madcap software

WHITE PAPER

The Definitive Guide to Creating API Documentation

CONTENTS

Introduction

APIs stands for application programming interfaces. They are the building blocks, the code, used by developers to code applications. For example, you won't buy any APIs off-the-shelf at Staples, but it is likely that many of the products you can buy off the shelf were developed (coded) using APIs.

Most of the principles that apply to technical writing for other products, such as GUI-based applications and end-user applications, apply to documenting APIs as well. But there are some major differences that often elude technical writers moving into this area of technical writing. This white paper covers ten best practices for writers to keep in mind:

1. Follow the standard template or outline for organizing reference pages.

2. Use a terse, factual writing style. Sentence fragments are desirable. Avoid adjectives and adverbs.

3. Provide complete information about each API component.

4. List all error messages alphabetically, specify the level of the message, and provide suggested workarounds and solutions.

5. Provide working code snippets for each method, function, and resource. You don't need complete examples, but show a common use of that element.

6. Provide flow charts showing the sequence of the most commonly used methods for common use cases.

7. Provide sample programs demonstrating common use cases.

8. Provide a "Getting Started" guide showing how to develop a program for a common use cases.

9. Provide performance and tuning information.

10. Provide a contact in case developers have questions or need additional assistance.

Best Practice 1: Follow a Standard Template or Outline for Reference Pages

There is a commonly accepted baseline for API documentation. That baseline is a complete, comprehensive, and accurate set of reference pages that documents every interface, method, function, or resource.

Most APIs use a very similar template or organizational structure for API components, including methods, functions, or resources. These categories of API elements each perform a single operation or task, such as opening a file, starting an interactive session, and logging into a computer system. C-type languages use the terms "method" or "function" interchangeably. The Java programming language uses the term "method." Web service APIs use the term "resource." But all three names are functionally the same and are documented in much the same way. You should follow the standard template or outline for organizing reference pages.

Note: In this white paper, the term "method" is used to generically refer to all three terms.

An example of an API reference page template for a method is shown below.

Template or an API Reference Page

Method_Name

A brief description of what the method does.

Syntax

Display the actual method call used in the code.

Description

More detailed information about the method; information developers need to know to call the method.

Note: For all of the following sections, if the section doesn't apply to the specific method or has no elements, such as no return value or has no related methods, explicitly state that by using a term such as "None" or "Not applicable." This tells developers that you didn't forget to put information in those sections.

Parameters

List all of the parameters here, in the order they appear in the syntax. You can either use a pseudo-list format as the following or put the information into a table, shown after the pseudo-list.

Parameter1 - data type

What the parameter does. Optional/required. Examples.

Parameter*n* - data type

What the parameter does. Optional/required. Examples.

Listing parameters in a table format:

Parameter	Optional / Required	Description	Data Type
parameter1			
parameter2			
parameter*n*			

Return Value

Value returned by the method:

returnValue - data type

Describe the return value.

Examples

A short code snippet here, with explanation, showing use of this method.

Errors

List the errors that can be returned for this method, with the level of error and the recommended workaround or solution.

Notes

Any supplemental, “nice to know” information.

See Also

Cross-references to related methods.

Sample API Reference Page

For a sample API reference page for the **SessionLogin** method, which shows the organization and content discussed in this section, see “**Appendix: Sample API Reference Page**” on page 19.

Best Practices 2:
Use a Terse, Factual Writing Style

Most of the well-established practices for technical writing also apply to documenting APIs:

- Write clearly in plain English.
- Be factual.
- Be consistent with your terminology.

