Internet Engineering Task Force
INTERNET-DRAFT
Expires: December, 2005
Informational

Jacob Palme
Stockholm University
Royal Institute of Technology
7 June 2005

The IETF Golden Rules

draft-palme-golden-rules-00.txt

1. Abstract

This memo presents the following rules, which to some extend can be regarded as the golden rules of IETF, even though there are exceptions when these rules should not be adhered to.

- Be liberal in what you accept, and conservative in what you send
- Do not munge forwarded data
- Modify as late as possible
- Cause no harm
- Leave nothing undefined
- Keep it simple, stupid
- No voting, rough consensus
- Plain ASCII text is enough

2. IPR Statement

3. Table of contents

1. **Abstract** 1
2. **IPR Statement** 1
3. **Table of contents** 2
4. **Mailing list and web area** 3
5. **Introduction** 3
6. **Rule one: Be liberal in what you accept, and conservative in what you send** 3
7. **Rule two: Do not munge forwarded data** 6
8. **Rule three: Modify as late as possible** 7
9. **Rule four: Leave nothing undefined** 8
10. **Rule five: Cause no harm** 8
11. **Rule six: Keep it simple, stupid** 8
12. **Rule seven: No voting, rough consensus** 9
13. **Rule eight: Plain ASCII text is enough** 10
14. **Contradiction between the rules** 11
15. **Other documents** 11
16. **References** 12
17. **Acknowledgements** 12
18. **Author's address** 12
19. **Intellectual Property Statement** 12
20. **Full Copyright Statement** 13

4. Mailing list and web area

Comments and discussion on this MEMO can be sent to the ietf-golden@dsv.su.se mailing list. More information about this list, as well as information how to subscribe, unsubscribe and view the archives of the list, can be found at http://lists.dsv.su.se/cgi-bin/mailman/listinfo/ietf-golden

The latest version of this memo can be found at http://dsv.su.se/jpalme/ietf/golden-rule/.

5. Introduction

The Internet Engineering Task Force (IETF) is the largest and most successful standards organization in the area of Internet protocols. The success of IETF, as compared to other standards making organization, depends on its written and unwritten rules. This memo summarizes some of the most important of these rules. The rules are not carved in stone, there are cases where the IETF itself has chosen to partially break these rules. Still, some of these rules are important and need to be written down.

This memo presents the rules in a rough priority order, with the most important rule first. For each rule, some explanation is given, as well as some examples of how it has been used.

6. Rule one: Be liberal in what you accept, and conservative in what you send

This is the oldest, most wellknown and most important of the rules. It is usually attributed to [RFC1123]. Below is a direct quote of the text in [RFC1123] which explains this rule:

> At every layer of the protocols, there is a general rule whose application can lead to enormous benefits in robustness and interoperability:
>
> > "Be liberal in what you accept, and conservative in what you send"
>
> Software should be written to deal with every conceivable error, no matter how unlikely; sooner or later a packet will come in with that particular combination of errors and attributes, and unless the software is prepared, chaos can

> ensue. In general, it is best to assume that the network is filled with malevolent entities that will send in packets designed to have the worst possible effect. This assumption will lead to suitable protective design, although the most serious problems in the Internet have been caused by unenvisaged mechanisms triggered by low-probability events; mere human malice would never have taken so devious a course!
>
> Adaptability to change must be designed into all levels of Internet host software. As a simple example, consider a protocol specification that contains an enumeration of values for a particular header field -- e.g., a type field, a port number, or an error code; this enumeration must be assumed to be incomplete. Thus, if a protocol specification defines four possible error codes, the software must not break when a fifth code shows up. An undefined code might be logged (see below), but it must not cause a failure.
>
> The second part of the principle is almost as important: software on other hosts may contain deficiencies that make it unwise to exploit legal but obscure protocol features. It is unwise to stray far from the obvious and simple, lest untoward effects result elsewhere. A corollary of this is "watch out for misbehaving hosts"; host software should be prepared, not just to survive other misbehaving hosts, but also to cooperate to limit the amount of disruption such hosts can cause to the shared communication facility.

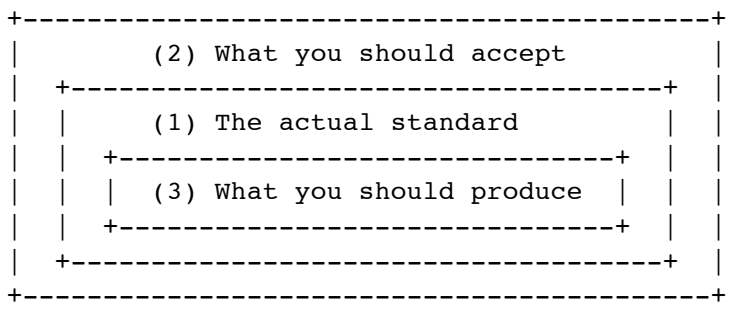
One could interpret this rule to mean that there are actually three versions of each standard:

(1) The actual standard as specified in the standards specification.

