Enha

***Assessing the need for a set of guidelines /best practices for nlm api documentation***

***RoseMary Hedberg, MLIS Associate Fellow, 2012-2013 Project Leaders: Tammy Magid and Patrick McLaughlin***

**Final Report**

***Spring 2013***

TABLE OF CONTENTS

Acknowledgements 3

Abstract 4

Introduction 5

Methodology 7

Results 9

Discussion 16

Recommendations 17

References 18

Appendix A: A Short Primer on REST vs SOAP 19

Appendix B: The Guidelines 20

ACKNOWLEDGEMENTS

I would like to thank my project sponsors Tammy Magid and Patrick McLaughlin for their immeasurable support, encouragement, and advice.

Thank you to Kathel Dunn, Coordinator of the Associate Fellowship Program, for making allowances and letting me change my mind so many times.

I would especially like to thank Olivier Bodenreider, Stephanie Dennis, Steve Emrick, David Hale, Lee Peters, and Rex Robison for their individual contributions to the project and for volunteering their time and expertise.

ABSTRACT

OBJECTIVE:

NLM has created over 20 APIs that provide access to many of its different datasets. While these APIs have been collected and catalogued for users, documentation of these APIs has not been standardized. The purpose of this project is to assess the need for the development of a set of guidelines or best practices for documenting NLM APIs.

METHODS:

I conducted a SWOT analysis of documentation to highlight its importance and determine how an assessment of documentation practices could specifically benefit NLM. I then examined the documentation practices of each NLM API and ran comparisons between them to find common elements. I also conducted interviews with NLM developers to ascertain their perspectives on good documentation and guidelines. Concurrently, I conducted an environmental scan of the documentation practices of external organizations and reviewed developer blogs, in order to gain a comprehensive perspective of the current trends in documentation.

RESULTS:

The SWOT analysis determined that well-documented APIs would increase the likelihood of platform adoption and reduce the cost of support, as well as serve as marketing tools for the datasets. Common documentation elements for NLM APIs included overviews, usage guidelines, and code samples. API developers at NLM generally agreed that some guidelines would be beneficial in terms of consistency, navigation, and ease of use. The environmental scan showed that there are no established industry standards, yet there were many documentation practices common across well-designed APIs that NLM could adopt. Each of these research methods assisted in the development of a set of fifteen elements to be included and serve as a set of guidelines for documentation, based on best practices.

DISCUSSION:

Based on this research, it would be beneficial to NLM API developers to adhere to this set of guidelines for documentation. Additionally, I recommend the formation of an API developer working group within NLM to use and promote these guidelines, as well as discuss future opportunities for API development.

***INTRODUCTION***

APIs are becoming especially important in the Federal government thanks in part to the Digital Government Strategy and an increased initiative in the White House towards creating a more open and accessible government platform. The Digital Government Strategy enlists an “information-centric” approach, in that it focuses on the treatment of content as data that can be retrieved, downloaded, indexed, searched, and manipulated, all while maintaining interoperability and openness to public consumers. One way to make data open is to create Application Programming Interfaces (APIs), which provide access to data within a user’s own application by using specific program calls for consumption and presentation.

APIs are sets of routines and protocols for building useful software applications. They form ways for two computer applications to communicate over a network using a common language. They also allow applications to communicate directly with a data source to retrieve and manipulate data without having to download the data source locally. At the most basic level, APIs allow a product or service to talk to other products or services, allowing for the opening up of data and functionality to other developers and users. Increasingly it is the way in which organizations exchange data and services. APIs differ from web sites in that, rather than simply providing snapshots of government information, they decouple information from presentation by providing functionality to the data in a machine-readable format. In this way, the implementation of the data and API may change, and the interface remains consistent.

APIs provide users access to data without the need for another application to serve that data. Users of APIs can make data calls and consume the data or present it in their own applications. This makes the data more open. Designing for openness and embracing the use of APIs will enable the government to more easily deliver information and services while making data accessible and usable. According to the good practices developed by the participants of a usability testing program hosted by the General Services Administration (including an API development team from NIH), usability is dependent on a number of factors, including quality design and clear, consistent documentation.

While the Federal government has realized the potential usefulness of APIs, the Office of Management and Budget (OMB) has yet to develop and publish a comprehensive policy for open data, content, and APIs which would officially dictate how to open Federal data effectively and accessibly. Nor has the OMB established a set of guidelines or best practices for API documentation.

NLM has over 20 APIs that provide access to many of its different datasets. While these APIs have been collected and catalogued for users, documentation of these APIs, including type, methods, outputs, etc., has not been standardized. The purpose of this project is to address the need for a set of guidelines/best practices for documenting NLM APIs, incorporating usability standards and addressing HHS 508 and plain language requirements.

METHODOLOGY

A SWOT analysis was conducted in order to determine why documentation should be improved and to find out how a possible restructuring of practices could benefit NLM specifically. An analysis of NLM’s API documentation was then performed; so as to determine its current state, as well as see how it compares to what should be included, based on current trends. Developers of some of NLM’s APIs were then interviewed to ascertain their perspectives on the qualities of good documentation and see if they felt there was a need for the development of guidelines. Throughout all this, concurrently executed was an environmental scan of the documentation practices of external organizations and reviewed developer blogs, in order to find what documentation practices have been adopted by successful APIs outside NLM.

SWOT ANALYSIS / STRENGTHS AND WEAKNESSES

The project started focusing on the purpose of investigating documentation. Why should there be an interest in improving NLM’s documentation? How could an examination and possible restructuring of current documentation practices benefit NLM? A SWOT analysis approach was originally considered for each of the APIs individually; however it became clear that the opportunities and threats for each remained largely similar across the APIs. Therefore, a SWOT analysis of API documentation as a whole, derived from group discussion, served as a guidepost for evaluation. There was a great deal of collaborative debate regarding what would be the strengths and opportunities of well-documented APIs, as well as what would be the perceived weaknesses and threats of poorly-documented APIs.