But there are some significant differences in documenting APIs:

- Write very tersely. Developers do not like flowery prose.
- Sentence fragments are acceptable. For example, when you are describing what a specific method does, it is acceptable and desirable to use a sentence fragment. For example, for a sessionTerm method, your brief description would read, "Ends a session and closes all open files."
- Avoid adjectives and adverbs. Developers are very factual. They don't want to read how fast something works, how easy it is to use, etc. Developers will quickly lose confidence and trust in your documentation if you use adjectives and adverbs.
- Be alert for homegrown terms that are used by a specific developer or team of developers that are not industry-standard. If you encounter a jargon-like term that you don't recognize, ask your developer or team if that is a standard term in your industry. If it isn't, push to determine the correct standard term. If it is a new term but your team insists it should be used, define it the first time you use it in your documentation. (Optionally, consider creating a glossary where you define all of the new terms used in your API.)
- Often method names are formed by concatenating many words, which leads to very long terms. Accept these as is. Do NOT make up your own terms, because you think the names are too long or break awkwardly in printed or PDF output.

Best Practices 3: Provide Complete Information About Each API Component

Let's look at each of these sections of the API template, in the order they should appear on the reference pages.

Method_Name

The Method_Name should be your heading at the top of a reference page. Most API documentation does not include "Method" after the Method_Name. Each method is documented starting on a new page. This makes it easy for developers to view or print the complete reference information for a single method. It is confusing if you start a new section for a new method partway through a page for printed documentation. For online help, having each reference page as a separate topic is much easier for developers to read and find the information they need.

Following the Method_Name, provide a brief description of what the method does. For example:

Logs into a previously created session, where a session is an interactive sequence of user requests and system responses.

Syntax

In this section, you document the complete syntax for the method, exactly as it is used in the code. For example:

```
STATUS SessionLogin(
                SESSION_HANDLE session,
                const STRING username,
                const STRING password,
                UINT32 timeout);
```

The "STATUS" that starts the syntax is the status code returned by this method. Many methods return a status code or value that can be used to determine what the program does next or some alphanumeric value that will be used to calculate some math result or data to be stored or otherwise acted on.

Next is the method name, SessionLogin, followed by an open parenthesis. Then, all the parameters preceded by their data types, separated by commas. The following table shows the relationship between the parameter and its data type. You do not show the syntax this way in your documentation.

Data Type	Parameter
const STRING	username
const STRING	password
UINT32	timeout

The syntax is closed with a closing parenthesis followed by a semicolon. This is standard coding in many programming languages.

Description

In this section, provide more detailed information about the method. This should be essential information. As a guideline, if 90% of the developers need to know this information or will find it useful, include the information in this section. For example:

An application can maintain multiple sessions. You can have up to 8 sessions per application. You need to use the same API User name for each (you will get an ERR_USERNAME_NOT_VALID error if you do not).

Parameters

In this section, list all the parameters, in the order they appear in the syntax. For example:

session - SESSION_HANDLE

A handle to the session.

username - STRING

The API User name defined in the TPM. The TPM must have Topic entitlement rights (publish or subscribe) associated with this API User name before you can publish or subscribe to messages on those Topics.

password - STRING

The password for the API User in string format. The secure value will be put into the securityContext structure.

timeout - UINT32

The length of time that a login will be attempted before failure. This value is in milliseconds. A timeout of 0 means the login attempt will never timeout, and will continue to retry to connect should the initial connect attempt fail.

Return Value

In this section, if the method returns a value, you document that value in the code and what the data type is. Depending on the number of return values, you might need to document all of the possible return values. If the method does not return a value, enter "None." For example:

STATUS - If the function succeeds, then OK is returned; otherwise, one of the status codes listed in the ERRORS section is returned.

Examples

In this section, you provide a small code snippet that shows a typical use or call of the specific method being documented. Developers often copy these code snippets into the code they are writing and customize as needed for their application. For example:

```
returnCode = SessionLogin(sess, DFT_USERNAME, DFT_PASSWORD, 5000);
```

Errors

In this section, you list all possible error messages that can be returned by a method, in alphabetical order, with the recommended solution or workaround. For example:

ERR_NETWORK_WRITE – A generic network error. Check the client log for more information.
ERR_NULL_PASSWORD – You didn't supply a password. Enter a password as described above and re-run your application.
ERR_NULL_USERNAME – You didn't supply a username. Enter a userName as described above and re-run your application.
ERR_USER_NOT_LOGGED_IN – You attempted to use a session without logging in first.

