Operational Identification of Software Components in the Virtual Observatory

Version 1.0

IVOA Note 2021-01-15

Working group
Operations
This version
http://www.ivoa.net/documents/softid/20210115
Latest version
http://www.ivoa.net/documents/softid
Previous versions
This is the first public release
Author(s)
Markus Demleitner, Mark Taylor
Editor(s)
Markus Demleitner

Abstract

In the Virtual Observatory, software often talks to other software. During development, while gathering statistics (e.g., ignoring validators), or in operations (e.g., enabling workarounds for known bugs), it is often beneficial to know something about the counterpart. This note collects recommended practices on both the client and the server side for HTTP-based protocols. These recommendations make use of the existing standard HTTP headers `User-Agent` for client requests and `Server` for server responses.

Status of this document

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/documents/.

Contents

Acknowledgement

The bulk of this material was taken from a page on the IVOA wiki[1] that resulted from a discussion at the 2018 College Park Interop (client side) and a talk[2] at the 2019 Groningen Interop (server side).

[1] `https://wiki.ivoa.net/twiki/bin/view/IVOA/UserAgentUsage`

[2] `https://wiki.ivoa.net/internal/IVOA/InterOpOct2019Ops/serversoftware.pdf`

Conformance-related definitions

The words “MUST”, “SHALL”, “SHOULD”, “MAY”, “RECOMMENDED”, and “OPTIONAL” (in upper or lower case) used in this document are to be interpreted as described in IETF standard RFC2119 (Bradner, 1997).

The *Virtual Observatory (VO)* is a general term for a collection of federated resources that can be used to conduct astronomical research, education, and outreach. The International Virtual Observatory Alliance (IVOA) is a global collaboration of separately funded projects to develop standards and infrastructure that enable VO applications.

1 Introduction

Very early on in the construction of client-server architectures it was found that it is useful to have mechanisms for discovering which software runs at the other side of a connection, rather typically to aid in debugging. In particular, HTTP, which is the basis of many of the VO’s protocols (Fielding and Gettys et al., 1999), specifies the request header `User-Agent` that identifies the client software (“for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations”) and the response header `Server`.

These general rules were found insufficient for various purposes in the Virtual Observatory. For instance, use case 1.1.1 was brought up at the College Park Interop in 2018. Also, it was found that most servers did not identify the relevant protocol implementations; the server headers rather identified libraries the services were built with or even reverse proxies they ran behind – information that is usually not useful when diagnosing operational problems in the VO, which mostly originate in the higher layers of request handling.

This note gives recommendations on how to make the header fields more useful for VO operations. While strictly only applicable to HTTP-based protocols, we believe similar practices should be implemented where, as with most VOEvent transports, communication mechanisms are not built on top of HTTP.

To facilitate per-client analyses or similar use cases, adopters of this note are requested to give the identification strings for their software on the SoftID page in the IVOA wiki[3].

1.1 Use Cases

[3]`https://wiki.ivoa.net/twiki/bin/view/IVOA/SoftID`

1.1.1 Recognising Maintenance Queries

In the VO, several clients connect to services for operational purposes, for instance in order to perform service validation or monitoring. If service providers are gathering statistics on service usage, they may wish to distinguish these different classes of requests from requests presumably coming from science users.

1.1.2 Dropping Workarounds

Clients occasionally work around bugs in server software; these workarounds over time are a maintenance liability, and hence it is advantageous to drop them when they are no longer needed. To establish when a workaround can safely be dropped, developers need a simple way to enumerate which server software is still running in the VO.

Conversely, a server may contain workarounds for client bugs. Again, being able to find out whether code implementing these can be safely dropped without adverse consequences on users is obviously beneficial.

1.1.3 Client Snooping

A server may offer experimental or advanced features to clients it knows (though such use should in general be frowned upon, as it violates the spirit of interoperability). Similarly, a client might under certain circumstances want to enable or disable certain behaviour when realising it is communicating with a known server component.

1.1.4 Debugging

For diagnosing failures, it is often useful to know which components are part of a communication leading up the failure. This is in particular true to avoid unnecessary analysis when known-obsolete or highly experimental components are the root cause.

1.1.5 Notifications

As part of a responsible disclosure of a software weakness (or simply a request for a software update), server developers might want to contact deployers of vulnerable or otherwise broken software.

1.2 Security and Privacy Considerations

Several guidelines on IT security discourage giving details on the software that drives a certain site in order to not give attackers information that might be useful in an attack.

Following the practices proposed here will, indeed, weaken the "security by obscurity" put forward in these treatments; on the other hand, when, as is the case in the VO, attackers only have to scan perhaps several hundred URLs, relying on security by obscurity does not seem a promising policy.

On the other hand, in the VO, where software providers rather typically are members of the community, and given the Registry which allows rapid discovery of active services, these software providers can contact operators of vulnerable services given sufficiently precise software identification even before a vulnerability is disclosed.

The identification of the client has fewer security implications, as it seems unlikely that rogue services could be aided by information on the client version when they target clients.

Software identification does play a role in user privacy; user agents are regularly employed in user tracking on the WWW. While, presumably, the generally non-profit operators in the VO will not use such data to significantly violate their users' privacy, client authors may want to give users the possibility to somewhat reduce the information content of the headers proposed here.

On the other hand, the mechanisms proposed here are most relevant for automated clients for which there usually are no privacy concerns.

Note also that the recommendations presented in this Note have only incremental impacts on security and privacy; the standard server and client header usage on which they build already have implications in these areas.