NLM API COMMON DOCUMENTATION ELEMENTS

In considering the need for the development of a set of guidelines or best practices for NLM API documentation, it was deemed appropriate to gain a solid understanding of current documentation practices within NLM. This was done by accessing the home page of NLM’s APIs at <http://www.nlm.nih.gov/api>, which lists each of the 25 APIs and provides a description of the nature of the data in each API, as well as links to both documentation pages and their corresponding web interfaces. The documentation provided for each of NLM’s APIs was examined, noting the elements the developers included. Then a cross-comparison of elements served to find the common elements seen in the documentation of NLM APIs. This was done by scoring each API on the inclusion or exclusion of various documentation elements, such as overview, usage guidelines, and code samples. Scoring in this manner served to highlight the elements most likely to be perceived as necessary, under the assumption that more commonly used elements were indicative of how important they are perceived in overall API documentation by developers.

INTERVIEWS WITH NLM API DEVELOPERS

After all of the internal analysis, it was determined that it would be beneficial to meet with some of the API developers here at NLM to discuss their impressions on the necessity and feasibility of the development of a set of guidelines or best practices for documentation. These NLM API developers included Olivier Bodenreider and Lee Peters for RxNorm, RxTerms, and the NDF-RT APIs, Stephanie Dennis and Rex Robison for the MedlinePlus and MedlinePlus Connect APIs, Steve Emrick for the UMLS Terminology Services API, and David Hale for the Pillbox beta API. They discussed trends in documentation and the concepts of standards. Each of the developers also demonstrated some of the unique features of their individual APIs. This is a sampling of some of the questions asked:

1. Why did you decide to build an API?
2. What was the API development process like?
3. Who is using your API? Do you keep track of usage statistics?
4. Who wrote the documentation for your API?
5. How did you approach writing the documentation? Did you follow a documentation structure you had seen before, consult a style guide, etc.?
6. What components are essential in good documentation?
7. Do you think there is a need for the development of a set of best practices for NLM API documentation?

ENVIRONMENTAL SCAN

It would be beneficial to look at external sources outside NLM after looking at NLM’s own documentation practices and speaking with API developers, in order to gain a more comprehensive perspective on the true importance and nature of good documentation. As such, an environmental scan of over 50 APIs from several organizations was performed to find the elements of documentation used by successful APIs. Many of the external APIs were selected for inclusion based on their high ratings and mentions in blogs and books dedicated to API development. APIs from private sector enterprises such as Google, Amazon, Facebook, and Twitter, as well as APIs from other Federal agencies, including the White House, NASA, and the CDC were examined. Additionally, popular developer blogs like Programmable Web, API Evangelist, github, and HowTo.gov were reviewed.

RESULTS

*SWOT ANALYSIS / STRENGTHS AND WEAKNESSES*

The initial approach of attempting to perform a SWOT analysis for each individual API was not very successful. Strengths and weaknesses varied greatly between the APIs’ documentation pages, but the perceived opportunities and threats for each API remained largely the same across the board. Therefore, the decision was made to instead focus on the strengths, weaknesses, opportunities, and threats of documentation as a whole. These are some of the points found:

Strengths and opportunities of well-documented APIs:

* Increase likelihood of platform adoption because it’s a less frustrating experience for developers
* Reduce cost of support because developers can find answers more easily
* Marketing tool of the APIs and the Library
* NLM serves as a leader and/or at the forefront of API standards/documentation
* Helpful for new internal (and external) API developers
* Show how one person’s data can link to another’s at NLM

Weaknesses and threats of poorly-documented APIs:

* NLM falling behind other organizations that produce APIs
* APIs not used, as a result of not understanding or not being user friendly
* NLM appearing unprofessional or not uniform as an organization
* Shift in policies away from open data/APIs
* Budget/funding reductions or changes

The SWOT analysis did help in gaining some insight into the benefits of well-documented APIs, as well as further reinforcing the need for good documentation, but it provided abstractions rather than specifics. It was determined that it would be necessary to examine the state of NLM documentation, in order to gain a more comprehensive understanding of current documentation practices within NLM.

*COMMON DOCUMENTATION ELEMENTS*

In order to learn more about which elements should be included in a set of guidelines or best practices, the common documentation elements found in NLM’s APIs were examined. The basic thinking behind this practice was that elements seen many times across APIs are popular, and if many API developers are using these elements, reason stands to fact that they are used so frequently because they are considered by developers to be “good” or necessary for inclusion. Following this, a spreadsheet was created, listing each API and noting the inclusion or exclusion of various documentation elements. Figure 1 shows a sampling of APIs and the documentation elements found (or not found).

| API | Web Service Interface | Overview | Usage Guidelines | Code Example | XML Example |
| --- | --- | --- | --- | --- | --- |
| BLAST | both | X | X |  |  |
| ChemSpell | SOAP | X |  | X | X |
| DailyMed | REST | X |  | X | X |
| DIRLINE | REST | X |  | X | X |
| Entrez | SOAP | X | X | X | ? |

Figure 1: NLM API Documentation Elements

# The results of this analysis were not as telling as they had been hoped to be. By examining NLM’s API documentation from a yes-no criteria, common data element view and then attempting to determine whether or not they met the criteria, it invariably limited the amount of information that could be gleaned. This type of thinking also removed the possibility of seeing any qualitative nuance in documentation elements. It also served to clearly show that looking at only NLM’s APIs would not nearly be enough upon which to build a set of comprehensive recommendations.

*INTERVIEWS WITH NLM API DEVELOPERS*

Meeting with NLM API developers served a number of purposes. It allowed the opportunity of gaining more of an understanding of what real developers want out of their documentation and find what elements they thought necessary. It also afforded the ability to discuss with them the concept of introducing a set of guidelines or best practices for documentation – if it would be feasible or even necessary. The particular selection of API developers to interview benefitted from a wide diversity of viewpoints and opinions, stemming from the divisions within NLM at which they work, their relative experience with APIs and development, and even the nature of the APIs.