Notes

In this section, you include any supplemental, "nice to know" information. This is information that probably only will be used by 15% of the developers or used 15% of the time. For example:

Before calling SessionLogin, you will need your API User name and password.

When your application is finished processing all messages, call SessionTerm to end the established session.

See Also

In this section, you provide hyperlinked cross-references to all of the related methods for the method being documented. For example:

SessionLogin
SessionTerm

Best Practices 4: Document All Error Messages

You should list all possible error messages that can be returned by a method, in alphabetical order. Putting the error messages in alphabetical order makes it easy for developers to find the specific error message of interest, especially if a method can return many error messages, which is not uncommon in this area. This also makes it easier for you to maintain the list and ensure you don't have duplicates, which leads to confusion. Developers might be able to find some of the error messages returned for a method in the source code, but that is all they can mine from the source code. Unless an error message is intuitively obvious, such as ERR_NULL_PASSWORD, having the error message doesn't help them. You should describe what the error message means and the level (informational, error, fatal, severe, warning, etc.). It is very helpful if you describe what caused the error and the recommended solution or workaround for the error. For example:

> ERR_NETWORK_WRITE – A generic network error. Check the client log for more information.
>
> ERR_NULL_PASSWORD – You didn't supply a password. Enter a password as described above and re-run your application.

Tip: Often an error message can be returned for multiple methods. If your authoring tool provides snippets capability, consider making an error message that is returned for many methods into a reusable snippet that you call as needed. This will make maintenance of the description for an error message much easier.

Best Practices 5: Provide Working Code Snippets for Each Method

Developers love examples. They will copy these from your documentation, paste them into their code, modify as needed, and continue working. So, it is important to provide working code snippets. You do not need to have a complete working example, including all header information, included files, etc., but the example should work when pasted into an existing program. These examples should show typical or common use cases for the method. You will likely need to work with your developers to determine typical or common use cases. It is your responsibility to test code snippets given to you by developers to ensure they will work when used in a working program. If you cannot get an example to work, it is appropriate to ask the developer who provided it why it doesn't work. And work with him/her to correct the example or provide another example that does work. For example:

