Network Working Group
Request for Comments: 4329
Category: Informational

B. Hoehrmann

April 2006

Scripting Media Types

Status of This Memo

Copyright Notice

Abstract

This document describes the registration of media types for the ECMAScript and JavaScript programming languages and conformance requirements for implementations of these types.

Table of Contents

1. Introduction

This memo describes media types for the JavaScript and ECMAScript programming languages. Refer to "Brief History" and "Overview" in [ECMA] for background information on these languages.

Programs written in these programming languages have historically been interchanged using inapplicable, experimental, and unregistered media types. This document defines four of the most commonly used media types for such programs to reflect this usage in the IANA media type registry, to foster interoperability by defining underspecified aspects, and to provide general security considerations.

2. Conformance and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119] and indicate requirement levels for compliant implementations. Requirements apply to all implementations unless otherwise stated.

An implementation is a software module that supports one of the media types defined in this document. Software modules may support multiple media types but conformance is considered individually for each type.

Implementations that fail to satisfy one or more "MUST" requirements are considered non-compliant. Implementations that satisfy all "MUST" requirements, but fail to satisfy one or more "SHOULD" requirements, are said to be "conditionally compliant". All other implementations are "unconditionally compliant".

3. Deployed Scripting Media Types and Compatibility

Various unregistered media types have been used in an ad-hoc fashion to label and exchange programs written in ECMAScript and JavaScript. These include:

text/javascript	text/ecmascript
text/javascript1.0	text/javascript1.1
text/javascript1.2	text/javascript1.3
text/javascript1.4	text/javascript1.5
text/jscript	text/livescript
text/x-javascript	text/x-ecmascript
application/x-javascript	application/x-ecmascript
application/javascript	application/ecmascript

Use of the "text" top-level type for this kind of content is known to be problematic. This document thus defines text/javascript and text/ecmascript but marks them as "obsolete". Use of experimental and unregistered media types, as listed in part above, is discouraged. The media types,

* application/javascript
* application/ecmascript

which are also defined in this document, are intended for common use and should be used instead.

This document defines equivalent processing requirements for the types text/javascript, text/ecmascript, and application/javascript. Use of and support for the media type application/ecmascript is considerably less widespread than for other media types defined in this document. Using that to its advantage, this document defines stricter processing rules for this type to foster more interoperable processing.

The types defined in this document are applicable to scripts written in [JS15] and [ECMA], respectively, as well as to scripts written in a compatible language or profile such as [EcmaCompact].

This document does not address scripts written in other languages. In particular, future versions of JavaScript, future editions of [ECMA], and extensions to [ECMA], such as [E4X], are not directly addressed. This document may be updated to take other content into account.

Updates of this document may introduce new optional parameters; implementations MUST consider the impact of such an update. For the application/ecmascript media type, implementations MUST NOT process content labeled with a "version" parameter as if no such parameter had been specified; this is typically achieved by treating the content as unsupported. This error handling behavior allows extending the definition of the media type for content that cannot be processed by implementations of [ECMA].

The programming languages defined in [JS15] and [ECMA] share a common subset. Choice of a type for scripts compatible with both languages is out of the scope of this document.

This document does not define how fragment identifiers in resource identifiers ([RFC3986], [RFC3987]) for documents labeled with one of

the media types defined in this document are resolved. An update of this document may define processing of fragment identifiers.

4. Character Encoding Scheme Handling

Refer to [RFC3536] for a discussion of terminology used in this section. Source text (as defined in [ECMA], section 6) can be binary source text. Binary source text is a textual data object that represents source text encoded using a character encoding scheme. A textual data object is a whole text protocol message or a whole text document, or a part of it, that is treated separately for purposes of external storage and retrieval. An implementation's internal representation of source text and source text are not considered binary source text.

Implementations need to determine a character encoding scheme in order to decode binary source text to source text. The media types defined in this document allow an optional charset parameter to explicitly specify the character encoding scheme used to encode the source text.