For example, the RxNorm, RxTerms, and NDF-RT APIs created by Olivier Bodenreider’s group all operate in several ways: individually, under a graphical client known as RxNav, and under a combinational interface known as RxMix (beta). Each of the APIs, as well as RxNav and RxMix, has related documentation. Additionally, the three content interfaces have two versions of APIs: SOAP/WSDL and REST. These two styles require their own documentation. For a short explanation of a few differences between REST and SOAP, please read Appendix A at the end of this report. RxNav first used SOAP, but it was then made RESTful after users requested it. REST is generally seen as simpler and more efficient, with less overhead, which falls in line with the group’s approach to documentation. They started their documentation focusing on use cases. They only developed and added documentation for functions that people needed or requested.

This is a different approach than the one that the UMLS Terminology Services API developers took. The UMLS API includes documentation for every call and function possible, which results in a large number of methods. The API is SOAP-based; however the team might develop a RESTful interface in the future, in response to recent usage trends of SOAP vs. REST. It is possible that the large number of calls could make documentation be seen as generic or difficult to navigate. On the other hand, one advantage of having documentation on this level of granularity is how thorough it is, including the provision of code samples and the categorization of methods.

However many differences there may be between them, generally the developers felt that a loose set of guidelines for documentation would be beneficial as a jumping off point. They did feel, however, that very strict rules, or a “template” format of documentation, might not be the best possible avenue of approach. The varied natures of the data sets within NLM APIs might not all fit well in the same format, and functionality could suffer because of that. Instead the developers felt that there should be more of a focus on the inclusion of design details: each functions should contain a description, a full list of input and output parameters, and examples.

*ENVIRONMENTAL SCAN*

The environmental scan provided several ideas for different elements that should be included in good documentation. One thing was made very clear: there currently are not any official guidelines or standards in the industry for documenting APIs. The APIs and blogs researched provided many options and a lot of advice, but no true standards.

Based on the environmental scan, there are four main components of good API documentation: overviews, formatting, examples, and reference material. The components themselves are fairly straight-forward, and there are iterations of each in most of the reviewed APIs.

An overview quickly explains what data is found in the API, as well as what developers can do with that data. This is extremely important to include, because if developers do not understand this right away, it is possible that they will move on to another resource to find the information they need. Developers want to get started using an API as quickly as they can, and they may need help finding what they need. That is why clear formatting is important. Examples are also incredibly important, especially when they appear in as many programming languages as the particular API allows. The reference material lists all of the functionality of an API in detail. This includes examples of request and response values. It would also include any tutorials or frequently asked questions.

*CREATING A SET OF GUIDELINES*

Using what was learned from the SWOT analysis, the analysis of NLM API documentation, the developer interviews, and the environmental scan, a group of elements was selected to serve as a set of guidelines for NLM’s API documentation. Each method of research applied led to answering the question of which documentation elements should be included and omitted. The SWOT analysis reinforced the importance of well-designed documentation as a whole, necessitating the inclusion of elements that promote simple navigation and reference tools for new users. The analysis of NLM API documentation emphasized the fact that attention should be focused on commonly used elements, and also that there was a need to look externally to find a comprehensive answer to the question of standards. The interviews conducted afforded the opportunity to meet with developers and find out what they really wanted out of documentation. The developers said that examples are the key: examples for requests and responses, examples for code, and examples for use cases. They also said that, when looking into making a set of guidelines, the focus should be on making them open for flexibility and interpretation. The environmental scan led to the four general components of effective documentation.

In considering the trends of documentation and the research findings, the following is a list of fifteen elements that should be included and serve as a guideline for good API documentation. Each of the fifteen elements falls under one of the four main components. For a full list of the guideline elements, including examples from NLM APIs, see Appendix B: The Guidelines.

| Overview |
| --- |
| Reference or Summary Pages |
| Authentication |
| Error Handling |
| Versioning History |

| Formatting |
| --- |
| Clean Overview Page |
| Navigational Breadcrumbs/Sidebar |
| Section Headings |

| Examples |
| --- |
| Sample Code |
| Focus on Use Cases |
| Calls in Several Languages |

| Reference Materials |
| --- |
| Parameter Descriptions |
| Request and Response Examples |
| Terms and Conditions |
| Link to FAQ or Tutorials |
| Link to Data Download |

***Overview***

The overview component consists of four elements: reference or summary pages, authentication, error handling, and versioning history. This component is especially important to include, because this information may be obvious to the people developing the API, but it might not be as obvious for potential users.

The reference or summary page should be clearly visible on the main page of documentation. It serves to describe the nature of the API, including who should use the API and what they would use it for. It also explains what advantages developers would have in using the API, and sometimes even provide an architectural description of the API platform. Basically, it is an introduction to the API.

Authentication is an important element to have for security in RESTful APIs. SOAP based APIs already support WS-Security for enterprise security features. For RESTful APIs, look into HTTP Basic/Digest or SSL for authentication, as well as encryption and digital signatures.

Error handling is important for RESTful APIs because it helps make the API more intuitive. From the perspective of an outside developer using the API, errors become a tool in providing context and visibility into how to use an API. Developers tend to learn to write their code through errors in a test driven development model that represents a natural way for developers to work1. Error handling does not have to be complicated. Comply with best practices by using a few HTTP status codes. When an application and an API interact, there are basically three possible outcomes: everything worked, the application did something wrong, or the API did something wrong. This could translate in HTTP to: 200 – OK, 404 – Not Found, and 500 – Internal Server Error. It is important that the return code can be consumed and acted upon by the user.

Versioning is equally important for RESTful APIs. Applications and their corresponding APIs evolve over time, thanks to changing parameters and user needs. It is not necessary to design a new version of the API from scratch. Over time new resources and new attributes to each resource may be added, but the method users follow to access those resources should not change. Maintaining earlier versions of an API and allowing access to those versions for restricted uses makes APIs backwards compatible without affecting current version users2. The best course of action is to use a URL prefixed API version for a limited time to gather and keep a collection of resources and methods separate across different versions.

