



Packaged Web Apps (Widgets) - Packaging and XML Configuration (Second Edition)

W3C Recommendation 27 November 2012
obsoleted 11 October 2018

This version:

<https://www.w3.org/TR/2018/OBSL-widgets-20181011/>

Latest version:

<http://www.w3.org/TR/widgets/>

Previous versions:

<http://www.w3.org/TR/2012/REC-widgets-20121127/>

Latest editor's draft:

<http://w3c.github.io/packaged-webapps/packaging/>

Test suite:

<http://dev.w3.org/2006/waf/widgets/test-suite/>

Implementation report:

<http://dev.w3.org/2006/waf/widgets/imp-report/>

Editor:

[Marcos Cáceres](#), W3C Invited Expert

Please refer to the [errata](#) for this document, which includes some normative corrections.

See also [translations](#).

See also [differences](#) between this and last published draft and this document.

[Copyright](#) © 2012 [W3C](#)® ([MIT](#), [ERCIM](#), [Keio](#)), All Rights Reserved. W3C [liability](#), [trademark](#) and [document use](#) rules apply.

Abstract

This specification updates the *Widget Packaging and XML Configuration*, and addresses some [errata](#) found in the original recommendation. It also updates the name of the specification, to be more *in vogue* with industry trends towards the naming of this class of application.

This specification standardizes a packaging format and metadata for a class of software known commonly as *packaged apps* or *widgets*. Unlike traditional user interface widgets (e.g., buttons, input boxes, toolbars, etc.), widgets as specified in this document are full-fledged client-side applications that are authored using technologies such as HTML and then packaged for distribution. Examples range from simple clocks, stock tickers, news casters, games and weather forecasters, to complex applications that pull data from multiple sources to be "mashed-up" and presented to a user in some interesting and useful way.

The specification relies on PKWare's Zip specification as the archive format, XML as a configuration document format, and a series of steps that runtimes follow when processing and verifying various aspects of a package. The packaging format acts as a container for files used by a widget. The configuration document is an XML vocabulary that declares metadata and configuration parameters for a widget. The steps for processing a widget package describe the expected behavior and means of error handling for runtimes while processing the packaging format, configuration document, and other relevant files.

This specification is part of the [Widgets family of specifications](#), which together standardize widgets as a whole.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index](https://www.w3.org/TR/) at <https://www.w3.org/TR/>.

This specification is obsolete and should no longer be used as a basis for implementation.

The Widget specifications became W3C Recommendations in 2012-2013. They were designed to enable interactive single purpose application for displaying and/or updating local data or data on the Web, packaged in a way to allow a single download and installation on a user's machine or mobile device.

Since 2013, Widgets has had limited deployment and its usage has been reduced since then. [Service Workers](#) and [Web App Manifest](#) are considered to provide better solutions nowadays.

For purposes of the W3C Patent Policy this [Obsolete Recommendation](#) has the same status as an active Recommendation; it retains licensing commitments and remains available as a reference for old implementations but is no longer recommended for future implementation.

Table of Contents

- [1 Introduction](#)
 - [1.1 Design Goals and Requirements](#)
 - [1.2 How This Document is Organized](#)
 - [1.3 Typographic Conventions](#)
 - [1.4 The Widget Family of Specifications](#)
- [2 Conformance](#)
- [3 Definitions](#)
 - [3.1 Character Definitions](#)
- [4 User Agents](#)
 - [4.1 Optional Aspects of the Zip Specification](#)
- [5 Zip Archive](#)
 - [5.1 Compression Methods](#)
 - [5.2 Version of Zip Needed to Extract a File Entry](#)
 - [5.3 Zip Relative Paths](#)
 - [5.4 Interoperability Considerations](#)
- [6 Widget Packages](#)
 - [6.1 Invalid Widget Package](#)
 - [6.2 Files and Folders](#)
 - [6.3 Reserved File and Folder Names](#)
 - [6.4 Digital Signatures](#)
 - [6.5 Start Files](#)
 - [6.5.1 Custom Start File](#)
 - [6.5.2 Default Start Files](#)
 - [6.6 Icons](#)
 - [6.6.1 Custom Icons](#)
 - [6.6.2 Default Icons](#)
 - [6.7 Media Type](#)
 - [6.8 File Extension](#)
- [7 Configuration Document](#)
 - [7.1 Example Configuration Document](#)
 - [7.2 Namespace](#)
 - [7.3 Proprietary Extensions](#)
 - [7.4 Types of Attributes](#)
 - [7.5 Global Attributes](#)
 - [7.5.1 The `xml:lang` Attribute](#)
 - [7.5.2 The `dir` Attribute](#)
 - [7.5.3 Examples of Usage](#)