How implementations determine the character encoding scheme can be subject to processing rules that are out of the scope of this document. For example, transport protocols can require that a specific character encoding scheme is to be assumed if the optional charset parameter is not specified, or they can require that the charset parameter is used in certain cases. Such requirements are not considered part of this document.

Implementations that support binary source text MUST support binary source text encoded using the UTF-8 [RFC3629] character encoding scheme. Other character encoding schemes MAY be supported. Use of UTF-8 to encode binary source text is encouraged but not required.

4.1. Charset Parameter

The charset parameter provides a means to specify the character encoding scheme of binary source text. Its value MUST match the mime-charset production defined in [RFC2978], section 2.3, and SHOULD be a registered charset [CHARSETS]. An illegal value is a value that does not match that production.

4.2. Character Encoding Scheme Detection

It is possible that implementations cannot interoperably determine a single character encoding scheme simply by complying with all requirements of the applicable specifications. To foster interoperability in such cases, the following algorithm is defined.

Implementations apply this algorithm until a single character encoding scheme is determined.

1. If a charset parameter with a legal value is specified, the value determines the character encoding scheme.

2. If the binary source text starts with a Unicode encoding form signature, the signature determines the encoding. The following octet sequences, at the very beginning of the binary source text, are considered with their corresponding character encoding schemes:

Leading sequence	Encoding
FF FE 00 00	UTF-32LE
00 00 FE FF	UTF-32BE
FF FE	UTF-16LE
FE FF	UTF-16BE
EF BB BF	UTF-8

   The longest matching octet sequence determines the encoding. Implementations of this step MUST use these octet sequences to determine the character encoding scheme, even if the determined scheme is not supported. If this step determines the character encoding scheme, the octet sequence representing the Unicode encoding form signature MUST be ignored when decoding the binary source text to source text.

3. The character encoding scheme is determined to be UTF-8.

If the character encoding scheme is determined to be UTF-8 through any means other than step 2 as defined above and the binary source text starts with the octet sequence EF BB BF, the octet sequence is ignored when decoding the binary source text to source text. (The sequence will also be ignored if step 2 determines the character encoding scheme per the requirements in step 2).

In the cited case, implementations of the types text/javascript, text/ecmascript, and application/javascript SHOULD and implementations of the type application/ecmascript MUST implement the requirements defined in this section.

4.3. Character Encoding Scheme Error Handling

The following error processing behavior is RECOMMENDED for the media types text/javascript, text/ecmascript, and application/javascript, and REQUIRED for the media type application/ecmascript.

- If the value of a charset parameter is illegal, implementations MUST either recover from the error by ignoring the parameter or consider the character encoding scheme unsupported.

- If binary source text is determined to have been encoded using a certain character encoding scheme that the implementation is unable to process, implementations MUST consider the resource unsupported (i.e., they MUST NOT decode the binary source text using a different character encoding scheme).

- Binary source text can be determined to have been encoded using a certain character encoding scheme but contain octet sequences that are not legal according to that scheme. This is typically caused by a lack of proper character encoding scheme information; such errors can pose a security risk, as discussed in section 5.

  Implementations SHOULD detect such errors as early as possible; in particular, they SHOULD detect them before interpreting any of the source text. Implementations MUST detect such errors and MUST NOT interpret any source text after detecting such an error. Such errors MAY be reported, e.g., as syntax errors as defined in [ECMA], section 16.

This document does not define facilities that allow specification of the character encoding scheme used to encode binary source text in a conflicting manner. There are only two sources for character encoding scheme information: the charset parameter and the Unicode encoding form signature. If a charset parameter is specified, binary source text is processed as defined for that character encoding scheme.

5. Security Considerations

Refer to [RFC3552] for a discussion of terminology used in this section. Examples in this section and discussions of interactions of host environments with scripts and extensions to [ECMA] are to be understood as non-exhaustive and of a purely illustrative nature.

The programming language defined in [ECMA] is not intended to be computationally self-sufficient, rather it is expected that the computational environment provides facilities to programs to enable

specific functionality. Such facilities constitute unknown factors and are thus considered out of the scope of this document.