***Formatting***

Clear formatting is a key in good documentation. Users of an API need to know how to navigate the documentation page quickly in order to find what they want. This is why including elements like a clean overview page, navigational breadcrumbs/sidebar, and section headings are so important.

Having a clean overview page is simple. It means making sure that all the documentation elements are visible, with many appearing “above the fold” of the web page. This may be difficult to accomplish without making the page look cluttered. However, documentation should not be spread out onto many different pages. Minimize clicking and keep related topics close to each other on the same page.

This can be done by using navigational breadcrumbs or a persistent navigation sidebar. Using these allows users to see the big picture of a long single page guide by easily zooming into details. This also allows users to search all of a documentation page’s content with a quick in-page browser search.

Finally, include section headings. This is an element that may be passed over by API developers, but it will be of benefit to users new to navigating an API. The section headings have the added benefit of serving as labels for the navigational sidebar.

***Examples***

Developers learn new platforms and APIs by beginning with code that they know works correctly, then modifying. Most developers will find it easier to learn by doing, rather than by reading3. Include sample code for as many methods and calls as possible. All variable, class, member, and function names should be clear. Additionally, in good sample code, relevant information should be grouped together, clarity is more important than efficiency, and simplicity is more important that a good-looking user interface.

Use cases can also be effective in teaching developers how to use APIs, but they are only as effective as that value a user can derive from them. Use cases, seen as possible scenarios for interaction, should be easily understood. When writing them, focus on well-written simplicity rather than over-complicated details. Use cases are seen as an iterative process that can be reworked and refined4. Also, remember the 80/20 rule: 20% of use cases with typically account for 80% of activity in an API. Consider this when deciding whether or not to make an exhaustive list.

When you are writing sample code and use cases, also remember that developers program in multiple languages. To make platform adoption simpler, and to avoid making developers learn a new language just to use your API or read your documentation, list examples in all the languages your API supports, whether they are curl, Ruby, Python, Java, C#, PHP, or others.

***Reference Materials***

Reference material tends to make up the majority of API documentation. There needs to be detailed information about what each class, member, function, XML element, and call is and how it is used. Keep the reference documentation in a consistent style. Include short descriptions of all parameters, including type. Additionally, list every possible request (query) and response possibility, with descriptions.

Including terms and conditions for use is also very important. For NLM specifically, it is recommended that API documentation authors include a disclaimer statement found on the NLM API web page:

“This product uses publicly available data from the U.S. National Library of Medicine (NLM), National Institutes of Health, Department of Health and Human Services; NLM is not responsible for the product and does not endorse or recommend this or any other product.”

This data use disclaimer serves to allow outside developers the free and open use of any data set found in the NLM APIs, while separating NLM from any responsibility or liability in regards to applications created by third-party developers.

Additionally, reference materials should include links to frequently asked questions or tutorials, as well as links to download the original data, if it is ever deemed necessary for developers.

***DISCUSSION***

NLM makes it a point to follow standards and best practices. However, what happens when there aren’t any, and everyone has their own ideas? This is a chance for NLM to become a real leader and point of reference in the area of API documentation standards. There is a real need for guidelines, and NLM can be the organization that establishes them.

There is a lot of advice in the industry for developers who want to write documentation. This advice varies greatly in quality. Following these guidelines will help API developers write clear and comprehensive documentation, helping them focus their time and attention to get the best effect and the best experience for their users.

The Federal government has embraced API development thanks to the Open Government Initiative and the idea of open data. However, how truly useful or open is that data if people do not know how to extract or manipulate it? That is where the importance of good documentation practice really comes into focus. Clear documentation makes developers want to use NLM’s APIs and increases the likelihood of platform adoption. It allows developers to create applications that integrate data in ways that highlight new levels of accessibility and functionality.

It is our responsibility as government representatives and librarians to do whatever we can to make information more accessible and open to the public, as well as respond to the technological demands of our users. To quote the Digital Government Strategy, we need to “require that newly developed IT systems are architected for openness and expose high-value data and content as web APIs at a discrete and digestible level of granularity”. One way we can do that is through the development and implementation of clear, effective API documentation standards.

***RECOMMENDATIONS***

NLM could consider forming a working group at NLM of API developers who can collaborate to adopt these guidelines and look into the possible implementation of other trends in documentation popular in external APIs.

Based on findings from researching external API development trends and through discussions with some of NLM’s API developers, and in hopes of increasing the usability of open data, linking APIs and datasets is the way to really get true utility out of the datasets. Some groups at NLM have already started. RxMix was created by Olivier Bodenreider’s group as an interface for building applications that allows users to combine functions of the RxNorm, RxTerms, and NDF-RT APIs. Their efforts have been met with overwhelmingly positive feedback from users.

NLM’s API developers can also take advantage of other trends in API design, such as interactive documentation. These clients, such as I/O Docs from Mashery, allow you to execute live API calls right from your documentation. This allows users access to play around with calls without having to code and lets them really run APIs through their paces, discovering their true potential. According to the SWOT analysis and the interviews with several developers, increasing accessibility will increase adoption. That type of simple, fluid functionality is something we can achieve.

