Internet Engineering Task Force (IETF)
Request for Comments: 9239
Obsoletes: 4329
Category: Informational
ISSN: 2070-1721

M. Miller

M. Borins
GitHub
M. Bynens
Google
B. Farias

May 2022

Updates to ECMAScript Media Types

Abstract

This document describes the registration of media types for the ECMAScript and JavaScript programming languages and conformance requirements for implementations of these types. This document obsoletes RFC 4329 ("Scripting Media Types)", replacing the previous registrations with information and requirements aligned with common usage and implementation experiences.

IESG Note

This document records the relationship between the work of Ecma International's Technical Committee 39 and the media types used to identify relevant payloads.

That relationship was developed outside of the IETF and as a result is unfortunately not aligned with the best practices of BCP 13. Consequently, consensus exists in the IETF to document the relationship and update the relevant IANA registrations for those media types, but this is not an IETF endorsement of the media types chosen for this work.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9239.

Table of Contents

1. Introduction

This memo describes media types for the JavaScript and ECMAScript programming languages. Refer to the sections "Introduction" and "Overview" in [ECMA-262] for background information on these languages. This document updates the descriptions and registrations for these media types to reflect existing usage on the Internet, and it provides up-to-date security considerations.

This document replaces the media type registrations in [RFC4329] and updates the requirements for implementations using those media types defined in [RFC4329] based on current existing practices. As a consequence, this document obsoletes [RFC4329].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Compatibility

This document defines equivalent processing requirements for the various script media types. The most widely supported media type in use is text/javascript; all others are considered historical and obsolete aliases of text/javascript.

The types defined in this document are applicable to scripts written in [ECMA-262]. New editions of [ECMA-262] are subjected to strong obligations of backward compatibility, imposed by the standardization process of Ecma International's Technical Committee 39 (TC39). As a result, JavaScript code based on an earlier edition is generally compatible with a JavaScript engine adhering to a later edition. The few exceptions to this are documented in [ECMA-262] in the section "Additions and Changes That Introduce Incompatibilities with Prior Editions". JavaScript developers commonly use feature detection to ensure that modern JavaScript features are only used when available in the current environment. Later editions of [ECMA-262] are not directly addressed in this document, although it is expected that implementations will behave as if applicability were extended to them. This document does not address other extensions to [ECMA-262] or scripts written in other languages.

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.

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.

Note that this use of the "text" media type tree willfully does not align with its original intent per [RFC2045]. The reason for this is historical. [RFC4329] registered both the text/* and application/* types, marking the text/* types obsolete. This was done to encourage people toward application/*, matching the guidance in [RFC4288], the predecessor to [RFC6838]. Since then, however, the industry widely adopted text/* anyway. The definitions in this document reflect the current state of implementation across the JavaScript ecosystem, in web browsers and other environments such as Node.js alike, in order to guarantee backward compatibility with existing applications as much as possible. Future registrations should not view this as a repeatable precedent.

3. Modules

In order to formalize support for modular programs, [ECMA-262] (starting with the 6th Edition) defines two top-level goal symbols (or roots to the abstract syntax tree) for the ECMAScript grammar: Module and Script. The Script goal represents the original structure where the code executes in the global scope, while the Module goal represents the module system built into ECMAScript starting with the 6th Edition. See the section "ECMAScript Language: Scripts and Modules" in [ECMA-262] for details.

This separation means that (in the absence of additional information) there are two possible interpretations for any given ECMAScript

source text.

Ecma International's Technical Committee 39 (TC39), the standards body in charge of ECMAScript, has determined that media types are outside of their scope of work [TC39-MIME-ISSUE].

It is not possible to fully determine if a source text of ECMAScript is meant to be parsed using the Module or Script grammar goals based upon content or media type alone. Therefore, as permitted by the media types in this document, scripting environments use out-of-band information in order to determine what goal should be used. Some scripting environments have chosen to adopt the file extension of .mjs for this purpose.