Derived programming languages are permitted to include additional functionality that is not described in [ECMA]; such functionality constitutes an unknown factor and is thus considered out of the scope of this document. In particular, extensions to [ECMA] defined for the JavaScript programming language are not discussed in this document.

Uncontrolled execution of scripts can be exceedingly dangerous. Implementations that execute scripts MUST give consideration to their application's threat models and those of the individual features they implement; in particular, they MUST ensure that untrusted content is not executed in an unprotected environment.

Specifications for host environment facilities and for derived programming languages should include security considerations. If an implementation supports such facilities, the respective security considerations apply. In particular, if scripts can be referenced from or included in specific document formats, the considerations for the embedding or referencing document format apply.

For example, scripts embedded in application/xhtml+xml [RFC3236] documents could be enabled through the host environment to manipulate the document instance, which could cause the retrieval of remote resources; security considerations regarding retrieval of remote resources of the embedding document would apply in this case.

This circumstance can further be used to make information, that is normally only available to the script, available to a web server by encoding the information in the resource identifier of the resource, which can further enable eavesdropping attacks. Implementation of such facilities is subject to the security considerations of the host environment, as discussed above.

The facilities defined in [ECMA] do not include provisions for input of external data, output of computed results, or modification of aspects of the host environment. An implementation of only the facilities defined in [ECMA] is not considered to support dangerous operations.

The programming language defined in [ECMA] does include facilities to loop, cause computationally complex operations, or consume large amounts of memory; this includes, but is not limited to, facilities that allow dynamically generated source text to be executed (e.g., the eval() function); uncontrolled execution of such features can cause denial of service, which implementations MUST protect against.

A host environment can provide facilities to access external input. Scripts that pass such input to the eval() function or similar language features can be vulnerable to code injection attacks. Scripts are expected to protect against such attacks.

A host environment can provide facilities to output computed results in a user-visible manner. For example, host environments supporting a graphical user interface can provide facilities that enable scripts to present certain messages to the user. Implementations MUST take steps to avoid confusion of the origin of such messages. In general, the security considerations for the host environment apply in such a case as discussed above.

Implementations are required to support the UTF-8 character encoding scheme; the security considerations of [RFC3629] apply. Additional character encoding schemes may be supported; support for such schemes is subject to the security considerations of those schemes.

Source text is expected to be in Unicode Normalization Form C. Scripts and implementations MUST consider security implications of unnormalized source text and data. For a detailed discussion of such implications refer to the security considerations in [RFC3629].

Scripts can be executed in an environment that is vulnerable to code injection attacks. For example, a CGI script [RFC3875] echoing user input could allow the inclusion of untrusted scripts that could be executed in an otherwise trusted environment. This threat scenario is subject to security considerations that are out of the scope of this document.

The "data" resource identifier scheme [RFC2397], in combination with the types defined in this document, could be used to cause execution of untrusted scripts through the inclusion of untrusted resource identifiers in otherwise trusted content. Security considerations of [RFC2397] apply.

Implementations can fail to implement a specific security model or other means to prevent possibly dangerous operations. Such failure could possibly be exploited to gain unauthorized access to a system or sensitive information; such failure constitutes an unknown factor and is thus considered out of the scope of this document.

6. IANA Considerations

This document registers four new media types as defined in the following sections.

7. JavaScript Media Types

7.1. text/javascript (obsolete)

Type name: text
Subtype name: javascript
Required parameters: none
Optional parameters: charset, see section 4.1.
Encoding considerations:
The same as the considerations in section 3.1 of [RFC3023].

Security considerations: See section 5.
Interoperability considerations:
None, except as noted in other sections of this document.

Published specification: [JS15]
Applications which use this media type:
Script interpreters as discussed in this document.

Additional information:

Magic number(s): n/a
File extension(s): .js
Macintosh File Type Code(s): TEXT

Person & email address to contact for further information:
See Author's Address section.

Intended usage: OBSOLETE
Restrictions on usage: n/a
Author: See Author's Address section.
Change controller: The IESG.