***REFERENCES***

1. Mulloy B. RESTful API Design: what about errors? [Internet]. 2011 Dec 06. Available from: <https://blog.apigee.com/detail/restful_api_design_what_about_errors>
2. Best practices for API versioning? [Internet]. 2008 Dec 23. Available from: <http://stackoverflow.com/questions/389169/best-practices-for-api-versioning>
3. Gruenbaum P. A coder’s guide to writing API documentation. MSDN Magazine [Internet]. 2010 Nov. Available from: <http://msdn.microsoft.com/en-us/magazine/gg309172.aspx>
4. Levy D. Use Case Examples – Effective Samples and Tips [Internet]. 2013 June 12. Available from: <http://www.gatherspace.com/static/use_case_example.html>
5. Cerami E. Top Ten FAQs for Web Services. O’Reilly Media Inc. [Internet]. 2002 Feb 12. Available from: <http://www.oreillynet.com/lpt/a/webservices/2002/02/12/webservicefaqs.html>
6. Francia S. REST Vs SOAP, The Difference Between Soap and Rest [Internet]. 2010 Jan 15. Available from: <http://spf13.com/post/soap-vs-rest>
7. Jacobson D, Brail G, and Woods D. APIs: A Strategy Guide. O’Reilly Media Inc. 2011 Dec.

***APPENDIX A: A Short Primer on REST vs SOAP***

Web APIs tend to fall under one of two architecture frameworks: Simple Object Access Protocol (SOAP) and representational state transfer (REST). SOAP based APIs are historically considered more of a web service, designed to support interoperable machine-to-machine interaction over a network. They have interfaces described in Web Services Description Language (WSDL), and use XML-based Remote Procedure Calls (RPCs) to transmit messages through HTTP5. They also have the added benefit of increased security, thanks to the support of WS-Security which supports identity through intermediaries and provides a standard implementation of data integrity and privacy6. While SOAP-based web services are useful and important, recent years have found developers moving away from SOAP-based resources towards “lighter-weight” RESTful APIs, which are seen as generally simpler than SOAP in terms of development. RESTful implementations divide into a set of resources based on unique URI patterns and use standard HTTP verbs (GET, POST, PUT, and DELETE) to map operations on top of these resources through readable XML or JSON7.

***APPENDIX B: THE GUIDELINES***

The following is a list of the 15 guidelines for API documentation based on best practices, including a brief description of each and a screenshot of their use within NLM APIs.

***Overview***

* 1. **Reference or Summary Pages**

*Describes the nature of the data found in the API, as well as its intended use and audience. Explains the advantage of the use of the API over others. Serves as an introduction to the API.*

*Example from ChemSpell:*

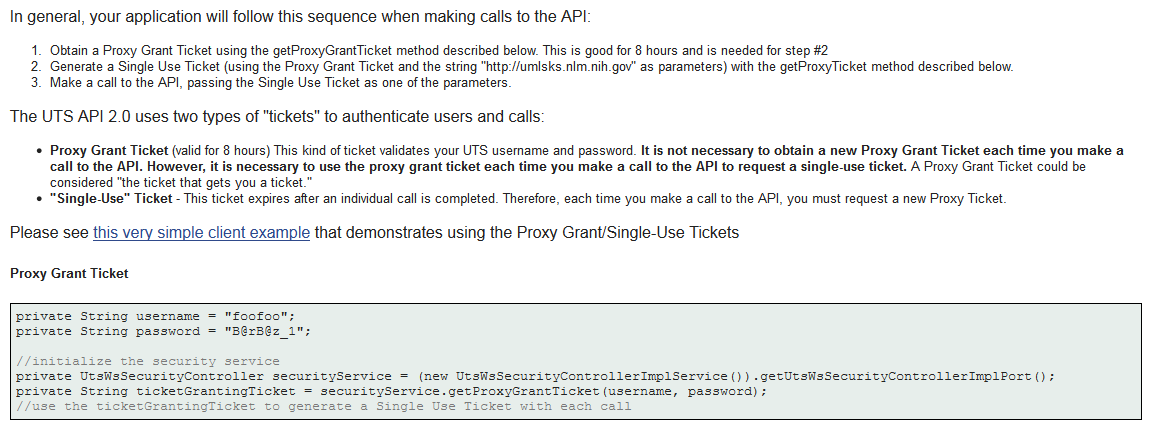
*The ChemSpell Web Service API provides chemical name spell checking and chemical name synonym look-up. ChemSpell contains more than 1.3 million chemical names related to organic, inorganic, pharmaceutical, toxicological, and environmental health topics. ChemIDplus is the source for ChemSpell. When a chemical name is entered, ChemSpell returns a list of chemical synonyms if the spelling is correct. If the spelling is incorrect, it returns a list of suggestions for alternate spellings.

Software developers can write applications that connect remotely to the ChemSpell Web service. Communication is performed via Simple Object Access Protocol (SOAP), an XML-based mechanism for exhanging typed information.*

* 1. **Authentication**

*Allows for the secure encryption of calls between clients and servers. Secures back-end resources for the open development of applications. Include a registration or licensing policy. [REST] Depending on the complexity of the API, use HTTP Basic, SSL, or OAuth 2.0. [SOAP] Use WS-Security.*

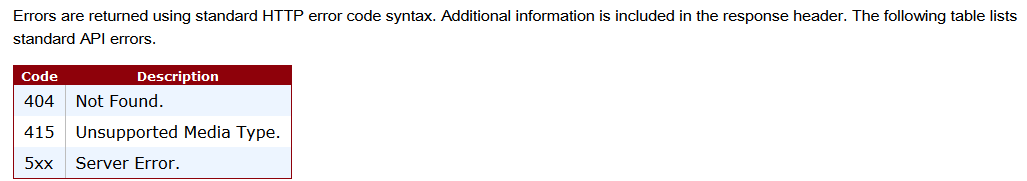
*Example from UMLS Terminology Services:*



* 1. **Error Handling**

*Sends return codes to users for troubleshooting issues. Keep this simple and use only a few HTTP codes. Distinguish between communication error codes and application-level issue codes.*