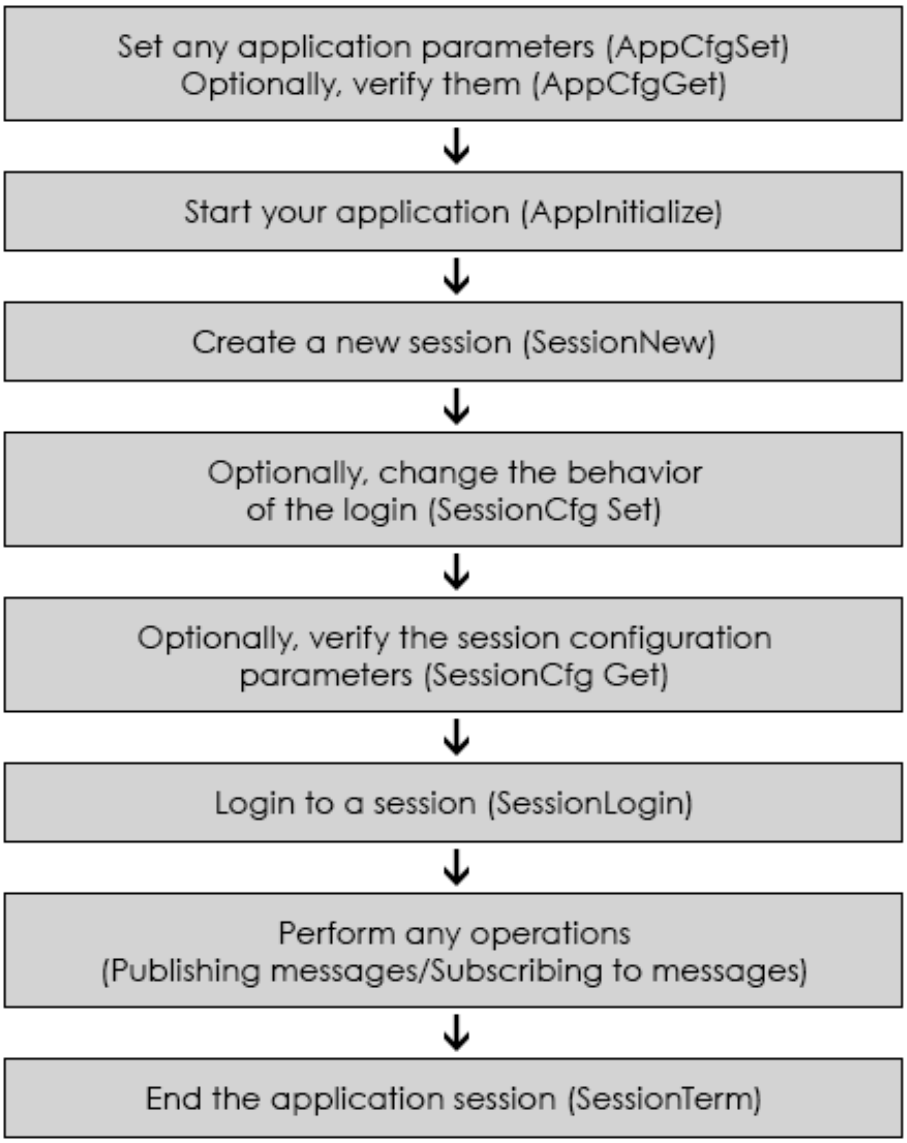
```
returnCode = SessionLogin(sess, DFT_USERNAME, DFT_PASSWORD, 5000);
```

Best Practices 6: Provide Flowcharts Showing Common Use Cases

You should provide flowcharts showing the sequence of the most commonly used methods for common use cases.

APIs often have many methods in them, having several hundred methods is common. However, the rule that 85% of your users will only use 15% of your product applies to APIs as well. The challenge for API users is to determine which methods need to be used all or most of the time vs. which are only used occasionally or rarely. One way of showing the most commonly used methods is to provide flowcharts for the most common use cases, showing the methods used in those use cases and the order they should be called.

For example, the following flowchart from an API documentation set shows how to log into a system:



This flowchart is very helpful to developers, because it shows the required methods to call to log into a system and what order to call them, as well as some optional methods that might be helpful. Note the brief but concise descriptions of what each method does or what operation is performed at each step.

Best Practices 7: Provide Sample Programs Demonstrating Common Use Cases

Developers love to see sample programs included with APIs. Developers use them to experiment and see how your API works without having to learn your API in detail. There are several considerations you need to address with your sample programs to make them useful to your customers:

- Sample programs should show common or typical use cases for your API.
- They should be complete and run with no errors or warnings.
- You should provide readme files explaining what the program does, how to run it, any necessary input data, and expected results.

Best Practices 8: Provide a "Getting Started" Guide

It is very helpful to identify and document how to program a common use case as a "Getting Started Guide" for new users of your API. This solves the problem of information overload with large APIs. Also, developers prefer to copy and paste existing code and modify it for their use, reusing as much as possible. The challenge for the developers and writers creating an API is to know how users might use the API, what business problems your users are trying to solve, and what some typical use cases are for your users. It is sometimes helpful to talk to staff in marketing or product management, who often have more contact with users than in-house development teams.

The following example shows the outline for a "Getting Started Guide" that shows developers how to speech-enable an application:

Quick Start: Creating a Simple Application

1. Overview - Explains what the Quick Start covers.

2. To speech-enable a simple application - Provides detailed, specific steps to write the code needed to speech-enable an application.

