The Lynx Media Type

Introduction

This document defines a new hypertext media type designed to represent and describe documents and the connections among them, with enough semantic detail to allow the development of client applications without imposing strict display constraints or unnecessary coupling between clients and servers. By reducing display constraints and coupling, developers have more freedom to create client applications that take better advantage of the unique characteristics of each platform and device.

Conventions Used in This Document

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 RFC 2119.

Terminology

JSON
: Refers to a the media type JSON RFC 4627.

URI
: Refers to a URI as defined in RFC 3986.

URL
: Refers to a URL as defined in RFC 3986.

URN
: Refers to a URN as defined in RFC 3986.

Same-Document Reference
: Refers to a same-document reference as defined in section 4.4 of RFC 3986.

Media Type Name
: Refers to a media type name as defined in RFC 6838.

Base Hint
: Refers to one of the following hints: `object`, `link`, `form`, `submit`, `array`, `media`, and `text`.

Display
: The conveyance of a value to a user in a manner appropriate for the user agent and the user.

Resource
: See Section 5.2.1.1 "Resources and Resource Identifiers", [Fielding Dissertation])

Representation
: See Section 5.2.1.2 "Representations", [Fielding Dissertation])

Compliance

An implementation is "non-compliant" if it fails to satisfy one or more of the MUST or REQUIRED level requirements. An implementation that satisfies all of the MUST or REQUIRED and all of the SHOULD level requirements is said to be "unconditionally compliant"; an implementation that satisfies all of the MUST level requirements but not all of the SHOULD level requirements is said to be "conditionally compliant."

Registration Status

The registration status of this media type is as follows:

- application/lynx+json (unregistered)

Parameters

content

The `content` parameter indicates whether the body of a message contains a value or a specification.

- The parameter is OPTIONAL. The default value is `value`.
- If present, the value MUST be either `value` or `spec`.

Example

```
GET /spec/movie HTTP/1.1
Accept: application/lynx+json;content=spec
```

spec

The `spec` parameter describes the value in the body of a message.

- The parameter is OPTIONAL.
- If present, its value MUST be either a specification JSON object or a specification URL.
- If the value of the `content` parameter is `spec`, then this parameter MUST NOT be present.

Examples

```
HTTP/1.1 200 OK
Content-Type: application/lynx+json;spec="{ "hints": ["text"] }"

"Hello, World!"
```

```
HTTP/1.1 200 OK
Content-Type: application/lynx+json;spec="http://example.com/specs

"Hello, World!"
```

baseURI

- The parameter is OPTIONAL.
- If present, its value MUST be an absolute URI to be used for relative URI resolution. For more information, see [Section 5.1.2 "Base URI from the Encapsulating Entity" in RFC 3986](#).

Example

```
HTTP/1.1 200 OK
Content-Type: application/lynx+json;baseURI="http://example.com/";

"Hello, World!"
```

realm

- The parameter is OPTIONAL.
- If present, its value MUST comply with the rules defined for Realm URI.

Example

```
HTTP/1.1 200 OK
Content-Type: application/lynx+json;spec="./specs/array-of-text";r

[ "Hello, World!" ]
```

File Extensions

- `.lnx` => `application/lynx+json`
- `.lnxs` => `application/lynx+json;content=spec`

Content

The content of all documents defined by this media type are extensions of JSON and MUST comply with the rules of that media type. All usages of the JSON keywords "string", "number", "object", "array", "true", "false", "null", and "value" refer to the concepts in the JSON specification unless otherwise stated. The word "property" is used to refer to the concept of "pair", "name/value pair", or "member" in the JSON specification. The word "integer" is used to refer to a JSON "number" with no fractional part. The phrase "simple value" is used to refer to strings, numbers, `true`, `false`, or `null`.

User Agent Rules

If the user agent encounters a relative URI it MUST resolve an absolute URI using the process defined in Section 5 "Reference Resolution" in RFC 3986.

Objects

This section extends a JSON object to provide metadata so that a user agent may effectively display the object to a user.

Format Rules

- MUST be described by a specification, either by its own `spec` property, by the `children` property of its container's specification, or by a `spec` media type parameter.
- If `spec` is present, the value MUST be a specification or a specification URL.

- If the object is a document, it MAY have a `baseURI` property. If present, the value MUST be an absolute URI to be used for relative URI resolution. For more information, see Section 5.1.1 "Base URI Embedded in Content" in RFC 3986.
- If the object is a document, it MAY have a `realm` property. If present, the value MUST comply with the rules defined for realm URI.

Examples

Object

```
{
  "title": "Groundhog Day",
  "spec": "http://www.example.com/specs/movie"
}
```

Document

```
{
  "title": "What About Bob?",
  "baseURI": "http://www.example.com",
  "realm": "http://www.example.com/movies",
  "spec": "http://www.example.com/specs/movie"
}
```

User Agent Considerations

The user agent must anticipate that an object may contain additional properties (a title, text, objects, arrays, links, etc.).