- [7.6 The `widget` Element and its Attributes](#)
 - [7.6.1 The `id` Attribute](#)
 - [7.6.2 The `version` Attribute](#)
 - [7.6.3 The `height` Attribute](#)
 - [7.6.4 The `width` Attribute](#)
 - [7.6.5 The `viewmodes` Attribute](#)
 - [7.6.6 The `defaultlocale` attribute](#)
 - [7.6.7 Example of Usage](#)
 - [7.6.8 Example of Usage of the `defaultlocale` attribute](#)
- [7.7 The `name` Element and its Attributes](#)
 - [7.7.1 The `short` Attribute](#)
 - [7.7.2 Example of Usage](#)
- [7.8 The `description` Element and its Attributes](#)
 - [7.8.1 Example of Usage](#)
- [7.9 The `author` Element and its Attributes](#)
 - [7.9.1 The `href` Attribute](#)
 - [7.9.2 The `email` Attribute](#)
 - [7.9.3 Example of Usage](#)
- [7.10 The `license` Element and its Attributes](#)
 - [7.10.1 The `href` Attribute](#)
 - [7.10.2 Example of Usage](#)
- [7.11 The `icon` Element and its Attributes](#)
 - [7.11.1 The `src` Attribute](#)
 - [7.11.2 The `width` Attribute](#)
 - [7.11.3 The `height` Attribute](#)
 - [7.11.4 Example of Usage](#)
- [7.12 The `content` Element and its Attributes](#)
 - [7.12.1 The `src` Attribute](#)
 - [7.12.2 The `type` Attribute](#)
 - [7.12.3 The `encoding` Attribute](#)
 - [7.12.4 Example of Usage](#)
- [7.13 The `feature` Element and its Attributes](#)
 - [7.13.1 The `name` Attribute](#)
 - [7.13.2 The `required` Attribute](#)
 - [7.13.3 Example of Usage](#)
- [7.14 The `param` Element and its Attributes](#)
 - [7.14.1 The `name` Attribute](#)
 - [7.14.2 The `value` Attribute](#)
 - [7.14.3 Example of Usage](#)
- [7.15 The `preference` Element and its Attributes](#)
 - [7.15.1 The `name` Attribute](#)
 - [7.15.2 The `value` Attribute](#)
 - [7.15.3 The `readonly` Attribute](#)
 - [7.15.4 Example of Usage](#)
- [7.16 The `span` Element and its Attributes](#)
 - [7.16.1 Example of Usage](#)
- [8 Internationalization and localization](#)
 - [8.1 Bidirectional text](#)
 - [8.2 Localization Model](#)
 - [8.3 Folder-based localization](#)
 - [8.4 Element-Based Localization](#)
 - [8.5 Localization Examples](#)
 - [8.5.1 Simple Example](#)
 - [8.5.2 Complex Example](#)
 - [8.5.3 Fallback Behavior Example](#)
- [9 Steps for Processing a Widget Package](#)
 - [9.1 Processing Rules](#)
 - [9.1.1 Rule for Verifying a Zip Archive](#)
 - [9.1.2 Rule for Extracting File Data from a File Entry](#)
 - [9.1.3 Rule for Finding a File Within a Widget Package](#)
 - [9.1.4 Rule for Determining Directionality](#)
 - [9.1.5 Rule for Getting a Single Attribute Value](#)

[9.1.6 Rule for Getting a List of Keywords From an Attribute](#)

[9.1.7 Rule for Verifying a File Entry](#)

[9.1.8 Rule for Getting Text Content](#)

[9.1.9 Rule for Getting Text Content with Normalized White Space](#)

[9.1.10 Rule for Parsing a Non-negative Integer](#)

[9.1.11 Rule for Identifying the Media Type of a File](#)

[9.1.12 Rule for Deriving the *user agent locales*](#)

[9.1.13 Rule for Determining if a Potential Zip Archive is a Zip Archive](#)

[Step 1 - Acquire a Potential Zip Archive](#)

[9.1.1 Acquisition of a Potential Zip archive Labeled with a Media Type](#)

[9.1.2 Acquisition of Potential Zip Archive not Labeled with a Media Type](#)

[Step 2 - Verify the Zip Archive](#)

[Step 3 - Set the Configuration Defaults](#)

[Step 4 - Locate and Process the Digital Signature](#)

[Step 5 - Derive the User Agent's Locales](#)

[Step 6 - Locate the Configuration Document](#)

[Step 7 - Process the Configuration Document](#)

[9.1.1 Terminology Used in Processing Algorithm](#)

[9.1.2 Algorithm to Process a Configuration Document](#)

[Step 8 - Locate the Start File](#)

[Step 9 - Process the Default Icons](#)

[Appendix](#)

[Media Type Registration for *application/widget*](#)

[Linking To a Widget Package From a HTML Document](#)

[Table of Elements and Their Attributes](#)

[Acknowledgements](#)

[Normative References](#)

[Informative References](#)

1 Introduction

This section is non-normative.