3. To add voice commands to the simple application - Provides detailed, specific steps to write the code needed to add voice commands to the application.

Best Practices 9: Provide Performance and Tuning Information

Many APIs are used to process large amounts of data at high speeds. Developers often want to "tune up" the performance of your API beyond your out-of-the-box performance. Even if you provide parameters and switches to customize various performance metrics, it is often hard and timeconsuming for developers to try the various combinations of parameters, switches, and ranges of valid values. If your product is tunable, at a minimum, you should provide the following information for all tuning parameters and switches:

- Parameter or switch name, an explanation of the parameter or switch, the range of valid values, and default values.
- What effect the default value has and why that value was chosen.
- Guidance on setting the default value to other values in the valid value range according to your specific needs. For example, if you need more data throughput, change parameter x to a value such as the following: n. Conversely, if you need less or more latency, change the value of parameter x to the following: m. Developers can and should experiment around the recommended setting for a specific scenario once they set the value to your recommended setting. But your recommendation, based on your own testing in-house, saves your users from spending a lot of unnecessary time guessing what values to try to achieve a desired result.

For example, you might have a PUB_BW_LIMIT (publisher maximum bandwidth), tuning parameter with the following description:

PUB_BW_LIMIT - The maximum bandwidth, in Mbps, of the publisher. The default value is 100 (100 Mbps). The minimum value is 0, which disables the limit. Use this parameter if you noticed (or your sys-admin told you) that your publisher was flooding the network. You might use it to limit a particularly fast publisher to half the link (500) so that it doesn't overwhelm the subscriber.

You should not use increments smaller than 50.

Best Practices 10:
Provide a Contact Person for you API

Though it is common to have a Technical Support department, whether you work in a large or small company and provide a means of submitting questions about using your company's products by email, telephone, or some web-based interface, APIs present special challenges to this practice:

- Your users are probably software developers themselves, and very computer-literate. Their questions might overwhelm tech support people, who might be inexperienced or don't have deep programming backgrounds.

- Often, developers need to submit log files, screenshots of the run of an application, or other information that is cumbersome at best to communicate by telephone.

Set up a special email account for your API that is directed either to a high-level technical support person who is fluent with your API(s) or a developer on your team who is designated to respond to customer queries. Instead of the common pattern of "**support@yourCompany.com**", create an account, such as "**nameOfYourAPI_support@yourCompany.com**" and have all queries go to the designated technical support person or designated developer.

If you are documenting internal APIs for internal use only, consider specifying the actual developer who is assigned to support internal users of that API. In this case, your email link would go directly to that developer, Jane_Developer@yourCompany.com.

Publish a short topic named something like "Technical Support" that you include in your documentation after your content topics/chapters. Suggested content:

For more information or help on using this API, contact **Jane_Developer@yourCompany.com**.

Providing an email link to a knowledgeable technical support or designated API developer benefits both your company and customers. Your company resolves customer issues faster, thus saving time and money. Your customers get their questions resolved faster and more accurately. Both your company and customer get increased customer satisfaction.

Summary

To review the ten best practices for API documentation:

1. Use a standard template or outline to organize reference pages.

2. Use a terse, factual writing style. Sentence fragments are desirable. Avoid adjectives and adverbs.

3. Document every API component completely.

4. List all error messages alphabetically, specify the level of the message, and provide suggested workarounds and solutions.

5. Provide a working code snippet that shows a common use for each method.

6. Show the sequence of the most commonly used methods for common use cases.

7. Provide full, working sample programs that demonstrate common use cases for your API.

8. Develop some common use cases and provide a "Getting Started" guide that documents how to program for them.

9. Provide performance and tuning information with recommended adjustments to default values, to improve performance for unusual scenarios.

10. Designate a contact person, who is fluent with your API, as a resource for your users, not just a generic "**tech_support@xyz.com**" mailing address.

By implementing these best practices, you will provide documentation that not only meets the needs and expectations of your users but will substantially help them get started quickly in learning and using your API to solve their technical challenges.