Other Values

Format Rules

MUST be described by a specification, either by the `children` property of its container's specification or by a `spec` media type parameter.

URI Fragments

URI fragments are used to locate values in a document by property name.

Example

```
{
  "linkToCast": {
    "title": "Cast"
    "href": "#cast"
  },
  "cast": [
    "Robin Williams",
    "Matt Damon",
    "Ben Affleck"
  ],
  "spec": "http://www.example.com/specs/movie"
}
```

User Agent Rules

To locate a value using a URI fragment, the user agent MUST search the document for a property with a name matching the fragment. Using depth-first traversal, the value of the first property whose name matches the fragment is the result. Depth-first traversal is performed according to the the order of the specifications in the `children` property and, if a value is an array, the order of the items in the array.

Authoring Consideration

A URI fragment referencing a property that is not described by a specification will not be located.

Specifications

A specification describes a JSON value.

Format Rules

Specification

MAY have any number of properties. This document defines some properties (e.g. `hints`, `children`, etc.). Authors may also define extended properties.

Specification URL

A specification URL is used to refer to a remote specification. The value MUST be an absolute or relative URL and MUST NOT contain a URI fragment.

Examples

Specification Object

```
{
  "title": "Fletch",
  "spec": {
    "hints": [ "object" ],
    "children": [
      {
        "name": "title",
        "hints": [ "title", "text" ]
        }
      ]
    }
}
```

Specification URL

```
{
  "title": "Fletch",
  "spec": "http://www.example.com/specs/object"
}
```

User Agent Rules

If the `spec` property value is a specification URL, then the user agent MUST fetch the remote specification using the default retrieval action for the protocol (e.g. GET for HTTP). The media type to be used for retrieval is `application/lynx+json;content=spec`.

User Agent Considerations

A user agent should ignore any specification property that it does not understand.

Hints

The `hints` property of a specification describes the meaning of a value.

Format Rules

The `hints` property is REQUIRED when referring to a value that does not have its own `spec` property. Otherwise, the `hints` property is OPTIONAL. If present, it MUST comply with the rules of a hint array.

Hint Object

- MUST have a `name` property to identify the hint.
- MAY have a `documentation` property. If present, its value MUST be a URL defining the location of human-readable documentation for the hint.

Hint Array

- MUST be ordered from more specific to less specific.
- MAY contain a string instead of a hint object. If a string value is present, it is assumed to be a hint object with a name equal to the string value. For example, a hint of `"text"` is equivalent of `{ "name": "text" }`.
- SHOULD end with a base hint.

Example

```
{
  "name": "Irwin M. Fletcher",
  "spec":{
    "hints": [ "object" ],
    "children": [
      {
        "name": "name",
        "hints": [
          {
            "name": "com:example:character-name",
            "documentation": "http://www.example.com/hints/charact
          },
          "text"
        ]
      }
    ]
  }
}
```

User Agent Rules

- The user agent MUST process hints in order from more specific to less specific. The user agent MUST find the most specific hint that it understands and process the value based on that understanding.
- If the user agent does not understand any hints it SHOULD ignore the value.
- SHOULD understand all of the base hints.

Authoring Considerations

This content pertains to using existing hints. For rules about authoring hints see extending hints.

Object

An `object` hint describes a value that should be interpreted as an object.

Format Rules

MUST comply with the rules defined for all objects.

Example

```
{
  "title": "Groundhog Day",
  "spec": {
    "hints": [ "object" ],
    "children": [
        {
            "name": "title",
            "hints": [ "title", "text" ]
        }
      ]
    }
}
```

User Agent Rules

MUST comply with the rules defined for all objects.

Link

A `link` hint describes a value that should be interpreted as a hyperlink.

Format Rules

- MUST comply with the rules defined for all objects.
- MUST have an `href` property whose value is a valid URI.

Example

```
{
  "title": "Fletch",
  "href": "http://www.example.com/m/fletch",
  "spec": {
    "hints": [ "link" ],
    "children": [
        {
            "name": "title",
            "hints": [ "title", "text" ]
        }
    ]
  }
}
```

User Agent Rules

- MUST comply with the rules defined for all objects.
- The user agent MUST provide the user with a control to interact with the link in order to follow it.
- If the user follows the link, then the user agent MUST fetch the target identified by the `href` using the default retrieval action for the protocol (e.g. GET for HTTP).
- The user agent SHOULD display or make the `href` accessible to the user so he/she can consider its value prior to following the link.

User Agent Considerations

- The user agent must anticipate that a link may contain additional properties (a title, text, objects, arrays, links, etc.).

- The user agent must anticipate that a link may contain no visible content.