Widgets are full-fledged client-side applications that are authored using Web standards such as [HTML](#) and packaged for distribution. They are typically downloaded and installed on a client machine or device where they run as stand-alone applications, but they can also be embedded into Web pages and run in a Web browser. Examples range from simple clocks, stock tickers, news casters, games and weather forecasters, to complex applications that pull data from multiple sources to be "mashed-up" and presented to a user in some interesting and useful way (see [\[Widgets-Landscape\]](#) for more information).

This specification is intended to specify a part of the Web platform closely related to [HTML](#).

1.1 Design Goals and Requirements

This section is non-normative.

The design goals and requirements for this specification are documented in the [\[Widgets-Requirements\]](#) document.

This document addresses the 25 requirements relating to "[Packaging](#)" and "[Configuration Document](#)" of the 30 April 2009 Working Draft of the *Widgets Requirements* Document:

1. [Packaging Format](#): see [packaging format](#).
2. [Media Type](#): see the [valid widget media type](#).
3. [File Extension](#): see [widget file extension](#).
4. [Internal Abstract Structure](#): see [Zip archive](#) and [widget package](#).
5. [Reserved Resource Names](#): see [reserved file names table](#).
6. [Addressing Scheme](#): see [valid path](#).
7. [Multilingual File Names](#): see [Zip relative path](#), particularly in respect to [support](#) for [UTF-8](#).
8. [Localization Guidelines](#): see [element-based localization](#), [folder-based localization](#).
9. [Automatic Localization](#): see [element-based localization](#), [folder-based localization](#).

10. [Device Independent Delivery](#): all aspects of this document where developed with this requirement in mind.
11. [Data Compression](#): see the [valid compression methods](#).
12. [Derive the Media Type of Resources](#): see the [rule for identifying the media type of a file](#).
13. [Format and Schema](#): see [configuration document](#), [table of configuration defaults](#), and the [\[Widgets-Relax NG Schema\]](#) for the configuration document.
14. [Widget Metadata](#): see [configuration document](#) (particularly the elements).
15. [Authorship Metadata](#): see the [author](#) element.
16. [Copyright Notice and License Metadata](#): see the [license](#) element.
17. [Visual Rendering Dimensions](#): see the [widget](#) element.
18. [Declarative Bootstrap](#) see the [content](#) element.
19. [Automated Bootstrap](#) see the [default start file](#).
20. [Iconic Representations](#): see the [icon](#) element, [default icons table](#) and [custom icons table](#).
21. [Configuration Parameters](#): the [param](#) element (used in conjunction with the [feature](#) element).
22. [Author-defined Start-up Values \(Preferences\)](#): see the [preference](#) element.
23. [Feature Access Declarations](#): see the [feature](#) element.
24. [Configuration Document Independence](#): see [configuration document](#).
25. [Preferred Display Mode](#): see the `viewmodes` attribute of the [widget](#) element.

1.2 How This Document is Organized

This section is non-normative.

This document is organized into two halves, but not explicitly marked as such. The first half defines the various aspects of what constitutes the [packaging format](#), the [configuration document](#), and [reserved](#) files, such as [default icons](#) and [locale folders](#). Where possible, the first half avoids describing aspects related to processing, which are described in detail in the second half of the document.

The second half, which starts with the section titled "[Steps for Processing a widget package](#)", defines the steps required to process a widget package as well as the expected behavior of a user agent as it processes the [packaging format](#), the configuration document, and attempts to find localized content. The second half of this document also deals with error handling in the event that a user agent encounters [unsupported](#) or missing files, or DOM nodes that are [in error](#) in the configuration document. Wherever processing is relevant, sections in the first half of the document link to sections in the second half of the document.

1.3 Typographic Conventions

This section is non-normative.

This section defines the **typographical conventions** used by this specification. Some text in this specification is non-normative. Non-normative text includes:

- sections marked with the text "*This section is non-normative*",
- authoring guidelines,
- examples,
- and notes.

Everything else in this specification is normative.

Defined terms appear as this **sample defined term**. Such terms are referenced as [sample defined term](#), providing a link back to the term definition.

Words that denote a conformance clause or testable assertion use keywords from [\[RFC2119\]](#): MUST, MUST NOT, SHOULD, RECOMMENDED, MAY and OPTIONAL. The keywords MUST, MUST NOT, SHOULD, RECOMMENDED, MAY and OPTIONAL in this specification are to be interpreted as described in [\[RFC2119\]](#).

Variables are formatted specially, e.g. *variable*. Code is also specially formatted, such as `code`.

Words in *italics* denote a formal definition given in an external specification.

This is an example. Examples are used to explain concepts or demonstrate how to use a feature. Examples are non-normative.

Note:

This is a note, it usually contains useful supplementary information in a non-normative form.

Authoring Guidelines:

This is an Authoring Guideline. Its purpose is to provide authors with best-practice authoring techniques. Authoring guidelines are non-normative.

1.4 The Widget Family of Specifications

This section is non-normative.