(2) What good implementers should accept, and which is a superset of (1).

(3) What good implementers should produce, and which is a subset of (1).

This is shown in this figure:



But if there are in reality three standards, would it not be better if the standards document described all three of them? Well, this would

make the standards documents much larger. However, it is more and more common that standards especially specify, at least in some cases, the difference between each standard, for example by saying that certain functions are deprecated or obsolete. Functions marked as deprecated or obsolete belong to category (2) above, what you should accept but not produce.

Example, quote from [RFC2068], the HTTP standard:

> The "charset" parameter is used with some media types to define the character set (section 3.4) of the data. When no explicit charset parameter is provided by the sender, media subtypes of the "text" type are defined to have a default charset value of "ISO-8859-1" when received via HTTP. Data in character sets other than "ISO-8859-1" or its subsets MUST be labeled with an appropriate charset value.
>
> Some HTTP/1.0 software has interpreted a Content-Type header without charset parameter incorrectly to mean "recipient should guess." Senders wishing to defeat this behavior MAY include a charset parameter even when the charset is ISO-8859-1 and SHOULD do so when it is known that it will not confuse the recipient.

This text thus says that it is allowed to send a text with HTTP without a character set indication, but that it MAY be good to include a character set indication, even when it has the default value ISO-8859-1. A good (conservative) implementation should thus always indicate the character set, even though it is not mandatory.

It is unfortunate that the HTTP standard says that when no character set is specified, the default is ISO-8859-1, while the e-mail standards say that when no character set is specified, the default is plain ASCII. This discrepancy between two mayor standards has caused lots of confusion and incorrect implementations. The robustness rule quoted above says that it is good practice to always identify the character set of HTTP-transmitted text, even though this is formally optional.

The e-mail standard [RFC2046] says:

> The default character set, US-ASCII, has been the subject of some confusion and ambiguity in the past. Not only were there some ambiguities in the definition, there have been wide variations in practice. In order to eliminate such ambiguity and variations in the future, it is strongly recommended that new user agents explicitly specify a character set as a media type parameter in the Content-Type header field. "US-ASCII" does not indicate an arbitrary 7-bit character set, but specifies that all octets in the body must be interpreted as characters according to the US-ASCII character set. National and application-oriented versions of ISO 646 [ISO-646] are

> usually NOT identical to US-ASCII, and in that case their use in Internet mail is explicitly discouraged. The omission of the ISO 646 character set from this document is deliberate in this regard. The character set name of "US-ASCII" explicitly refers to the character set defined in ANSI X3.4-1986 [US-ASCII]. The new international reference version (IRV) of the 1991 edition of ISO 646 is identical to US-ASCII. The character set name "ASCII" is reserved and must not be used for any purpose.

Note that again, the standard says that the Character-set attribute is not mandatory, but good (conservative) implementations should always specify it for textual body parts in e-mail. Both e-mail and HTTP standards does say that this attribute is optional but should be used in what you produce.

One of the most comprehensive explicit specification of the difference between the recommended and the allowed standard is the e-mail body format standard, [RFC2822]. This standard has a special chapter 4, with the title "Obsolete syntax". This chapter goes into great detail in specifying everything a good implementation should accept, but not produce. Probably, such explicit specification of what a good implementation should accept but not produce will occur in the future. When a standard grows old, the experience on how to interpret the liberal-conservative rule increases.

Note: By "produce" in Rule one is meant new data which an agent produces. It does normally not apply to data which an agent receives and forwards. For such data, instead, the "Rule two: Do not munge forwarded data" below applies.

7. Rule two: Do not munge forwarded data

It may seem natural for an implementer, when receiving incorrect data, to correct it before sending it further. Such corrections of incorrect data is in IETF vocabulary called "munging", and it is in general disapproved. The reason for this is that long experience has shown that attempts to correct incorrect data will usually cause more confusion than keeping it as it is. This might be seen as a contraction to the liberal-conservative rule, but it is not, since the liberal-conservative rule talks about what you produce, not about what you forward.

A commonly occuring case where implementers wrongly do not adhere to "Rule two: Do not munge" is certain mail systems which when it receives a message converts it to an internal format. When the same message is then sent out again, it is converted back again to standard format. The

correct behaviour of such systems is that even if they need to convert the message to an internal format to be able to handle it locally, they should keep a copy of the original message before conversion, and use this when sending it out again. A much-used mailer will for example change the Message-ID when resending a message. Doing this is bad, because it can cause mailing-list loops, because threads are broken and for other reasons.