4. Encoding

Refer to [RFC6365] for a discussion of terminology used in this section. Source text (as defined in the section "Source Text" in [ECMA-262]) 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 is 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.

In order to ensure interoperability and align with widespread implementation practices, the charset parameter is optional rather than required, despite the recommendation in BCP 13 [RFC6838] for text/* types.

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 defined by this document.

Implementations that support binary source text MUST support binary source text encoded using the UTF-8 [RFC3629] character encoding scheme. Module goal sources MUST be encoded as UTF-8; all other encodings will fail. Source goal sources SHOULD be encoded as UTF-8; other character encoding schemes MAY be supported but are discouraged. Whether U+FEFF is processed as a Byte Order Mark (BOM) signature or not depends on the host environment and is not defined by this document.

4.1. Charset Parameter

The charset parameter provides a means to specify the character encoding scheme of binary source text. If present, the value of the charset parameter MUST be a registered charset [CHARSETS] and is considered valid if it matches the mime-charset production defined in Section 2.3 of [RFC2978].

The charset parameter is only used when processing a Script goal source; Module goal sources MUST always be processed as UTF-8.

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 the binary source text is not already determined to be using a Module goal and 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
EF BB BF	UTF-8
FF FE	UTF-16LE
FE FF	UTF-16BE

Table 1

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.

2. Else, if a charset parameter is specified and its value is valid and supported by the implementation, the value determines the character encoding scheme.

3. Else, the character encoding scheme is assumed to be UTF-8.

If the character encoding scheme is determined to be UTF-8 through any means other than step 1 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.

4.3. Character Encoding Scheme Error Handling

Binary source text that is not properly encoded for the determined character encoding can pose a security risk, as discussed in Section 5. That said, because of the varied and complex environments scripts are executed in, most of the error handling specifics are left to the processors. The following are broad guidelines that processors follow.

If binary source text is determined to have been encoded using a certain character encoding scheme that the implementation is unable to process, implementations can consider the resource unsupported (i.e., do 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 valid according to that scheme. Implementations can substitute those invalid sequences with the replacement character U+FFFD (properly encoded for the scheme) or stop processing altogether.

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, modules, and extensions to [ECMA-262] are to be understood as non-exhaustive and of a purely illustrative nature.

The programming language defined in [ECMA-262] 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 not defined by this document.

Derived programming languages are permitted to include additional functionality that is not described in [ECMA-262]; such functionality constitutes an unknown factor and is thus not defined by this document. In particular, extensions to [ECMA-262] 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.

Module scripts in ECMAScript can request the fetching and processing of additional scripts; this is called "importing". Implementations that support modules need to process imported sources in the same way as scripts. See the section "ECMAScript Language: Scripts and Modules" in [ECMA-262] for details. Further, there may be additional privacy and security concerns, depending on the location(s) the original script and its imported modules are obtained from. For instance, a script obtained from "host-a.example" could request to import a script from "host-b.example", which could expose information about the executing environment (e.g., IP address) to "host-

b.example".

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 also 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 programming language defined in [ECMA-262] 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.

With the addition of SharedArrayBuffer objects in ECMAScript version 8, it could be possible to implement a high-resolution timer, which could lead to certain types of timing and side-channel attacks (e.g., [SPECTRE]). Implementations can take steps to mitigate this concern, such as disabling or removing support for SharedArrayBuffer objects, or can take additional steps to ensure that this shared memory is only accessible between execution contexts that have some form of mutual trust.

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 Common Gateway Interface (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 not defined by this document.

6. IANA Considerations

The media type registrations herein are divided into two major categories: (1) the sole media type "text/javascript", which is now in common usage and (2) all of the media types that are obsolete (i.e., "application/ecmascript", "application/javascript", "application/x-ecmascript", "application/x-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", and "text/x-ecmascript").