This specification is part of the **Widgets family of specifications**, which together standardize widgets as a whole. The [list of specifications](#) that make up the Widgets Family of Specifications can be found on the Working Group's wiki.

2 Conformance

There is only one class of product that can claim conformance to this specification: a [user agent](#).

Note:

Implementers can partially check their level of conformance to this specification by successfully passing the test cases of the [P&C-Test-Suite](#). Note, however, that passing all the tests in the test suite does not imply complete conformance to this specification; It only implies that the implementation conforms to aspects tested by the test suite (the test suite does not provide tests for any optional conformance clauses).

3 Definitions

The following terms are used throughout this specification so they are gathered here for the readers convenience. The following list of terms is not exhaustive; other terms are defined throughout this specification.

Arbitrary means that a character, or text string, or [file-name](#), or [folder-name](#) is not [reserved](#) for the purpose of this specification.

An **author** is a person who created a [widget package](#) or an authoring tool that generated a [widget package](#).

Initialization means a user agent procedurally stepping through the [steps for processing a widget package](#).

A *language tag* is a text string that matches the production of a Language-Tag defined in the [\[BCP47\]](#) specifications (see the [IANA Language Subtag Registry](#) for an authoritative list of possible values, see also the [Maintenance Agency for ISO 3166 country codes](#)).

A *media type* is defined in the [\[MIME\]](#) specification.

Reserved means that a character, or text string, or [file-name](#), or [folder-name](#) has a specified purpose and semantics in this specification or in some other specification or system. The intended purpose for any reserved thing is given when the term is used.

Supported means that a user agent implements a mentioned specification, or conformance clause, or is able to process or otherwise render mentioned [media type](#).

Unsupported means the user agent does not implement a mentioned specification, or [feature](#), or is unable to render or otherwise process a mentioned [media type](#).

A *widget* is defined by the [\[Widgets-Landscape\]](#) as "an end-user's conceptualization of an interactive single purpose application for displaying and/or updating local data or data on the Web, packaged in a way to allow a single download and installation on a user's machine, mobile phone, or Internet-enabled device". Because widgets are packaged, they can be shared by users without relying on [\[HTTP\]](#).

3.1 Character Definitions

This section groups common sets of [\[Unicode\]](#) code points into definitions for the purpose processing in this specification.

The **space characters** are code points marked in the [\[Unicode\]](#) specification with the property "White_Space", including, but not limited to the following list (see [\[Unicode\]](#) for the authoritative list):

- U+0020 SPACE,
- U+0009 CHARACTER TABULATION (tab),
- U+000A LINE FEED (LF),
- U+000B LINE TABULATION,
- U+000C FORM FEED (FF),
- U+000D CARRIAGE RETURN (CR),
- U+0085 NEL (control character next line)
- U+00A0 NBSP (NO-BREAK SPACE)
- U+1680 OGHAM SPACE MARK
- U+180E MONGOLIAN VOWEL SEPARATOR
- U+2000-U+200A (different sorts of spaces)
- U+2028 LS (LINE SEPARATOR)
- U+2029 PS (PARAGRAPH SEPARATOR)
- U+202F NNBS (NARROW NO-BREAK SPACE)
- U+205F MMSP (MEDIUM MATHEMATICAL SPACE)
- U+3000 IDEOGRAPHIC SPACE

The **Zip forbidden characters** are code points:

- U+0000 NUL-U+001F INFORMATION SEPARATOR 1,
- U+007F DELETE,
- U+003C LESS-THAN SIGN,
- U+003E GREATER-THAN SIGN,
- U+003A COLON,
- U+0022 QUOTATION MARK,
- U+002F SOLIDUS,
- U+005C REVERSE SOLIDUS,
- U+007C VERTICAL LINE,
- U+003F QUESTION MARK,
- U+002A ASTERISK,
- U+005E CIRCUMFLEX ACCENT,
- U+0060 GRAVE ACCENT,
- U+007B LEFT CURLY BRACKET,
- U+007D RIGHT CURLY BRACKET,
- U+0021 EXCLAMATION MARK.

4 User Agents

A **user agent** is an implementation of this specification that also [supports \[XML\]](#), [\[XMLNS\]](#), [\[UTF-8\]](#), [\[Unicode\]](#), [\[DOMCore\]](#), [\[SNIFF\]](#), and [\[ZIP\]](#) (see [optional aspects of the Zip specification](#)).

In addition to [widget packages](#), a **user agent** MAY [support](#) other legacy and proprietary application packaging formats.

It is OPTIONAL for a **user agent** to [support](#) the [optional aspects of the Zip specification](#).

Note:

The user agent described in this specification does not necessarily denote a "**widget user agent**" at large: that is, a user agent that implements all the specifications, and dependencies, defined in the [Widgets Family of Specifications](#). The user agent described in this specification is only concerned with how to processes [Zip archives](#) and [configuration documents](#).