Authoring Considerations

While the concept of a link relationship ("rel") has been intentionally omitted from these rules, that does not mean that it has no value or that it should be omitted from authored documents. The purpose of this specification is to define the meaning of the values - not to define the relationship between the value and the resource identified by the context URI. The "rel" concept is still important and it could be included in a link object by extension.

Form

A `form` hint describes a value that should be interpreted as a form.

Format Rules

MUST comply with the rules defined for all objects.

Example

```
{
  "title": "A simple form",
  "exampleInput": null,
  "spec": {
    "hints": [ "form" ],
    "children": [
      {
        "name": "title",
        "hints": [ "title", "text" ]
      },
      {
        "name": "exampleInput",
        "hints": [ "text" ],
        "input": true
      }
    ]
  }
}
```

User Agent Rules

MUST comply with the rules defined for all objects.

Authoring Considerations

A form should contain at least one submit object.

Submit

A `submit` hint describes a value that defines where and how to submit form data.

Format Rules

- MUST comply with the rules defined for all objects.
- MUST have an `action` property whose value must be a valid URI.
- MAY have a `method` property indicating the protocol method used to submit the form data.
- MAY have an `enctype` property. If not present, the default value is `application/x-www-form-urlencoded`.

Example

```
{
  "title": "Submit Review",
  "action": "http://www.example.com/m/fletch/reviews",
  "method": "POST",
  "spec": {
    "hints": [ "submit" ],
    "children": [
        {
            "name": "title",
            "hints": [ "title", "text" ]
        }
    ]
  }
}
```

User Agent Rules

- MUST comply with the rules defined for all objects.
- The user agent MUST provide the user with a control to perform the submission.
- If the `method` property is not present, then the user agent MUST submit the form using the default retrieval action for the protocol (e.g. GET for HTTP).
- When a `submit` control is invoked, the user agent MUST prepare a form data set, encode the form data set into a representation, and submit the representation, as defined in Submitting Form Data.

User Agent Considerations

- The user agent must anticipate that a `submit` object may contain no visible content.

Array

An `array` hint describes a value that should be interpreted as an array.

Format Rules

The value MAY be `undefined`, `null`, `[]`, or a non-empty array.

Input

If the specification having an `array` hint has a `children` property value with an `input` property, then the array value is considered to be an array of inputs.

Example

Non-Input

```
{
  "title": "Top Quotes",
  "quotes": [
    "Print this!",
    "Act like you don't give a crap and you fit right in.",
    "You know if you shoot me, you're liable to lose all of those
  ],
  "spec": {
    "hints": [ "object" ],
    "children": [
      {
        "name": "title",
        "hints": [ "title", "text" ]
      },
      {
        "name": "quotes",
        "hints": [ "array" ],
        "children": {
          "hints": [ "text" ]
        }
      }
    ]
  }
}
```

Input

```
{
  "characters": [
    "Ty Webb",
    "Al Czervik",
    "Carl Spackler"
  ],
  "spec": {
    "hints": [ "form" ],
    "children": [
      {
        "name": "characters",
        "hints": [ "array" ],
        "children": {
            "hints": [ "text" ],
            "input": true
        }
```

```
            }
        ]
    }
}
```

User Agent Rules

If the array value is considered to be an array of inputs, the user agent MUST provide the user with controls to add values to the array and remove values from the array.

User Agent Considerations

- The user agent must anticipate that an array may contain simple content (text), complex content (objects, arrays, links, etc.), or a mixture of both.
- A user agent must anticipate that a value described by an `array` hint may contain any length and combination of values, including repeated values.

Content

A `content` hint describes a value that contains or references content that is to be considered a part of the containing representation (embedded).

Format Rules

- MUST comply with the rules defined for all objects.
- MUST have either an `src` property for remote content or a `data` property for inline content.
- SHOULD have an `alt` property whose value specifies alternate text to be displayed if the content cannot be displayed or if the user cannot view it.
- MAY have a `scope` property whose value specifies the content realm intended for display. If present, the value MUST comply with the rules defined for realm URI.

`src` present

- The `src` property value MUST be a valid URI.
- MUST NOT have a `data`, `type`, or `encoding` property.

`data` present

- The `data` property value MUST be a string representing the content to be embedded.
- MUST have a `type` property to indicate the media type of the content. Its value must be a string that represents a valid media type name.
- MAY have an `encoding` property whose value MUST be either `utf-8` or `base64`. If not present, the value is `utf-8`.
- MUST NOT have an `src` property.