For both categories, the "Published specification" entry for the media types is updated to reference [ECMA-262]. In addition, a new file extension of .mjs has been added to the list of file extensions with the restriction that contents should be parsed using the Module goal. Finally, the [HTML] specification uses "text/javascript" as the default media type of ECMAScript when preparing script tags; therefore, "text/javascript" intended usage has been moved from OBSOLETE to COMMON.

These changes have been reflected in the IANA "Media Types" registry in accordance with [RFC6838]. All registrations will point to this document as the reference. The outdated note stating that the "text/javascript" media type has been "OBSOLETED in favor of application/javascript" has been removed. The outdated note stating that the "text/ecmascript" media type has been "OBSOLETED in favor of application/ecmascript" has been removed. IANA has added the note "OBSOLETED in favor of text/javascript" to all registrations except "text/javascript"; that is, this note has been added to the "text/ecmascript", "application/javascript", and "application/ecmascript"

registrations.

Four of the legacy media types in this document have a subtype starting with the "x-" prefix:

* application/x-ecmascript

* application/x-javascript

* text/x-ecmascript

* text/x-javascript

Note that these are grandfathered media types registered as per Appendix A of [RFC6838]. These registrations predate BCP 178 [RFC6648], which they violate, and are only included in this document for backward compatibility.

6.1. Common JavaScript Media Types

6.1.1. text/javascript

Type name: text

Subtype name: javascript

Required parameters: N/A

Optional parameters: charset. See Section 4.1 of RFC 9239.

Encoding considerations: Binary

Security considerations: See Section 5 of RFC 9239.

Interoperability considerations: It is expected that implementations will behave as if this registration applies to later editions of [ECMA-262], and its published specification references may be updated accordingly from time to time. Although this expectation is unusual among media type registrations, it matches widespread industry conventions. See Section 2 of RFC 9239.

Published specification: [ECMA-262]

Applications that use this media type: Script interpreters as discussed in RFC 9239.

Additional information:
Deprecated alias names for this type: application/javascript, application/x-javascript, text/javascript1.0, text/javascript1.1, text/javascript1.2, text/javascript1.3, text/javascript1.4, text/javascript1.5, text/jscript, text/livescript
Magic number(s): N/A
File extension(s): .js, .mjs
Macintosh File Type Code(s): TEXT

Person & email address to contact for further information: See the
   Authors' Addresses sections of RFC 9239 and [RFC4329].

Intended usage: COMMON

Restrictions on usage: The .mjs file extension signals that the file
   represents a JavaScript module. Execution environments that rely
   on file extensions to determine how to process inputs parse .mjs
   files using the Module grammar of [ECMA-262].

Author: See the Authors' Addresses sections of RFC 9239 and
   [RFC4329].

Change controller: IESG <iesg@ietf.org>

6.2. Historic JavaScript Media Types

The following media types and legacy aliases are added or updated for
historical purposes. All herein have an intended usage of OBSOLETE
and are not expected to be in use with modern implementations.

6.2.1. text/ecmascript

Type name: text

Subtype name: ecmascript

Required parameters: N/A

Optional parameters: charset. See Section 4.1 of RFC 9239.

Encoding considerations: Binary

Security considerations: See Section 5 of RFC 9239.

Interoperability considerations: It is expected that implementations
   will behave as if this registration applies to later editions of
   [ECMA-262], and its published specification references may be
   updated accordingly from time to time. Although this expectation
   is unusual among media type registrations, it matches widespread
   industry conventions. See Section 2 of RFC 9239.

Published specification: [ECMA-262]

Applications that use this media type: Script interpreters as
   discussed in RFC 9239.

Additional information:
   Deprecated alias names for this type: application/ecmascript,
      application/x-ecmascript, text/x-ecmascript
   Magic number(s): N/A
   File extension(s): .es, .mjs
   Macintosh File Type Code(s): TEXT

Person & email address to contact for further information: See the
   Authors' Addresses sections of RFC 9239 and [RFC4329].