4.1 Optional Aspects of the Zip Specification

The **optional aspects of the Zip specification** are as follows. These aspects represent general features defined in the [\[ZIP\]](#) specification that this specification does not make use of:

- Compression or decompression algorithms other than [\[Deflate\]](#) and [Stored](#).
- Zip64 extensions.
- Digital signature methods.
- Decryption methods.
- Patented aspects.

5 Zip Archive

The **Zip archive** file format, defined in the [\[ZIP\]](#) specification, is the **packaging format** for [widget packages](#).

A **file entry** is the data held by a *local file header*, *file data*, and (optional) *data descriptor*, as defined in the [\[ZIP\]](#) specification, for each physical file or folder contained in a [Zip archive](#).

A **potential Zip archive** is a data object claiming to be a [Zip archive](#), that has not been [verified](#) to be a [valid Zip archive](#).

A **valid Zip archive** is a data object that the user agent has verified as conforming to the production of a *.Zip file* as defined by the *Zip File Format Specification* [\[ZIP\]](#) and [meets the requirements of this specification](#) (See [Step 2](#)).

The **magic numbers for a Zip archive** is the byte sequence: 50 4B 03 04.

5.1 Compression Methods

A **compression method** is the compression algorithm or storage method that was used to encode the file data of a [file entry](#) when the zip archive was created by the author. The compression method that encoded the *file data* of a [file entry](#) is identified by the numeric value derived from the *compression method field* defined in the [\[ZIP\]](#) specification.

The **valid compression methods**, as indicated by the *compression method field*, for a [file entry](#) are:

8

Data is compressed using [\[Deflate\]](#).

0

Data is *Stored* (no compression), as defined in the [\[ZIP\]](#) specification.

Authoring Guidelines:

To ensure interoperability, compress [file entries](#) in [Zip archives](#) with [\[Deflate\]](#) or [Stored](#) (no compression); other compression methods can result in the Zip archive being treated as an [invalid widget package](#). Of the valid compression methods, [\[Deflate\]](#) is the preferred compression method.

5.2 Version of Zip Needed to Extract a File Entry

The *version needed to extract* is the 2-byte sequence in the [local file header](#) of a [file entry](#) that indicates the minimum supported version of the [\[ZIP\]](#) specification needed to extract the [file data](#).

The **valid versions needed to extract** values are as follows. Each value is assigned one or more meanings by the [\[ZIP\]](#) specification:

1.0

Default value specified in the [\[ZIP\]](#) specification.

2.0

The file data is compressed using [\[Deflate\]](#), or the file data is a folder, or the file has been encrypted using traditional PKWARE encryption.

Note:

If the Zip archive has been encrypted using traditional PKWARE encryption, then the user agent will treat the Zip archive as an [invalid widget package](#) in [Step 2](#).

5.3 Zip Relative Paths

A **Zip relative path** is the variable-length string derived from the *file name field* of the [local file header](#) of a [file entry](#).

Note:

A Zip relative path is said to be relative as it stores the string that represents file and folder names relative to where the Zip archive was created on a file system (e.g. `images/bg.png`), as opposed to storing an absolute path (e.g. `c:\images\bg.png`). The value of a Zip relative path will generally resemble the string value of a name of the file or folder(s) on the device on which the Zip archive was created, but with the exception of the path delimiter being a U+002F SOLIDUS "/" character. Note also that a Zip relative path is not a URI reference; Zip relative paths need to be converted to URI references before they can be used in context that make use of URIs.

A **valid Zip relative path** is one that matches the production of [zip-rel-path](#) in the following [\[ABNF\]](#):

```

zip-rel-path    = [locale-folder] *folder-name file-name /
                  [locale-folder] 1*folder-name
locale-folder  = %x6C %x6F %x63 %x61 %x6C %x65 %x73
                  "/" lang-tag "/"
folder-name    = file-name "/"
file-name      = 1*allowed-char
allowed-char   = safe-char / zip-UTF8-char
zip-UTF8-char  = UTF8-2 / UTF8-3 / UTF8-4
safe-char      = ALPHA / DIGIT / SP / "$" / "%" /
                  "' " / "-" / "_" / "@" / "~" /
                  "(" / ")" / "&" / "+" / "," /
                  "=" / "[" / "]" / "."
UTF8-2         = %xC2-DF UTF8-tail
UTF8-3         = %xE0 %xA0-BF UTF8-tail / %xE1-EC 2( UTF8-tail ) /
                  %xED %x80-9F UTF8-tail / %xEE-EF 2( UTF8-tail )
UTF8-4         = %xF0 %x90-BF 2( UTF8-tail ) / %xF1-F3 3( UTF8-tail ) /
                  %xF4 %x80-8F 2( UTF8-tail )
UTF8-tail     = %x80-BF
lang-tag       = primary-subtag *( "-" subtag )
primary-subtag = 1*8low-alpha
subtag         = 1*8(alphanum)
alphanum      = low-alpha / DIGIT
low-alpha     = %x61-7a
  
```

ALPHA, *DIGIT*, and *SP* are defined in the [\[ABNF\]](#) specification (but essentially represent alphanumerical characters and the U+0020 SPACE code point respectively).