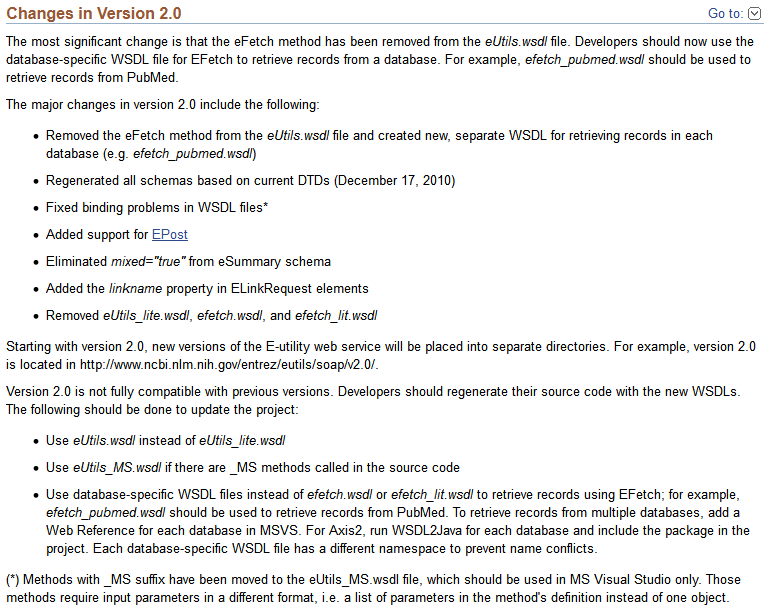
*Example from DailyMed:*

**

* 1. **Versioning History**

*Track and manage evolving information. View and recover earlier versions. Limit the number of past versions stored to manage space and avoid confusion. List specific changes with descriptions.*

*Example from Entrez Programming Utilities:*

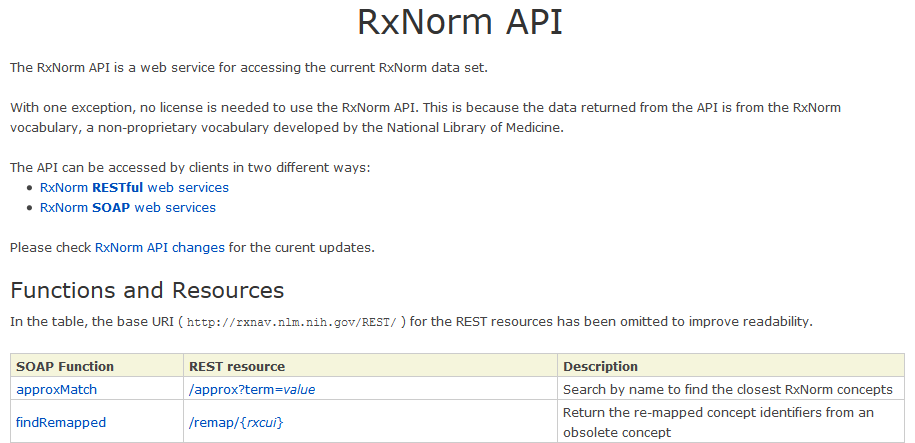
**

***Formatting***

**2.1 Clean Overview Page**

*Most documentation elements are visible from the front page without clicking. Documentation is not spread out among many pages. Related elements and topics are grouped close together.*

*Example from RxNorm:*

**

**2.2 Navigational Breadcrumbs/Sidebar**

*If documentation is spread out, breadcrumbs are clearly visible and descriptive. If used, include on all pages for uniformity. Sidebar includes quick links to all relevant documentation elements, listed in the order they appear on the page.*

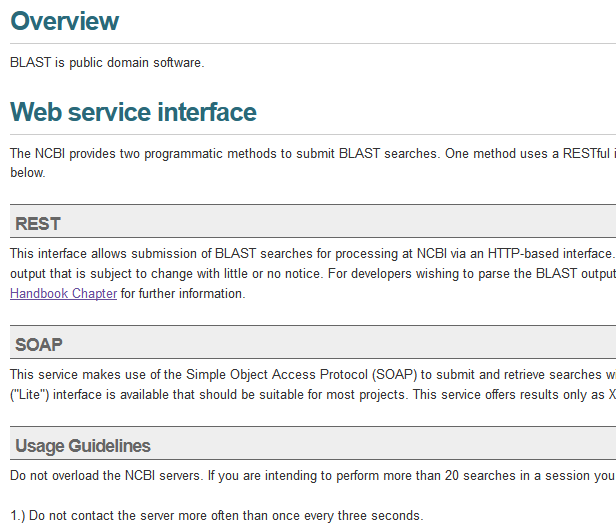
*Example from Entrez Programming Utilities:*

**

**2.3 Section Headings**

*Heading levels are clear and highly differentiated in font and style. Maintain uniformity among levels. Use section heading titles as labels for navigational sidebar (shortened, if necessary).*

*Example from BLAST:*

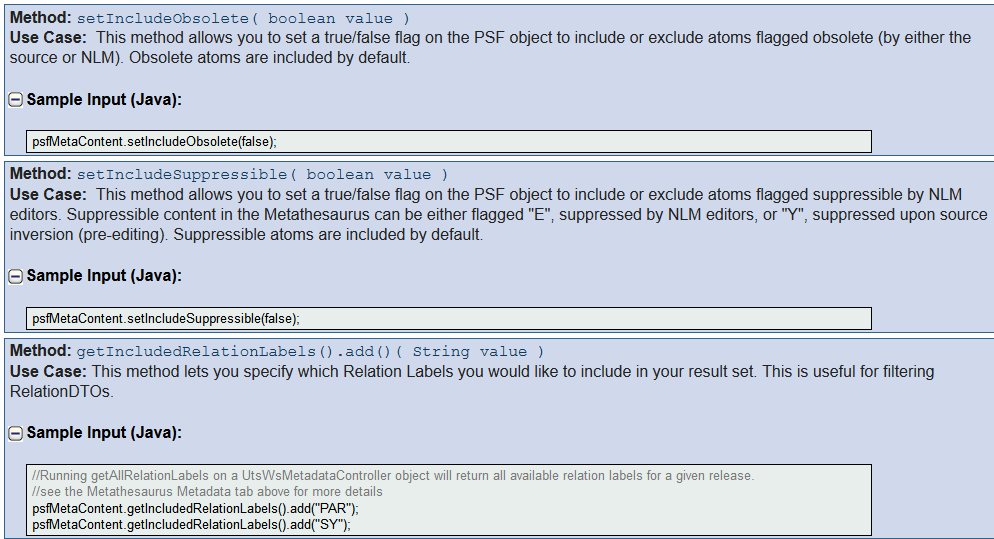
**

***Examples***

**3.1 Sample Code**

*Used by developers to learn how to manipulate APIs. Provide as many samples for methods and calls as possible. Relevant information is grouped together.*