Intended usage: OBSOLETE

Restrictions on usage: This media type is obsolete; current implementations should use text/javascript as the only JavaScript/ECMAScript media type. The .mjs file extension signals that the file represents a JavaScript module. Execution environments that rely on file extensions to determine how to process inputs parse .mjs files using the Module grammar of [ECMA-262].

Author: See the Authors' Addresses sections of RFC 9239 and [RFC4329].

Change controller: IESG <iesg@ietf.org>

7. References

7.1. Normative References

[CHARSETS] IANA, "Character Sets", <https://www.iana.org/assignments/character-sets>.

[ECMA-262] Ecma International, "ECMA-262 12th Edition, June 2021. ECMAScript 2021 language specification", June 2021, <https://262.ecma-international.org/12.0/>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, DOI 10.17487/RFC2397, August 1998, <https://www.rfc-editor.org/info/rfc2397>.

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, DOI 10.17487/RFC2978, October 2000, <https://www.rfc-editor.org/info/rfc2978>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", RFC 4288, DOI 10.17487/RFC4288, December 2005, <https://www.rfc-editor.org/info/rfc4288>.

[RFC4329] Hoehrmann, B., "Scripting Media Types", RFC 4329, DOI 10.17487/RFC4329, April 2006, <https://www.rfc-editor.org/info/rfc4329>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <https://www.rfc-editor.org/info/rfc6365>.

[RFC6648] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, DOI 10.17487/RFC6648, June 2012, <https://www.rfc-editor.org/info/rfc6648>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. Informative References

[HTML] WHATWG, "HTML Living Standard", May 2022, <https://html.spec.whatwg.org/multipage/scripting.html#prepare-a-script>.

[RFC3236] Baker, M. and P. Stark, "The 'application/xhtml+xml' Media Type", RFC 3236, DOI 10.17487/RFC3236, January 2002, <https://www.rfc-editor.org/info/rfc3236>.

[RFC3875] Robinson, D. and K. Coar, "The Common Gateway Interface (CGI) Version 1.1", RFC 3875, DOI 10.17487/RFC3875, October 2004, <https://www.rfc-editor.org/info/rfc3875>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/info/rfc3987>.

[SPECTRE] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution", DOI 10.48550/arXiv.1801.01203, January 2018, <https://arxiv.org/abs/1801.01203>.

[TC39-MIME-ISSUE] TC39, "Add 'application/javascript+module' mime to remove

ambiguity", Wayback Machine archive, August 2017, <https://web.archive.org/web/20170814193912/https://github.com/tc39/ecma262/issues/322>.

Appendix A. Changes from RFC 4329

* Added a section discussing ECMAScript modules and the impact on processing.

* Updated the Security Considerations section to discuss concerns associated with ECMAScript modules and SharedArrayBuffers.

* Updated the character encoding scheme detection to remove normative guidance on its use, to better reflect operational reality.

* Changed the intended usage of the media type "text/javascript" from OBSOLETE to COMMON.

* Changed the intended usage for all other script media types to obsolete.

* Updated various references where the original has been obsoleted.

* Updated references to ECMA-262 to match the version at the time of publication.

Acknowledgements

This work builds upon its antecedent document, authored by Björn Höhrmann. The authors would like to thank Adam Roach, Alexey Melnikov, Allen Wirfs-Brock, Anne van Kesteren, Ben Campbell, Benjamin Kaduk, Éric Vyncke, Francesca Palombini, James Snell, Kirsty Paine, Mark Nottingham, Murray Kucherawy, Ned Freed, Robert Sparks, and Suresh Krishnan for their guidance and feedback throughout this process.

Authors' Addresses

Matthew A. Miller
Email: linuxwolf+ietf@outer-planes.net

Myles Borins
GitHub
Email: mylesborins@github.com

Mathias Bynens
Google
Email: mths@google.com

Bradley Farias
Email: bradley.meck@gmail.com