Examples

`src` present

```
{
  "title": "Fletch",
  "src": "http://www.example.com/fletch-movie-review.pdf",
  "alt": "Movie Review",
  "spec": {
    "hints": [ "content" ],
    "children": [
        {
            "name": "title",
            "hints": [ "title", "text" ]
        }
    ]
  }
}
```

`data` present

```
{
  "title": "Fletch",
  "data": "# Review of Fletch\n##Pros\n\nToo many to list.\n##Cons
  "type": "text/markdown",
  "alt": "Movie Review",
  "spec": {
    "hints": [ "content" ],
    "children": [
        {
            "name": "title",
            "hints": [ "title", "text" ]
        }
    ]
  }
}
```

User Agent Rules

- MUST comply with the rules defined for all objects.
- If the user agent is unable to display the content, it MUST display a link to the content, with the text of the `alt` property if it exists.

Input

- The user agent MUST provide the user with a control to provide the content.

User Agent Considerations

- The user agent must anticipate that a `content` object may contain additional properties (a title, text, objects, arrays, links, etc.).
- The user agent must anticipate that a `content` object may contain no visible content.

Image

An `image` hint describes a value that should be interpreted as an embedded image.

Format Rules

- MUST comply with the rules defined for conent.
- MUST precede the `content` hint.
- MAY have a `height` property whose value MUST be a number representing the natural height of the image in pixels.
- MAY have a `width` property whose value MUST be a number representing the natural width of the image in pixels.

Example

```
{
  "title": "Bill Murray",
  "src": "http://www.fillmurray.com/g/300/400",
  "alt": "Picture of Bill Murray",
  "height": 400,
  "width": 300,
  "spec": {
    "hints": [ "image", "content" ],
    "children": [
        {
            "name": "title",
            "hints": [ "title", "text" ]
        }
    ]
  }
}
```

User Agent Rules

MUST comply with the rules defined for content.

Text

A `text` hint describes a value that should be interpreted as text.

Format Rules

The value MAY be `undefined` or a simple value.

Example

```
{
  "lastName": "Cronauer",
  "spec": {
    "hints": [ "object" ],
    "children": [
      {
        "name": "lastName",
        "hints": [ "text" ]
      }
    ]
  }
}
```

User Agent Considerations

A user agent must anticipate that a value described by a `text` hint may contain any length and combination of characters, including line breaks.

Title

A `title` hint describes a value that that represents a distinguishing name for something.

Format Rules

- MUST comply with the rules defined for text.
- MUST precede the `text` hint.

Example

```
{
  "title": "Fletch",
  "spec": {
    "hints": [ "object" ],
    "children": [
      {
        "name": "title",
        "hints": [ "title", "text" ]
      }
    ]
  }
}
```

User Agent Considerations

- MUST comply with the rules defined for text.
- A user agent may display a title more prominently or in a different order than other text.

Children

The `children` property of a specification describes the properties of an object or the items in an array.

Format Rules

Properties of an Object

- If present, MUST be an array.
- Each item MUST be a specification or a specification URL.
- Each item MUST include a `name` property.

Items in an Array

If present, MUST be a specification or a specification URL.

Examples

Properties of an Object

```
{
  "rating": "PG",
  "spec": {
    "hints": [ "form" ],
    "children": [
      {
        "name": "rating",
        "input": true,
        "hints": [ "urn:com:example:mpaa-rating", "text" ]
      }
    ]
  }
}
```

Items in an Array

```
{
  "quotes": [],
  "spec": {
    "children": [
      {
        "name": "quotes",
        "hints": [ "array" ],
        "children": {
          "hints": [ "text" ]
        }
      }
    ]
  }
}
```

User Agent Rules

If an object contains a `spec` property, the user agent MUST use that specification instead of the specification defined in `children`.

Name

The `name` property of a specification identifies the property value that the specification describes.

Format Rules

- If the specification describes a property value then `name` is REQUIRED.
- If present, the value must be the name of a property (which MAY be `undefined`).

Example

```
{
  "actor": "Chevy Chase",
  "spec": {
    "hints": [ "object" ],
    "children": [
      {
        "name": "actor",
        "hints": [ "text" ]
      }
    ]
  }
}
```

Visibility

The `visibility` property of a specification describes how a value should or should not be displayed to the user.

Format Rules

- The `visibility` property is OPTIONAL. The default value is `visible`.
- If present, the value MUST be one of the following: `visible`, `hidden`, or `concealed`.

Example

```
{
  "title": "Movie Trivia: What was Fletch's First Name?",
  "answer": "Irwin",
  "spec": {
    "hints": [ "object" ],
    "children": [
      {
        "name": "title",
        "hints": [ "title", "text" ]
      },
      {
        "name": "answer",
        "hints": [ "text" ]
      }
    ],
    "visibility": "concealed"
  }
}
```

User Agent Rules

- If `hidden`, the user agent MUST NOT display the value to the user.
- If `visible`, the user agent MUST display the value to the user.
- If `concealed`, the user agent SHOULD conceal the value. The user agent SHOULD provide a way to reveal the value.
- If the user agent does not understand the value of the `visibility` property, it MUST consider the value `visible`.

User Agent Considerations

The user agent must anticipate that the value may be input by the user. If `visibility` is `concealed`, the user agent should conceal the user's input.