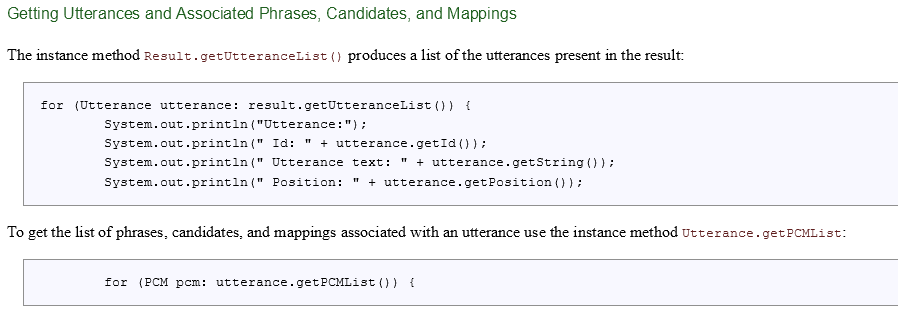
*Example from UMLS Terminology Services:*

**

**3.2 Focus on Use Cases**

*Examples of how to perform key calls in an API service. Thought of as scenarios made up of activities a developer might want to do. Should be simple and clearly written. Not necessarily an exhaustive list (80/20 Rule). Make them as generalizable as possible.*

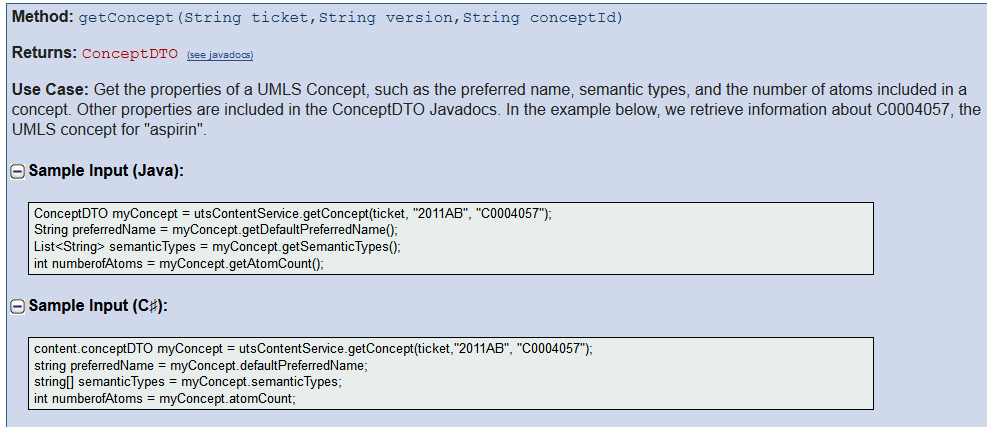
*Example from MetaMap:*

**

**3.3 Calls in Several Languages**

*Allow calls in Python, Ruby, Java, C#, PHP, and more. If impractical, focus on the languages most used by the users accessing the API.*

*Example from UMLS Terminology Services:*

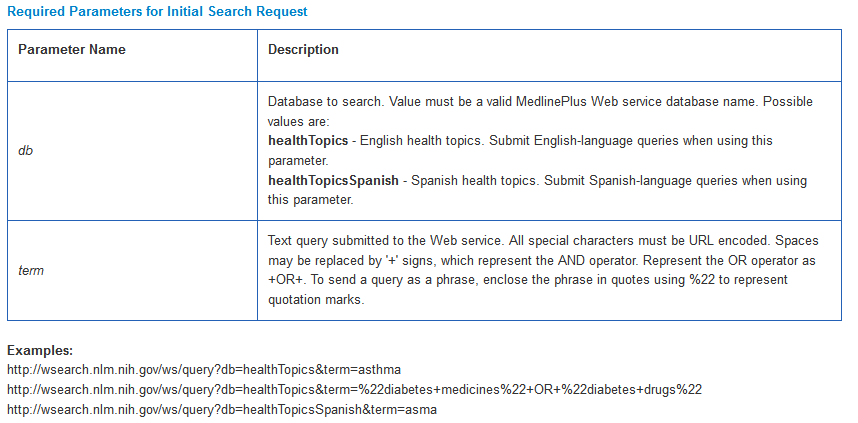
**

***Reference Materials***

**4.1 Parameter Descriptions**

*Clearly list all variable, class, member, and function names.*

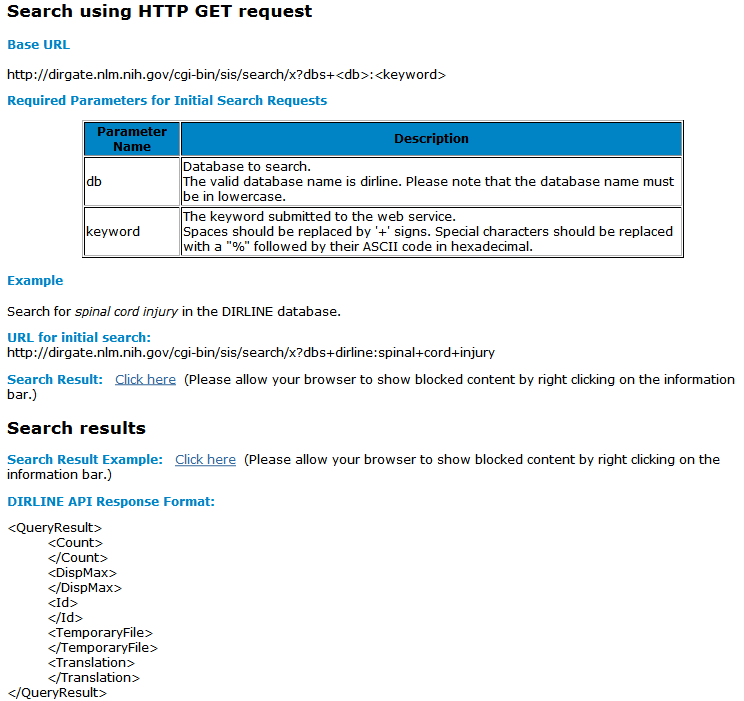
*Example from MedlinePlus Health Topics:*