7.2. application/javascript

Type name: application
Subtype name: javascript
Required parameters: none
Optional parameters: charset, see section 4.1.
Encoding considerations:
The same as the considerations in section 3.2 of [RFC3023].

Security considerations: See section 5.
Interoperability considerations:
None, except as noted in other sections of this document.

Published specification: [JS15]
Applications which use this media type:
Script interpreters as discussed in this document.

Additional information:

Magic number(s): n/a
File extension(s): .js
Macintosh File Type Code(s): TEXT

Person & email address to contact for further information:
See Author's Address section.

Intended usage: COMMON
Restrictions on usage: n/a
Author: See Author's Address section.
Change controller: The IESG.

8. ECMAScript Media Types

8.1. text/ecmascript (obsolete)

Type name: text
Subtype name: ecmascript
Required parameters: none
Optional parameters: charset, see section 4.1.
Encoding considerations:
The same as the considerations in section 3.1 of [RFC3023].

Security considerations: See section 5.
Interoperability considerations:
None, except as noted in other sections of this document.

Published specification: [ECMA]
Applications which use this media type:
Script interpreters as discussed in this document.

Additional information:

Magic number(s): n/a
File extension(s): .es
Macintosh File Type Code(s): TEXT

Person & email address to contact for further information:
See Author's Address section.

Intended usage: OBSOLETE
Restrictions on usage: n/a
Author: See Author's Address section.
Change controller: The IESG.

8.2. application/ecmascript

Type name: application
Subtype name: ecmascript
Required parameters: none
Optional parameters: charset, see section 4.1.

Note: Section 3 defines error handling behavior for content labeled with a "version" parameter.

Encoding considerations:
The same as the considerations in section 3.2 of [RFC3023].

Security considerations: See section 5.
Interoperability considerations:
None, except as noted in other sections of this document.

Published specification: [ECMA]
Applications which use this media type:
Script interpreters as discussed in this document.

Additional information:

Magic number(s): n/a
File extension(s): .es
Macintosh File Type Code(s): TEXT

Person & email address to contact for further information:
See Author’s Address section.

Intended usage: COMMON
Restrictions on usage: n/a
Author: See Author’s Address section.
Change controller: The IESG.

9. References

9.1. Normative References

[CHARSETS] IANA, "Assigned character sets", <http://www.iana.org/assignments/character-sets>.

[ECMA] European Computer Manufacturers Association, "ECMAScript Language Specification 3rd Edition", December 1999, <http://www.ecma-international.org/publications/standards/Ecma-262.htm>

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, October 2000.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3536] Hoffman, P., "Terminology Used in Internationalization in the IETF", RFC 3536, May 2003.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

9.2. Informative References

[E4X] European Computer Manufacturers Association, "ECMAScript for XML (E4X)", June 2004, <http://www.ecma-international.org/publications/standards/Ecma-357.htm>

[EcmaCompact] European Computer Manufacturers Association, "ECMAScript 3rd Edition Compact Profile", June 2001, <http://www.ecma-international.org/publications/standards/Ecma-327.htm>

[JS15] Netscape Communications Corp., "Core JavaScript Reference 1.5", September 2000, <http://web.archive.org/*/http://devedge.netscape.com/library/manuals/2000/javascript/1.5/reference/>.

[RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, August 1998.

[RFC3236] Baker, M. and P. Stark, "The 'application/xhtml+xml' Media Type", RFC 3236, January 2002.

[RFC3875] Robinson, D. and K. Coar, "The Common Gateway Interface (CGI) Version 1.1", RFC 3875, October 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

Author's Address

Bjoern Hoehrmann
Weinheimer Strasse 22
Mannheim D-68309
Germany

EMail: bjoern@hoehrmann.de
URI: http://bjoern.hoehrmann.de

Note: Please write "Bjoern Hoehrmann" with o-umlaut (U+00F6) wherever possible, e.g., as "Björn Höhrmann" in HTML and XML.

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).