Input

The `input` property of a specification describes how a value should be used to collect input from the user.

Format Rules

The `input` property is OPTIONAL.

- MAY be `true` or MAY be an object.
- If the value is an object, it MAY have a `name` property.

Example

```
{
  "rating": "PG",
  "spec": {
    "hints": [ "form" ],
    "children": [
      {
        "name": "rating",
        "input": true,
        "hints": [ "urn:com:example:mpaa-rating", "text" ]
      }
    ]
  }
}
```

Follow

The `follow` property of a specification describes a link that the user agent should follow on behalf of the user.

Format Rules

- The `follow` property is OPTIONAL.
- If present, the value MUST be a number indicating the number of milliseconds the user agent must wait before following the link.

Example

```
{
  "title": "Preparing movie for viewing. Please wait while we fini
  "href": "http://www.example.com/movies/fletch/view",
  "spec": {
    "hints": [ "link" ],
    "children": [
      {
        "name": "title",
        "hints": [ "title", "text" ]
      }
    ],
    "follow": 5000
  }
}
```

User Agent Rules

After displaying the document, the user agent MUST wait the specified number of milliseconds and then MUST follow the link.

Validation

The `validation` property of a specification describes the validity of form input values.

Format Rules

If the `input` property is present, the `validation` property is OPTIONAL. Otherwise, the `validation` property MUST NOT be present.

Example

```
{
  "name": "actor",
  "input": true,
  "validation": {}
}
```

User Agent Rules

- The user agent SHOULD ensure that validity has been evaluated for all inputs before submitting the form.
- If any inputs are invalid, the user agent MUST take the action prescribed by the validation property.
- The user agent MUST allow a user to submit a form even if inputs are invalid.

Required

The `required` property indicates that an input value is required.

Format Rules

- The `required` property is OPTIONAL.
- If present, it MUST have a `message` property whose value MUST be a same-document reference to the content to display when the input value fails validation.

Example

```
{
  "name": "actor",
  "input": true,
  "validation": {
    "required": {
      "message": "#actorRequired"
    }
  }
}
```

User Agent Rules

The user agent MAY ignore the `required` property, otherwise it MUST comply with the following rules:

- If the input value is `null`, `undefined`, `""`, or `[]`, then the input fails validation.
- If the input fails validation, then the user agent MUST set the `visibility` of the content identified by the `message` property to `visible`.

Text

The `text` property describes the validity of a text input.

Format Rules

The `text` property is OPTIONAL. If present, it MUST be an an object or array of objects matching the following rules:

- MUST have a `message` property whose value MUST be a same-document reference to the content to display when the input value fails validation.
- MAY have a `minLength` property. If present, the value of the property MUST be a nonnegative integer.
- MAY have a `maxLength` property. If present, the value of the property MUST be a nonnegative integer.
- MAY have a `pattern` property. If present, the value of the property MUST be a string representing a valid JavaScript regular expression as defined in ECMA 262, Section 15.10.

Examples

Text Validation Object

```
{
  "name": "actor",
  "input": true,
  "validation":{
    "text": {
      "message": "#actorPatternError",
      "pattern": "Chevy Chase|Bill Murray"
    }
  }
}
```

Text Validation Array

```
{
  "name": "actor",
  "input": true,
  "validation":{
    "text": [
      {
        "message": "#actorMinLengthError",
        "minLength": 2
      },
      {
        "message": "#actorPatternError",
        "pattern": "Chevy Chase|Bill Murray"
      }
    ]
  }
}
```

User Agent Rules

The user agent MAY ignore the `text` property, otherwise it MUST comply with the following rules:

- If the input value is `null`, `undefined`, or `""`, then the input passes validation.
- If the `minLength` property is present and its value is greater than the length of the input value, then the input fails validation.
- If the `maxLength` property is present and its value is less than the length of the input value, then the input fails validation.
- If the `pattern` property is present, the user agent MUST compile the regular expression specified in its specified value with the global, ignoreCase, and multiline options disabled and match against the full input value (If the pattern does not start with '^', it is implied. If the pattern does not end with '$', it is implied). If the input value does not match the regular expression, then the input fails validation.
- If the input fails validation, then the user agent MUST set the `visibility` of the content identified by the `message` property to `visible`.

Number

The `number` property describes the validity of a number input.

Format Rules

The `number` property is OPTIONAL. If present, it MUST be an an object or array of objects matching the following rules:

- If present, it MUST have a `message` property whose value MUST be a same-document reference to the content to display when the input value fails validation.
- MAY have a `min` property. If present, the value MUST be a number.
- MAY have a `max` property. If present, the value MUST be a number.
- MAY have a `step` property. If present, the value MUST be a positive number.