Appendix: Sample API Reference Page

SessionLogin

Logs into a previously created session, where a session is an interactive sequence of user requests and system responses.

Syntax

```
STATUS SessionLogin(
                SESSION_HANDLE session,
                const STRING username,
                const STRING password,
                UINT32 timeout);
```

Description

An application can maintain multiple sessions. You can have up to 8 sessions per application. You need to use the same API User name for each (you will get an ERR_USERNAME_NOT_VALID error if you do not).

Parameters

session - SESSION_HANDLE

Input. A handle to the session.

username - STRING

Input. The API User name defined in the host system.

password - STRING

Input. The password for the API User in string format.

timeout - UINT32

Input. The length of time that a login will be attempted before failure. This value is in milliseconds. A timeout of 0 means the login attempt will never timeout, and will continue to retry to connect should the initial connect attempt fail.

Return Value

STATUS - If the function succeeds, then OK is returned, otherwise one of the status codes listed in the ERRORS section is returned.

Errors

ERR_BAD_NETWORK_CONFIG – Error setting up the SSL channel.

ERR_NULL_PASSWORD – You didn't supply a password. Enter a password as described above and re-run your application.

ERR_NULL_USERNAME – You didn't supply a username. Enter a userName as described above and re-run your application.

ERR_USER_NOT_LOGGED_IN – You attempted to use a session without logging in first.

Notes

Before calling SessionLogin, you will need your API User name and password.

When your application is finished processing all messages, call SessionTerm to end the established session.

Example

```
returnCode = SessionLogin(sess, DFT_USERNAME, DFT_PASSWORD,5000);
```

See Also

SessionLogin
SessionTerm

About the Author

Ed Marshall is an independent consultant technical writer and the sole proprietor of Marshall Documentation Consulting, with over 28 years of experience. He specializes in technical documentation for developers, including APIs (application programming interfaces), SDKs (software developer's kits), Web Services products, etc. Over his career, he has developed expertise in using tools to "let the computer do the work," such as advanced tools for editing files, comparing files, and searching and replacing text. He has given many presentations at local STC chapters and Summits, the WritersUA conferences, Information Development World, the TEKOM trade show in Germany (October 2012) at the organization's invitation, and many other events. He is a Certified MadCap Advanced Developer (MAD) for MadCap Flare. His web site is **www.MarshallDocumentationServices.com**. He can be reached at **ed.marshall@verizon.net**, on LinkedIn, and on Twitter **@EdMarshall**.

Additional Resources

Here are some valuable resources to continue your exploration into single-sourcing and localization.

Flare Webinars:

- "Using MadCap Flare to Support Your International Content Strategy"
  http://www.madcapsoftware.com/webinars/using-madcap-flare-to-support-your-internationalstrategy/

- "A Case Study in Translation Management – How to Reduce Costs by 90% While Enabling New Markets"
  http://www.madcapsoftware.com/webinars/case-study-in-translation-management-how-toreduce-cost-by-while-enabling-new-markets/

- "Case Study: How Hewlett Packard Enterprise Leverages MadCap Lingo to Reduce Translation Costs by 50%"
  https://www.madcapsoftware.com/webinars/case-study-hewlett-packard-enterpriseleverages-madcap-lingo-reduce-translation-costs/

- Jennifer Schudel, Advanced Language Translations: Presentation, "Five Things to Consider When Developing Multilingual Content"
  http://www.madcapsoftware.com/webinars/madcap-flare-and-translation-five-things-toconsider-when-developing-multilingual-content/

e-Books and Sites

- Venga Global – eBook: "Single-Sourcing: Translate Once, Reuse Many Times"
  http://blog.vengaglobal.com/single-sourcing-translate-once-reuse-many-times

- Val Swisher, Content Rules
  http://contentrules.com

Looking for More Resources?

Browse our library of webinars, videos, white papers and more.

Learn more at **madcapsoftware.com/resources**