There are of course cases where the "do not munge" rule should not be adhered to. A well-known example is certain attributes in submission to Mail Transfer Agents (MTAs). Some MTAs will correct incorrect mail by adding missing "Date:" and "Content-ID" headers. This is in general not good. For example, if an MTA adds "Content-ID" to a message which it receives by SMTP from another MTA, different versions of the same message may turn up with different values of the Content-ID, which can cause various problems. However, the first MTA which receives a message from the originating Message User Agent (MUA) can add a missing Content-ID without this risk. Therefore, a specific standard has been written, [RFC2476] which specifies what kind of munging is allowed and not allowed for the first MTA which receives a message from an MUA. There are certain advantages with doing this in some cases, for example, the submission MTA may have a more reliable clock than the originating MUA.

8. Rule three: Modify as late as possible

If you absolutely have to modify incoming data, this modification should be done as late as possible. If possible, standards should be written so that data need not be modified. The reason for this rule is that modifying or converting data often causes some damage to the data (either loss of information or addition of possibly incorrect information or both). And if this conversion is done as late as possible, it is easier for the recipient to control how it is done. For example, if you receive an HTML document which your browser cannot display properly, you can choose to display it using another web browser instead of converting it. Another reason is that you may need to send out what you have received. And when re-sending information, "Rule two: Do not munge" usually means that the original document should be sent, unless you explicitly under manual control make changes to it or convert it.

This is the reason, for example, for the use of the Content-Location heading in the MHTML standard [MHTML]. Using Content-Location, HTML documents can be fetched unchanged from a web site and put into e-mail. The alternative (Content-ID, which is also allowed according to MHTML)

usually requires rewriting of the HTML code when it is to be sent by e-mail.

9. Rule four: Leave nothing undefined

When in a trouble about how to describe a certain function, there is a simple, but bad, way out. That is to leave the function undefined. This is bad, because implementors may choose to implement this function in different ways, which can cause interoperability problems.

10. Rule five: Cause no harm

Refractory implementers may follow a standards specification to the letter, but interpret it in ways which means that other agents receiving the information may be confused or even crash. Thus, the "cause no harm" rule says that you should not produce information which has a potential of causing harm to a well-working internet. Examples of this is to send unreasonably long data, which might cause so-called "buffer overflows" which are known to be a major cause of security problems in standards. An example is lines in e-mail, where the [RFC2822] standard says:

> Each line of characters MUST be no more than 998 characters, and SHOULD be no more than 78 characters, excluding the CRLF.

Note that it is possible to send e-mail with more than 78 characters per line, using either the MIME [RFC2046] standard and the BASE64 or the Quoted-Printable encodings, or by using the format=flowed format [RFC 2646].

11. Rule six: Keep it simple, stupid

There is a tendency for standards developers to make standards overly complex. One reason for this is that a simple way out of disagreements and reaching consensus is to include everything anyone wants in a standard.

There is also an obvious risk that people from large software vendors may prefer a complex standard, since this reduces the number of competitors who have the capacity to implement the full standard.

IETF is not immune from this tendency. For example the HTTP standard does probably provide unnecessarily too many ways of indicating that a page should not be automatically saved in any way in the receiving

computer. I read, in a bulletin board, the following recommendation of how to use HTTP to reduce the risk that a page is saved:

```
Expires: Mon, 26 Jul 1997 05:00:00 GMT                    ; Date in the past
Last-Modified: 4 June 2005 15:20:00 GMT                   ; Current date
Cache-Control: no-store, nocache, must-revalidate, max-age=0    ; HTTP/1.1
Cache-Control: no-store, no-cache, must-revalidate, max-age=0   ; HTTP/1.1
Pragma: no-cache                                          ; HTTP/1.0
```

The reason this recommendation is given, is that some HTTP implementations may not support all the methods, so therefore it is safest to use all of them. This is a typical example of the risk that complex standards cause different implementations to support different subsets of the standard, which can cause incompatibility problems.

Note: There are certainly reasons why the HTTP authors have chosen to provide so many ways of specifying this.

In spite of what is said above, IETF standards are usually much simpler than, for example, the competing OSI standards developed by ISO and ITU. And this is probably a major reason why IETF standards usually have been more successful than competing ISO and ITU standards.

The rule that any feature not implemented in two co-working independent standards, before an IETF standard is progressed from "Proposed standard" to "Draft standard" has certainly contributed to making the standards simpler.

The advantage with simpler standards are that they are usually less easy to misunderstand by implementers. Also, with a complex standard, the risk is larger that two implementers will implement non-compatible subsets of the standard. For ISO/ITU standards, a special set of standards called "functional standards" specify good subsets to the ISO/ITU standards. Such functional standards are not as much needed for IETF standards, since they are simpler from the beginning.