5.4 Interoperability Considerations

This section is non-normative.

Some issues can arise with regards to character encodings of file names, the length of zip relative paths, and the use of certain strings as file names. This sections is intended to help authors avoid potential interoperability issues.

Authoring Guideline

5.4.1 Paths Lengths

This section is non-normative.

Authors need to be aware that having excessively long path names (e.g. over 120 characters) can also result in interoperability issues on some operating systems. This is because some operating systems have restrictions on how long a path length can be, so authors should try to keep the lengths of paths at less than 250 bytes. In addition, Unicode code points may require more than one byte to encode a character, which can result in a path whose length is less than 250 characters but whose size is greater than 250 bytes!

5.4.2 Character sets

This section is non-normative.

Authors need to be aware that, at the time of publication, there are interoperability issues with regards to using characters outside the `safe-chars` range for file or folder names in a Zip archive when using Zipping tools bundled with operating systems. The interoperability issues have arisen from non-conforming implementations of the [\[ZIP\]](#) specification across operating systems: very few, if any, correctly support encoding file names in Unicode.

In the case where the Zip relative path is encoded using [\[UTF-8\]](#), the [language encoding flag \(EFS\)](#) needs to be set.

If an author chooses to use the `utf8-chars`, they need to thoroughly test their widgets on various platforms prior to distribution; otherwise it is suggested that authors restrict file and folder names to the `safe-chars` (characters in the US-ASCII range).

5.4.3 File Names

This section is non-normative.

Authors need to avoid using the [Zip forbidden characters](#) when naming the files used by a widget. These characters are reserved to maintain interoperability across various file systems and with [\[URI\]](#)s.

Authors need to avoid using the following words as either a [folder](#) or a [file-name](#) in a [Zip relative path](#) as they are reserved by some operating systems (case-insensitive): CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, LPT9, CLOCKS\$. For example, the following names are ok: "CON-tact.txt", "printer.lpt1", "DCOM1.pdf". However, "com3.txt" "Lpt1", "CoM9.gif" would not be.

In addition, authors need to avoid having a "." U+002E FULL STOP as the last character of a file or folder name as some operating systems will remove the character when the file is extracted from the Zip archive onto the device. Furthermore, avoid having the space character ([SP](#)) at the start or end of a file name; and take caution when using the "+" U+002B PLUS SIGN, as it might cause issues on some operating systems.

6 Widget Packages

A **widget package** is a [valid Zip archive](#) that contains the following:

- One or more [start files](#), located at the [root of the widget package](#) and/or at the root of [locale folders](#) or referenced by a [content](#) element's [src](#) attribute.

- One [configuration document](#), located at the [root of the widget package](#).
- Zero or more [icons](#), either located at the [root of the widget package](#) and/or at the root of [locale folders](#) or referenced by the [icon](#) element's [src](#) attribute.
- Zero or more [arbitrary files](#) located either at the [root of the widget package](#) and/or in [arbitrary folders](#) or in [locale folders](#).
- Zero or more [digital signatures](#) located at the [root of the widget package](#).

Note:

See [step 1](#) - Acquire a Potential Zip Archive for instructions on how to process a widget package.

6.1 Invalid Widget Package

During the [steps for processing a widget package](#), certain error conditions can result in a [Zip archive](#) being treated as an [invalid widget package](#). An **invalid Widget package** is a condition whereby a Zip archive, or a file within the Zip archive, is deemed to be corrupt beyond recovery or is non-conforming to, or [unsupported](#) by, this specification in such a way that it would not be possible for the user agent to continue processing.

6.2 Files and Folders

The **root of the widget package** is the top-most path level of the [Zip archive](#). The root of the widget package contains [files](#) and [folders](#), some of which are [reserved](#) (see [reserved file names table](#)).

A **file** is the decompressed physical representation of a [file entry](#) (i.e., a file extracted into its physical form as it was prior to being added to the [Zip archive](#)).

A **folder** is a [file entry](#) whose [file name field](#) matches the production of [folder-name](#) in a [valid Zip relative path](#) (the last character of the [file name field](#) is a U+002F SOLIDUS) and whose [version needed to extract](#) is 2.0.

A **processable file** is a [file](#) that:

- Exists within the [widget package](#).
- When the [rule for verifying a file entry](#) is applied to the file, it returns true.
- Is of a [media type supported](#) by the user agent, determined applying the [rule for identifying the media type of a file](#) to the file in question.

6.3 Reserved File and Folder Names

The [reserved file names table](#), below, contains a list of file names that are [reserved](#) for some purpose by this specification. The first column of the reserved file names table contains a case-sensitive list of file names. The second column of the table denotes the purpose for which the file name is [reserved](#).