Examples

Number Validation Object

```
{
  "name": "fletchViewings",
  "input": true,
  "validation":{
    "number": {
      "message": "#fletchViewingsError",
      "min": 10
    }
  }
}
```

Number Validation Array

```
{
  "name": "fletchRating",
  "input": true,
  "validation":{
    "number": [
      {
        "message": "#fletchRatingTooLowError",
        "min": 95
      },
      {
        "message": "#fletchRatingTooHighError",
        "max": 100
      },
      {
        "message": "#fletchRatingStepError",
        "step": 1
      }
    ]
  }
}
```

User Agent Rules

The user agent MAY ignore the `number` property, otherwise it MUST comply with the following rules:

- If the input value is `null`, `undefined`, or `""`, then the input passes validation.
- If the input value is not numeric, then it fails validation.
- If the `min` property is present and its value is greater than the input value, then the input fails validation.
- If the `max` property is present and its value is less than the input value, then the input fails validation.
- If the `step` property is present and the input value is not evenly divisible by the `step` value, then the input fails validation.
- If the input fails validation, then the user agent MUST set the `visibility` of the content identified by the `message` property to `visible`.

Content

The `content` property describes the validity of the content of a `content` input.

Format Rules

The `content` property is OPTIONAL. If present, it MUST be an an object, or an array of objects, matching the following rules:

- MUST have a `message` property whose value MUST be a same-document reference to the content to display when the input value fails validation.
- MAY have a `type` property. If present, the value of the property MUST be a string or an array of strings matching one of the following: -- A file extension (e.g., `.jpg`, `.png`, `.doc`). -- A media type range (e.g., `image/png`, `image/*`).
- MAY have a `maxLength` property representing the maximum length of the content in

bytes. If present, the value of the property MUST be a nonnegative integer.

Example

```
{
  "name": "movie",
  "hints": [ "content" ],
  "input": true,
  "validation": {
    "content": {
      "message": "#movieTypeAndLengthMessage",
      "type": "video/*",
      "maxLength": 4000000
    }
  }
}
```

User Agent Rules

The user agent MAY ignore the `content` property, otherwise it MUST comply with the following rules:

- If the value does not match one or more of the values of the `type` property, then the input fails validation.
- If the length of the value in bytes exceeds the value of the `maxLength` property, then the input fails validation.

User Agent Considerations

When providing a control to the user, the user agent may choose to assist the user in providing valid input. If the value has a hint of `content`, the user agent may choose to make items that match the `type` and/or `maxLength` properties more prominent.

Options

The `options` property of a specification describes the location of a value that represents the options available for an input.

Format Rules

If the `input` property is present and the value has a hint of `text`, the `options` property is OPTIONAL. Otherwise, the `options` property MUST NOT be present. If present, it MUST be an an object matching the following rules:

- MUST be a valid URI.

Example of Remote Options

```
{
  "favoriteMovie": null,
  "spec": {
    "hints": [ "form" ],
    "children": [
      {
        "name": "favoriteMovie",
        "hints": [ "urn:com:example:movie-title", "text" ],
        "input": true,
        "options": "http://www.example.com/movies/fav/options"
      }
    ]
  }
}
```

Example of Inline Options

```
{
  "favoriteMovie": null,
  "listOfMovies": [
    "Fletch",
    "Caddyshack",
    "Christmas Vacation"
  ],
  "spec": {
    "hints": [ "form" ],
    "children": [
      {
        "name": "favoriteMovie",
        "hints": [ "urn:com:example:movie-title", "text" ],
        "input": true,
        "options": "#listOfMovies"
      },
      {
        "name": "listOfMovies",
        "hints": [ "array" ],
        "children": {
          "hints": [ "urn:com:example:movie-title", "text" ]
        }
      }
    ]
  }
}
```

User Agent Rules

- A `media` object must be displayed for the input property instead of an input control. The content of the `media` object is the value targeted by the `options` property.
- While displaying the `media` object content, the user agent MUST consider any value an option if it has a hint matching the most specific hint of the input property.
- After resolving the options, the user agent MUST provide the user with a control to select

each option. If the option has no siblings that are options, then the option's parent MAY be considered the control to select the option.

Label

The `label` property of a specification describes how to label a value.

Format Rules

- The `label` property is OPTIONAL.
- If present, the value MUST be a same-document reference to the source of the label content.

Example

```
{
  "commentsLabel": "What did you think of the movie?",
  "comments": null,
  "spec":{
    "hints": [ "form" ],
    "children": [
      {
        "name": "commentsLabel",
        "hints": [ "text" ]
      },
      {
        "name": "comments",
        "hints": [ "text" ],
        "input": true,
        "label": "#commentsLabel"
      }
    ]
  }
}
```

Displaying Content

Selecting a Media Object for Document Display

When a document is retrieved as a result of an interaction with a `link`, `submit`, or from within an embedded document, the user agent MUST attempt to select a `media` object, according to the following steps:

1. Let "origin" equal the `link`, `submit`, or `media` object that is the source of the transition.
2. Iterate over the origin and its descendents, via depth-first traversal, and select the first `media` object with a `scope` that includes the realm of the document, according to the "Normalization and Comparison" rules for realm uri.
3. If no `media` object has been selected but the origin has a parent, let origin equal the parent

and repeat step 2. If the origin has no parent, but is contained in a `media` object, then let origin equal the containing `media` object and repeat step 2. Repeat until a `media` object is selected or the origin has no remaining ancestors.