12. Rule seven: No voting, rough consensus

IETF developers often claim that IETF has a rule of "no voting". Anyone who has participated in IETF meetings know that this is not true, some kinds of voting does happen now and then. Voting, however, is not part of the formal decision process in IETF, as it is in ISO and ITU.

One reason for this is that all are not equal in standards work. Opinions of people known to be reasonable and knowledgeable are more

important than those of other people participating in the standards work.

Another reason is that there is no voting procedure which will always produce the optimal result. This problem has been extensively discussed in the political science sub-area with the name "social choice theory". A simple example to illustrate the issue. Suppose we are to choose between three solutions.

	very good	good	OK	bad	very bad
Solution A	40 %				60 %
Solution B	35 %				65 %
Solution C	25 %	75 %			

It is not obvious, in the above situation, what is the best choice to make in a standards making organisation. Probably solution C is best, since everyone thinks it is good, even though it has the least number of people who thinks it is very good.

I have in fact implemented a voting program with the special goal of being suitable for IETF. In this program, people are given a list of alternatives, and can rate each alternative on a scale from very good to very bad. The results are not presented as numbers. Instead, the results are presented by listing the names of the people who voted in each box in the table as shown in the example above.

"Rough consensus" means that all or most reasonable experts agree, even though some may dissent.

13. Rule eight: Plain ASCII text is enough

This may be the least important of the golden rules, but all IETF standards are written in plain ASCII, with no other formatting than CRLF between the lines and FF between the pages. They are also written to be printed with a monospace font (a font where all characters have equal widths) otherwise some tables or diagrams (like in this memo) may not display well.

This rule is not only for standards text. Also discussion about standards in IETF working groups adhere to this rule. MIME, HTML and

other newfangled ideas are looked down upon in IETF mailing lists. And best is if lines are no longer than 72 characters.

14. Contradiction between the rules

There may, of course, be contradictons between the rules. Note however that Rule two and Rule three are rules about relaying information, while Rule one and Rule five are rules about producing information. The rest of the rules mainly concern how standards are written, rather than how they are implemented.

A well-known example of a contradiction is if you have to combine snipetts of HTML produced by different agents. An error in one of the snippets, for example <font color=white> might then damage HTML coming after the illegally formatted snippets. The best way around this problem is to surround the included HTML with <DIV> and </DIV>, or, even safer, put it into a separate FRAME or IFRAME (although FRAMEs and IFRAMEs may have other drawbacks). Some software also adds finishing HTML for unfinished elements, for example adds </P>, </FONT>, etc. at the end of the incorrectly formatted HTML snippet. These additions may be wrong, but the risk is certainly less than by adding start HTML elements to try to correct incorrect snippets.

15. Other documents

There are many RFCs describing IETF workings and procedure. Some of them are certainly more authorative than this memo. Examples of other such RFCs are (in publication order):

RFC2418: IETF Working Group Guidelines and Procedures, by S. Bradner, September 1998.

RFC3160: The Tao of IETF - A Novice's Guide to the Internet Engineering, by S. Harris, August 2001.

RFC3184: IETF Guidelines for Conduct, by S. Harris. October 2001.

RFC3233: Defining the IETF, by P. Hoffman, S. Bradner. February 2002.

RFC3844: IETF Problem Resolution Process. E. Davies, edited by J. Hofmann, August 2004.

RFC3929: Alternative Decision Making Processes for Consensus-Blocked Decisions in the IETF, by T. Hardie. October 2004.

RFC3935: A Mission Statement for the IETF, by H. Alvestrand, October 2004.

16. References

For more references, see chapter 14 above.

[RFC1123]: Requirements for Internet Hosts -- Application and Support, By Robert Braden, October 1989.

[RFC2046]: Multipurpose Internet Mail Extensions(MIME) Part Two: Media Type, by N Freed and N Borenstein, November 1996.

[RFC2476]: Message swubmission, by R. Gellens and J. Klensin, December 1998.

[RFC2557]: MIME Encapsulation of Aggregate Documents, such as HTML (MHTML), by J. Palme, A. Hoffman and N. Shellness, March 1999.

[RFC2646]: The Text/Plain Format and DelSp Parameter, by R. Gellens, February 2004.

[RFC2822]: Internet Message Format, by Pete Resnick, April 2001.

17. Acknowledgements

Funding for the RFC Editor function is currently provided by the Internet Society.

18. Author's address

Jacob Palme <jpalme@dsv.su.se>
Skeppargatan 73
11530 Stockholm
Phone: +46-8-16 16 67

19. Intellectual Property Statement

be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

20. Full Copyright Statement

This Internet-Draft will expire on 6 December 2005.