2 Client Identification

2.1 User-Agent Header Standard Usage

The HTTP `User-Agent` header may be used by clients to identify their nature or origin. The definition and usage of this header is described in RFC 2616 (Fielding and Gettys et al., 1999) section 14.43, with additional text on syntax in sections 3.8 and 2.2. The definition was updated in RFC 7231 (Fielding and Reschke, 2014), section 5.5.3. The basic rule is that the content of this field should consist of a sequence of tokens, where each token is either a product name (with an optional version indicator), or a free-text comment enclosed in parentheses. Formally (BNF from RFC 2616):

```
User-Agent        = "User-Agent" ":" 1*( product | comment )
product           = token ["/" product-version]
product-version   = token
comment           = "(" *( ctext | quoted-pair | comment ) ")"
ctext             = <any TEXT excluding "(" and ")">
token             = 1*<any CHAR except CTLs or separators>
quoted-pair       = "\" CHAR
```

Additional rules and conventions are that more-significant tokens should appear earlier in the sequence, and that the content should be "short and to the point".

2.2 User-Agent Header IVOA Recommendations

The Operations IG endorses and encourages use of these standard rules concerning the `User-Agent` header, and adds a further convention, which does not conflict with the above rules: clients whose primary purpose is *operational*, as opposed to *scientific*, should indicate that purpose by including a comment token of the form

`(IVOA-<op-purpose> <optional-extra-text>).`

Suggested `op-purpose` values are currently:

test The purpose of the access is to test service availability or performance (monitoring) or standards-compliance (validation); at this point, no good reason to separate the different cases was identified.

copy The purpose of the access is to replicate (parts of) the content published through the service, be it for aggregation (harvesting) or re-publication (mirroring).

This list may evolve in the future; extensions should be proposed on the ops@ivoa.net mailing list. Custom `op-purpose` values are permitted. Case is significant in `op-purpose` values and its "`IVOA-`" prefix.

The `<optional-extra-text>` could be used to indicate a URL at which more information about the client, or perhaps about the results it is gathering from the current request, can be found. However, in accordance with the injunction from RFC 7231 *"A user agent SHOULD NOT generate a User-Agent field containing needlessly fine-grained detail"*, such additional text should be added only when it serves a real purpose.

Formally:

```
ivoa-comment  = "(IVOA-" op-purpose *(
    ctext | quoted-pair | comment ) ")"
op-purpose   = "test" | "copy" | token
```

Tokens of the form `ivoa-comment` should not appear in the `User-Agent` field if the request is a "normal" user science query. There are obviously grey areas between operational and science requests; this convention does not attempt to provide a rigid definition of these categories.

This arrangement allows service operators to test in their logs for `User-Agent` values whose content matches the sequence "`(IVOA-`", or perhaps "`(IVOA-test`", and adjust their usage statistics appropriately. Note, however, that it is not feasible to force operational clients to follow this convention, so service operators will still need to be careful in analysing their usage statistics.

2.3 Examples

A science query from the STILTS tapquery TAP client might contain the HTTP header

```
User-Agent: STILTS/3.1-4 Java/1.8.0_181
```

while a query from the STILTS taplint TAP service validator might contain the header

```
User-Agent: STILTS/3.1-4 (IVOA-test) Java/1.8.0_181
```

or maybe (line break for typographic reasons)

```
User-Agent: STILTS/3.1-4 (IVOA-test
            http://validators.org/results) Java/1.8.0_181
```

The server identification example, tapstat.py, discussed in section 3 illustrates one way to add such headers using Python's built-in urllib.

In Java, an application can be configured to add tokens in this way to the `User-Agent` value for all HTTP client requests by setting the `http.agent` system property.

3 Server Identification

3.1 Server Header Standard Usage

The HTTP `Server` header may be used by servers to identify their implementation software. It is defined by RFC 2616 section 14.38, updated by RFC 7231 section 7.4.2. The header value syntax is identical to that for the `User-Agent` header described in section 2.1.

3.2 Server Header IVOA Recommendations

We recommend using the `Server` header in accordance with standard usage, but VO servers should where possible include product tokens for the VO software actually processing the request. As on the client side, list VO components in front of more generic HTTP server software.

3.3 Examples

A server running DaCHS might take steps to issue HTTP responses containing the header:

```
DaCHS/2.2.1 twistedWeb/18.9.0
```

which would be preferred over the basic

```
TwistedWeb/18.9.0
```

which the server infrastructure might provide by default.

This note comes with an example programme obtaining global server statistics for registered TAP services[4].

3.4 Notes

Use case 1.1.2 might appear to require Registry support for server identification to enable queries like "give me a list of all server software in use in the VO" or "which operators run version 21.2 of software X?" However, given that it is unlikely that the VO will ever host more than a few hundred distinct servers of a given type (under the assumption that each piece of software on a large data centre will serve many different resources), the use cases for global server identification can probably be satisfied by running one request each against these servers, access URLs for which can readily be discovered in the Registry as it is.

A Changes from Previous Versions

No previous versions yet.

References

Bradner, S. (1997), 'Key words for use in RFCs to indicate requirement levels', RFC 2119.
http://www.ietf.org/rfc/rfc2119.txt

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (1999), 'Hypertext transfer protocol – HTTP/1.1', rfc2616.
http://www.w3.org/Protocols/rfc2616/rfc2616.html

Fielding, R. and Reschke, J. (2014), 'Hypertext transfer protocol (HTTP/1.1): Semantics and content', RFC 7231.
https://tools.ietf.org/html/rfc7231

[4]http://www.ivoa.net/documents/softid/20210115/tapstats.py; for the S*AP protocols, some URL heuristics might be necessary in order to achieve true server enumeration.