from your API (for example, specifying window size and position or whether output appears in a separate window or as part of the application's main window)
- Catching and handling error messages

A reference manual

The reference manual provides a comprehensive description of the software's public interface. Because object-oriented code is encapsulated, the way your development team implemented the functionality of the API software is hidden from the integrating development team. It is the reference manual's duty to list the messages that the API understands and to explain the consequences of those messages. Typically, an API reference manual will include the following elements:

- A list of classes included in the API and a brief description of each class
- Inheritance information for each class
- A list of the messages that each class understands and the consequences of calling those messages
- A list of the data that each class holds and how to access that data
- A list of any special constructs, such as enumerations, defined within the API
- Examples, examples, and more examples

Overview of C++ and Java

Headers

A C++ class is generally composed of two files: a header file (with a .h file extension) in which the class is declared, and a source or implementation file (with a .cpp extension) in which the actual programming code resides. When you compile a C++ program, the implementation file is converted to binary but the header file remains as plain text. Programmers can browse the header file to learn about the class interface. The C++ header files are delivered with the API.

In Java, classes are declared and implemented in the .java file. The .java file is compiled into bytecode that is then interpreted when the program is run. Once the .java file is compiled, there is no way to browse the class interface.

Keywords

Both C++ and Java employ keywords to label discrete sections of code. These keywords can indicate function accessibility and other behavioral nuances that are important to know. Some keywords in C++ and Java that you need to be aware of are:

public Indicates that the code that follows is accessible to all areas of the program.

protected Indicates limited accessibility of the code that follows. Access rules are different for C++ and Java.

private Indicates that the code that follows is accessible by objects of the same class.

static Indicates that the function or variable belong to the class but not to any particular object of the class.

virtual Used in C++ to enable polymorphic behavior. When used with = 0, indicates that the class in which it is defined is an abstract base class. C++ calls this a pure virtual function. This function must be implemented in a subclass.

final Used in Java to disable polymorphic behavior.

#include Used with the name of a header file in C++ to access the features declared in the named header file from within the current file.

The accessibility modifier keywords (public, protected, and private) are used differently in C++ and Java. In general, C++ divides the code into sections according to accessibility modifier while Java labels each method.

Case Sensitivity

Although both C++ and Java are case sensitive; Java is more case sensitive than C++. Both languages see the difference between upper- and lower-case in class and variable names,

but C++ ignores differences in case for file names while Java does not.

Conventional Vocabulary

The following vocabulary is used to describe objects and actions in object-oriented programming:

method/ function	A discrete block of code with a name by which it can be called. The more object-oriented term is method, and this is the term Java uses. Function is used only in C++ and is a carry-over from C.
call	The action of invoking a method or function.
message	The content used to call a method.
instantiate	The act of creating an object, an instance of a class.
constructor	The method that instantiates an object.
destructor	The method that destroys an object.
signature	The format of a method or function. The method signature specifies its input parameters and return values.
type	The description of a value. In C++ and Java, all values are of a specific type. Types can be standard, such as integers (int) or characters (char), or implementation-specific, such as Employees or Managers.
scope	A term that indicates the lifetime of a value or action. For example, when a variable is declared within a block of code, that code block is the scope of the variable. When program execution leaves that block of code, the variable data is out of scope and no longer available.

parent class/ child class superclass/ subclass base class/ derived class	Used to describe an inheritance relationship. The child or subclass inherits all of the attributes of the parent or superclass. Objects of a subclasses are said to have an "is a" relationship with objects of the superclass. For example, if Manager is a subclass of Employee, an instance of class Manager is a(n) object of class Employee as well.
callback	Methods that are not explicitly called by an application but are set to trigger automatically when a specific event occurs or condition is met.

Class information

You'll need to include a complete list of classes included in the API and a brief description of each class. It's important to identify abstract base classes (or pseudo-classes). You cannot instantiate an object of an abstract base class; it serves as a template on which to base concrete classes.

In C++ and Java, a class declaration includes the keyword class followed by an identifier that is the name of the class. C++ stores this information in the header file for the class; Java does not make use of header files, so the class declaration is in the .java file.

Example:

```
class myClass
```

Inheritance information

Object-oriented code is organized in a hierarchical class structure. Subclasses inherit functionality from their superclasses, so knowing where a class is positioned within the class hierarchy is important. To show inheritance, you can include simple "inherits from" and "inherited by" statements or create a hierarchy chart. A subclass is said to be derived from its superclass.

C++ shows primary inheritance by following the class declaration with a colon and the name of the parent class.

Example:

```
class myClass : parentClass
```

Java shows primary inheritance using the keyword *extends* and the name of the parent class.

Example:

```
class myClass extends parentClass
```

Inheritance in C++

C++ supports multiple inheritance, so a class can inherit from more than one parent. To call a function that exists in a parent class, programmers use a double colon, called the scope resolution operator, to specify which function they are calling.