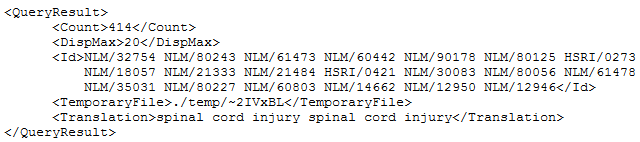
**

**4.2 Request and Response Examples**

*Include HTTP, XML, and JSON samples of requests and responses whenever possible. Describe the purpose of every call and explain every element. Include base URL and request URL for each example.*

**Example from DIRLINE:**

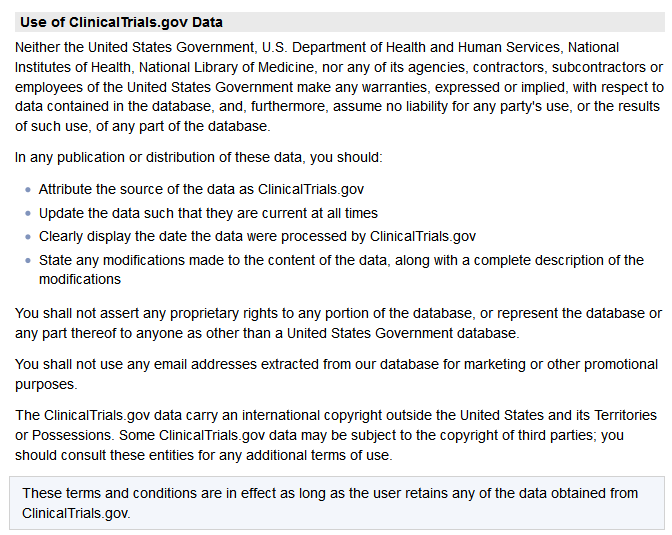
****

****

**4.3 Terms and Conditions**

Allows outside developers free and open use of data in the NLM APIs, while separating NLM from any responsibility or liability in regards to applications created by third-party developers.

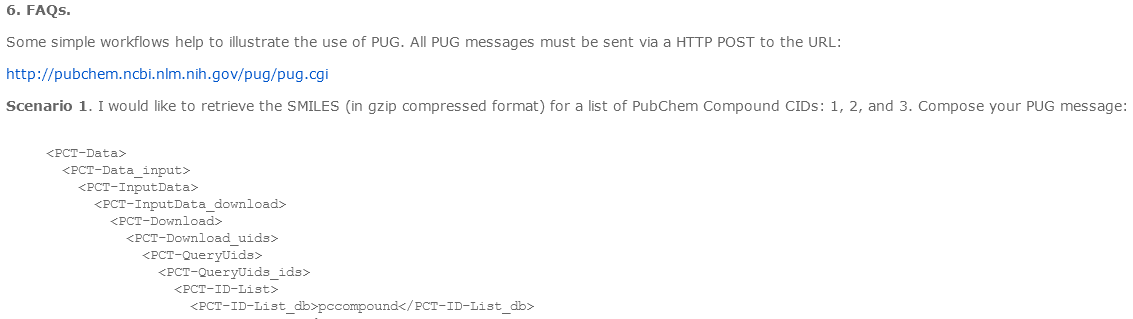
*Example from ClinicalTrials.gov:*

**

**4.4 Link to FAQ or Tutorials**

*Offers additional assistance to users through specific help examples and answers to common questions. May include a “Contact Us” link for more information.*

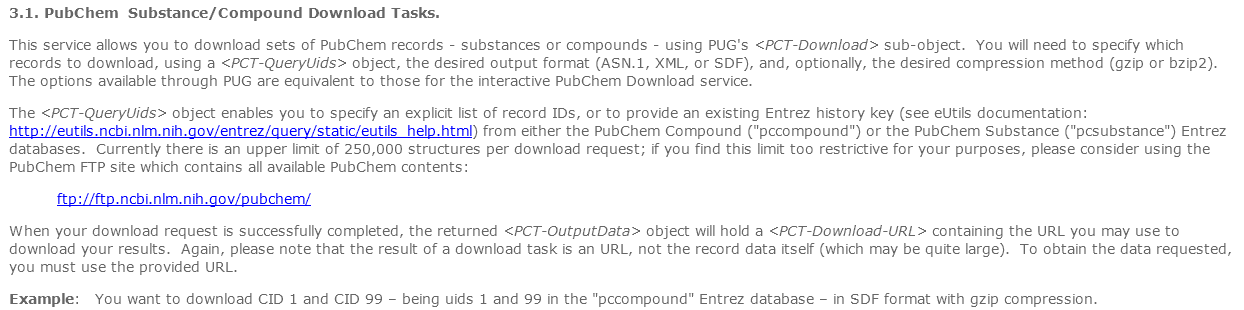
*Example from PubChem Power User Gateway (PUG):*

****

**4.5 Link to Data Download**

*Offers the ability to directly download API’s data sets if requested by the user. May require authentication.*

*Example from PubChem Power User Gateway (PUG):*

**