Reserved File Names Table

| file name | Type | reserved for purpose |
|------------|----------------------|--|
| config.xml | file | Configuration document |
| icon.png | file | Default icon |
| icon.gif | file | Default icon |
| icon.jpg | file | Default icon |
| icon.ico | file | Default icon |
| icon.svg | file | Default icon |
| index.html | file | Default start file |

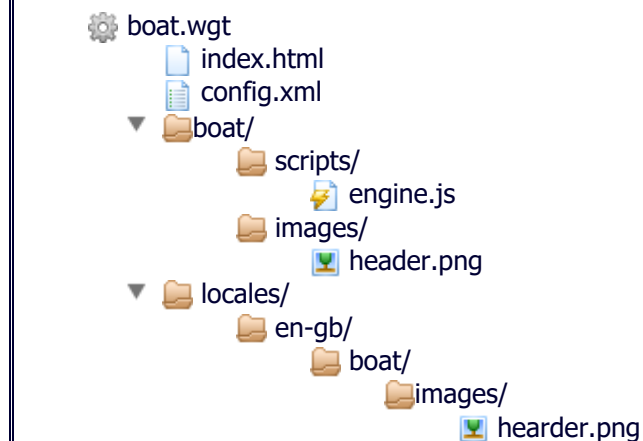
| file name | Type | reserved for purpose |
|-------------|--------|---|
| config.xml | file | Configuration document |
| index.htm | file | Default start file |
| index.svg | file | Default start file |
| index.xhtml | file | Default start file |
| index.xht | file | Default start file |
| locales | folder | Container for localized content |

Files named using the [naming convention for distributor signature](#) and the [naming convention for an author signature](#), as defined in the [\[Widgets-DigSig\]](#) specification, are also reserved in this specification.

Authoring Guideline:

Authors are strongly encourage to package all files, except the reserved files and the container for localized content, within a single subdirectory named, for instance, after the widget. This is to avoid unzipping several files into the end-user's current working directory. Future versions of this specification may include rules for locating index.html and config.xml within such directories.

For example, best-practice for packaging a widget would look something like this:



6.4 Digital Signatures

A [widget package](#) contains a **digital signature**, and hence is **digitally signed**, if the widget package contains one or more [files](#) that conform to the [\[Widgets-DigSig\]](#) specification.

6.5 Start Files

A **start file** designates a [file](#) from the widget package to be loaded by the user agent when it instantiates the widget. This specification defines two kinds of start file: [custom start file](#) and [default start file](#).

6.5.1 Custom Start File

A **custom start file** is a [processable file](#) inside the [widget package](#) identified by a [content](#) element's [src](#) attribute.

6.5.2 Default Start Files

A **default start file** is a [reserved start file](#) at the [root of the widget package](#) or at the root of a [locale folder](#) whose file name case-sensitively matches a file name given in the file name column of the [default start files table](#), and whose [media type](#) matches the [media type](#) given in the media type column of the table.

It is OPTIONAL for a [user agent](#) to [support](#) the [media types](#) listed in the [default start files table](#).

If a user agent encounters a [file](#) matching a file name given in the file name column of the [default start files table](#) in an [arbitrary folder](#), then [user agent](#) MUST treat that file as an [arbitrary file](#). {ta-RRZxvTFHx}

For example, "foo/bar/index.html" would be treated as an [arbitrary file](#).

Default Start Files Table

| file name | media type |
|-------------|-----------------------|
| index.htm | text/html |
| index.html | text/html |
| index.svg | image/svg+xml |
| index.xhtml | application/xhtml+xml |
| index.xht | application/xhtml+xml |

Note:

See [Step 8](#) for instructions on finding a [default start file](#).

Authoring Guidelines:

Always include at least one [start file](#) in a widget package.

6.6 Icons

An **icon** is a [file](#) that is used to represent the widget in various application contexts (e.g. the icon that the user activates to instantiate a widget, or an icon in a dock or task bar or some other visual context). The icon is intended to help users of visual browsers to recognize the widget at a glance. There are two kinds of icons defined by this specification, [custom icons](#) and [default icons](#).

6.6.1 Custom Icons

A **custom icon** is an [icon](#) explicitly declared by an author via an [i:icon](#) element in a [configuration document](#). A [custom icon](#) can be located at the [root of the widget package](#), or at the root of a [locale folder](#), or in an [arbitrary folder](#).

6.6.2 Default Icons

A **default icon** is a [reserved icon](#), either at the [root of the widget package](#) or at the root of a [locale folder](#), whose file name case-sensitively and exactly matches a file name given in the file name column of the [default icons table](#).

Default Icons Table

| file name | media type |
|-----------|--------------------------|
| icon.svg | image/svg+xml |
| icon.ico | image/vnd.microsoft.icon |
| icon.png | image/png |
| icon.gif | image/gif |
| icon.jpg | image/jpeg |

It is OPTIONAL for a [user agent](#) to [support](#) the [media types](#) listed in the [default icons table](#).