Example, multiple inheritance:

```
class myClass : parentClass1, parentClass2
```

Example, calling a function in the superclass:

```
parentClass1::doSomething( )
```

Inheritance in Java

In Java, a class can only inherit functionality from a single class. It can, however, inherit interfaces from additional classes. When a Java class inherits an interface, it essentially inherits the method signature without inheriting any functionality. Before the method can be used, it must be implemented in the new class. Java indicates this type of inheritance using the *implements* keyword.

Example, multiple inheritance:

```
class myClass extends parentClass1 implements parentClass2
```

Where C++ uses the scope resolution operator to call a method in the parent class, Java uses the keyword *super*.

Example, calling a function in the superclass:

```
super.doSomething( )
```

Listing the methods of a class

When you list the methods of a class, be sure to list the method signature exactly as it appears in the code. However, to make the reference document more scannable, you may want to list just the name of the method first. By convention, in both C++ and Java, method names are identified by trailing empty parentheses.

Example:

```
doSomething( )
```

Method signatures in C++ and Java have the same format.

keyword(s)	C++ uses the keywords <code>virtual</code> and <code>static</code> preceding a method signature. Java uses the keywords <code>static</code> and <code>final</code> and the accessibility modifiers (<code>public</code> , <code>protected</code> , and <code>private</code>) preceding a method signature. Keywords are not required in this position and are used to alter the method's behavior in some way.
return type	The first required element of the method signature, the return type, specifies what kind of value the method returns (sends back to its caller) when it finishes processing. Return types can be any standard type (<code>int</code> , <code>char</code> , <code>boolean</code>) or user-defined type (<code>Employee</code> , <code>Manager</code>). The keyword <code>void</code> indicates that the method does not return a value.
method name	The method name appears in the method signature immediately after the return type. This is the name used to invoke the method.
input parameter list	The input parameter list specifies the number and types of input values that the method accepts. This list immediately follows the method name and is enclosed in parentheses. A parameter specification consists of a type and a variable name. Multiple input parameters are separated by commas. Empty parentheses indicate that the method does not accept any input parameters.

Example:

```
void doSomething(int num1, int num2)
```

The above method:

- Does not use a keyword modifier
- Uses the keyword `void` to indicate that the method does not return a value to its caller
- Is named *doSomething()*
- Accepts two input values, the integer *num1* and the integer *num2*

Listing the public data of a class

In an object-oriented application, an object contains data as well as functionality. In some cases, that data is public, so it can be modified by other methods contained in other classes. When a class contains public data, you must describe it by including the type of the data and any limitations on it. For example, in an application that deals with dates, the day of the month may be public data. It would have a data type, such as integer, and a limit on the values to which it could be set (1 to 31).

In commercial applications, it is more customary to keep class data private and provide *set* and *get* accessor methods to alter and retrieve the values.

Listing special constructs such as enumerations

Your application may contain special constructs that need to be documented as well. Enumerations list the order of words so that they may be compared using less than and greater than operators. For example, if an enumeration lists the days of the week, in a comparison operation Monday is less than Tuesday.

Remember to ask if there are any special constructs defined within the API.

Exceptions

Exceptions are error messages that the application generates internally. If object A calls a method on object B and an error occurs, object B is said to throw an exception. To prevent the application from crashing, object A must catch the generated exception and handle it. Be sure to document the exceptions that your API can throw so that the development team can catch them appropriately.

Examples, examples, and more examples

Developers rely on examples extensively. Many programmers admit that they've created a great deal of code by copying a working example and changing the parameters to fit their needs. When you plan the documentation deliverables for an API, make sure that your developers provide extensive examples. Depending on the nature of your API, examples can take the form of code snippets or simplified applications.

Resources

As you plan and create API documentation, these resources will help.

<http://foldoc.doc.ic.ac.uk/foldoc/contents.html>

The free online dictionary of computing is a great place to check terminology.

www.rational.com

Follow the UML links for Universal Modeling Language information and tutorials when you need to represent object-oriented code graphically.

Backus Naur Form (BNF)

I have yet to find a succinct resource for BNF, the predominant notational convention for describing programming syntax. A web search finds many pages that talk about BNF, some of which claim that a succinct reference does not exist.

Susan W. Gallagher began working as a technical writer and trainer in the software industry in 1983. She has designed information strategies for target audiences that span the range from naïve end users to programmers. She is currently Technical Publications Manager at Expersoft Corporation in San Diego, California. She can be reached at work sgallagher@expersoft.com or after hours

<http://pw1.netcom.com/~gscale/susanwg/> or susanwg@ix.netcom.com.

© Copyright 1999, Susan W. Gallagher.

All rights reserved.