The user agent MUST update the embedded document of the selected `media` object and display it to the user. If no `media` object is selected, then the user agent MUST display the document at the root.

Submitting Form Data

When a `submit` control is invoked, the user agent MUST prepare a form data set, encode the form data set into a representation, and submit the representation.

Preparing a Form Data Set

To prepare a form data set:

- Let *form data set* be an empty set.
- Let *form* be the `submit` object's nearest `form` ancestor if found.
- If *form* does not exist, stop preparing a form data set.
- Let *inputs* be the descendants of *form* (in depth-first order) that have an `input` specification.

For each *input* in *inputs*:

- Let *result* be a tuple having *name* and *value* components.
  - Let *name* be the value of the *input* specification's `input.name` property, if it exists. Otherwise, let *name* be the `name` of the *input*, if it has a `name`. Otherwise, let *name* be the `name` of the *input's* first named ancestor.
  - Let *value* be a tuple having *data* and *type* components.
    - For `text` inputs, let *data* be the value's JSON value (as defined in RFC 4627, Section 2.1) if the value is not `null` or `undefined`; otherwise, let *data* be an empty string. Let *type* be an empty string.
    - For `content` inputs, let *data* and *type* be the value of the `content` object's `data` and `type` properties.
- Append *result* to *form data set.*

Encoding a Form Data Set

URL Encoding

When the value of the `submit` object's `enctype` property is `application/x-www-form-urlencoded`, the form data set MUST be encoded as follows:

- Let *result* be an empty string.
- For each *tuple* in the *form data set*:
  - Encode the *name* component according to URL Character Encoding and append to *result*.
  - Append the character '=' (U+003D) to *result.*
  - If the *value* component's *type* is an empty string, encode its *data* component according to URL Character Encoding and append to *result*; otherwise, continue to

the next step.
    - If the current *tuple* is not the last, append the character '&' (U+0026) to *result*.
- Let the *content type* of *result* be `application/x-www-form-urlencoded`.

URL Character Encoding

Encode all characters that are not in the "unreserved" set of RFC 3986, Section 2.3, according to the rules of percent encoding documented in RFC 3986, Sections 2.1 and 2.4.

Multipart Form Encoding

When the value of the `submit` object's `enctype` property is `multipart/form-data`, the form data set MUST be encoded according to the rules of RFC 2388 as follows:

- This section uses the concepts *boundary*, *field*, *field name*, *field value*, and *field content type* as defined in RFC 2388, Section 4.1. Each *tuple* in the *form data set* is correlated to these concepts as follows:
    - The *name* component correlates to *field name*. The *field name* MUST be encoded as UTF-8.
    - The *data* component of the *value* component correlates to *field value*.
    - The *type* component of the *value* component correlates to *field content type*. If the *field content type* is an empty string, the resulting part MUST NOT have a Content-Type header specified, and the *field value* MUST be encoded as UTF-8.
- Let *result* be an empty sequence of bytes.
- For each *tuple* in the *form data set*, encode and append each *tuple* to *result* according to the rules above and RFC 2388, Section 4.1.
- Let the *content type* of *result* be `multipart/form-data; boundary={boundary}`, where `{boundary}` is the value of *boundary*.

Other Encodings

Other encodings may be supported through extensions to this specification. If the user agent does not understand the `submit` object's `enctype`, it MUST encode the form data set using Multipart Form Encoding.

Submitting Encoded Data

HTTP and HTTPS

If the method is `GET`, `DELETE`, or another method that does not support an entity-body, submit the encoded data as follows:

- Let *result* be the encoded form data set.
- Let *method* be the `submit` object's `method` property value.
- Let *URL* be the submit object's `action` property value.
- Let the *query component* of the *URL* be *result*.
- Send an HTTP message, let its method be *method*, and let its retrieval URL be *URL*.

If the method is `POST`, `PUT`, or another method that supports an entity-body, submit the encoded data as follows:

- Let *result* be the encoded form data set.

- Let *method* be the `submit` object's `method` property value.
- Let *URL* be the submit object's `action` property.
- Let the URL's *query component* be empty.
- Send an HTTP message, let its method be *method*, let its retrieval URL be *URL*, let its "Content-Type" header be the content type of *result*, and let its entity body be *result*.

Other Schemes

For other schemes, user agents should act in a manner analogous to that defined in HTML, Section 4.10.22.3.

Extensions

Lynx can be extended with new hints or new specification properties. A new hint extension could be used to describe the meaning of a value more specifically than a base hint. A new specification property extension could be used to help a user agent improve a user's experience with a value. For example, it may describe a relationship between two values or it may describe an action that a user agent may take on behalf of a user.

Hint Extensions

While sufficient for describing the most basic characteristics of values, the hints defined in this document are not sufficient to meaningfully describe all values. Authors will find benefit in defining hints to describe the values in their domain.

Format Rules

- MUST conform to the rules defined for hint objects.
- MUST document the name of the hint. The name MUST be an absolute URI.
- SHOULD document the meaning of the hint and its relationship to a less specific hint, if applicable. When this relationship is identified, the two hints MUST always be present together in a `hints` array in proper order.
- SHOULD document the JSON value(s) the hint can describe and any relevant format rules.
- SHOULD document rules or considerations for how user agents process, interpret, and convey the value.

Example

The following is an example of the documentation that an author could use to define an extended hint:

Rating Hint

Name

http://example.com/rating

Meaning

The rating hint is used to describe values that represent how happy a person is with a product or service.

Related Hint

The rating hint is more specific than the `text` hint in Lynx.

Format Rules

- MUST be a number with a value from 1 to 5 (inclusive).
- If a fractional component is present and is not equal to 0.5, then the fractional component should be rounded to the nearest 0.5.

Example

```
{
  "title": "Good Will Hunting",
  "averageReview": 4.5,
  "spec": {
    "hints": [ "object" ],
    "children": [
      {
        "name": "title",
        "hints": [ "title", "text" ]
      },
      {
        "name": "averageReview",
        "hints": [
          "http://example.com/rating",
          "text"
        ]
      }
    ]
  }
}
```

User Agent Considerations

A user agent with a graphical user interface should consider using a more graphical representation to convey the rating to the user. For example, it is very common to use a graphical meter comprised of 5 stars where the number of filled stars represents the rating value.

Authoring Considerations

When defining an extended hint, authors are strongly encouraged to relate the new hint to a base hint. Doing so will ensure that the value can still be conveyed by user agents that understand Lynx hints even if they do not understand the extended hint.

Specification Property Extensions

Specification property extensions are used to help a user agent improve a user's experience with a value. They provide additional metadata for values independent of their hints or other specification properties. For example, it may describe a relationship between two values or it may describe an action that a user agent may take on behalf of a user.

Format Rules

- The property name SHOULD be an absolute URI.
- SHOULD document the meaning of the property value.
- SHOULD document the format rules for the property value.
- SHOULD document rules or considerations for how user agents process and interpret the property value.

Example

The following is an example of the documentation that an author could use to define an extended specification property:

Expanded

Name

http://example.com/expanded

Meaning

Indicates whether content is collapsible and whether the content is currently expanded or collapsed.

Format Rules

If present, MUST have a value of `true` or `false`.

Example

```
{
  "http://example.com/expanded" : true
}
```

User Agent Rules

- The user agent MUST provide the user with a control to interact with the expandable value in order to expand or collapse it.

- If the value described by the specification has a `title` the user agent MUST ensure that it is visible in collapsed and expanded states.

Authoring Considerations

- Authors should understand that by not using a property name that is an absolute URI, they are creating a risk of collision with other specification property extensions or future revisions of this media type.
- Authors should design documents that allow a user to successfully complete a process whether or not the user agent understands a particular specification extension.

Appendix

Realm URI

A realm is a field or domain of activity or interest that may be used to organize information (see Section 6.5.4 "Design of Media Types", [Fielding Dissertation]).

A realm URI is used to identify and refer to a realm. By disclosing a realm, the server provides additional information to user agents, or other components, that they may use to apply special treatment to the content within the realm.

Format Rules

MUST be an absolute URI.

Normalization and Comparison

Given realm URI A and realm URI B, if A starts with B (using a simple string comparison after URI normalization as defined in Section 6 "Normalization and Comparison", [RFC 3986]) then the realm identified by A is considered to be within the realm identified by B.

References

RFC 2119

"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997

RFC 4627

"The application/json Media Type for JavaScript Object Notation (JSON)", D. Crockford, July 2006

RFC 3986

"Uniform Resource Identifier (URI): Generic Syntax", T. Berners-Lee, January 2005

RFC 6838

"Media Type Specifications and Registration Procedures", N.Freed, J. Klensin, and T.Hansen, January 2013

ECMA 262

ECMA, "ECMAScript® Language Specification", June 2011

RFC 2616

"Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, June 1999.

Fielding Dissertation

"Architectual Styles and the Design of Network-based Software Architectures" (https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)", R. Fielding, 2000.

HTML

"HTML 5: A vocabulary and associated APIs for HTML and XHTML", W3C, October 2014