6.7 Media Type

The **valid widget media type** is the string application/widget.

Authoring Guidelines:

If the protocol over which the [widget package](#) is transferred [supports](#) the [MIME] specification (e.g. [HTTP](#)), then make sure that the widget is labeled with an `application/widget` media type. Failure to correctly label a [widget package](#) can result in the widget package being treated as an [invalid widget package](#).

6.8 File Extension

A **widget file extension** is the text string that case-insensitively matches the string `".wgt"` (e.g. `.wgt`, `.Wgt`, `.WgT`, etc. are all valid).

For example in `widget.WGT`, the `".WGT"` component is the file extension.

Authoring Guidelines:

If it is anticipated that the widget will be distributed by means lacking MIME support, then include the widget file extension. The widget file extension is not necessary if the widget package is labeled as `application/widget` when served over HTTP. The [widget file extension](#) is required for widget packages on systems where it is customary for file names to include a file extension that symbolizes (or is associated with) a [media type](#).

7 Configuration Document

A **configuration document** is an [XML](#) document that has a [widget](#) element at its root that is in the [widget namespace](#). A widget package has exactly one configuration document located at the [root of the widget package](#).

Note:

Please see [Step 7](#) for details of how the elements of the [configuration document](#) are processed by a user agent.

A **valid configuration document file name** is the string `config.xml`.

A [user agent](#) MUST treat any [file](#) in an [arbitrary folder](#) or [locale folders](#) that uses the file name `config.xml` as an [arbitrary](#) file. {ta-dxzVDWpaWg}

Authoring Guidelines:

Be sure to always include a [configuration document](#) at the [root of the widget package](#) and that the `config.xml` file name is in lowercase form. To ensure interoperability, encode the [configuration document](#) as [UTF-8](#).

7.1 Example Configuration Document

The following is an example of a typical [configuration document](#):

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns
  id          = "http://www.w3.org/ns/widgets"
  version     = "http://example.org/examplewidget"
  version     = "2.0 Beta"
  height      = "200"
  width       = "200"
  viewmodes   = "fullscreen">

  <name short="Example 2.0">
    The example Widget!
  </name>

  <feature name="http://example.com/camera">
    <param name="autofocus" value="true"/>
  </feature>
</widget>
```

```

</feature>

<preference name      = "apikey"
            value     = "ea31ad3a23fd2f"
            readonly  = "true" />

<description>
  A sample widget to demonstrate some of the possibilities.
</description>

<author href  = "http://foo-bar.example.org/"
        email = "foo-bar@example.org">Foo Bar Corp</author>

<icon src="icons/example.png"/>
<icon src="icons/boo.png"/>
<content src="myWidget.html"/>

<license>
Example license (based on MIT License)
Copyright (c) 2008 The Foo Bar Corp.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
INSULT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
</license>
</widget>

```

Note:

Implementers are encouraged to expose relevant information provided by the [configuration document](#) to the user. Having "visual metadata" encourages authors to make full use of the configuration document format. See [Step 7](#) for instructions on how to process a configuration document.

Authoring Guidelines:

The only mandatory element in a configuration document is the [widget](#) element. All other elements and their respective attributes are optional. The following example shows the smallest possible configuration document that a user agent will be able to process. The reason to include this sole element is to explicitly inform a user agent or conformance checker that this zip file attempts to conform to this specification.

```

<!-- example of the smallest possible conforming configuration document -->
<widget xmlns="http://www.w3.org/ns/widgets"/>

```

7.2 Namespace

The **widget namespace** URI for a [configuration document](#) is <http://www.w3.org/ns/widgets> [\[XMLNS\]](#).

No provision is made for an explicit version number in this specification. If a future version of this specification requires explicit versioning of the document format, a different namespace will be used.

Authoring Guidelines:

Be sure to declare the widget namespace as part of the [widget](#) element. If the namespace is absent, then the widget will be treated by the user agent as an [invalid widget package](#).

7.3 Proprietary Extensions

This section is non-normative.

Implementers or authors intending to extend the [configuration document](#) format with their own [XML](#) elements and attributes (or those defined in other specifications) can do so by using a separate [XMLNS](#) namespace. This specification does not define a model for processing [XML](#) elements outside the [widget namespace](#) (they are simply [ignored](#) during processing).

Example of extending the [configuration document](#) format:

```
<widget xmlns="http://www.w3.org/ns/widgets"
  xmlns:ex="http://example.org/"
  <icon src="idle.png" ex:role="inactive"/>

  <icon src="big.png" ex:role="big"/>
  <ex:datasource>{'a':'b','c':'d'}</ex:datasource>

  <content src="widget.html"/>
</widget>
```

7.4 Types of Attributes

This section defines the different **attribute types** used in the [configuration document](#) and what constitutes valid values for those attribute types.

An attribute is **invalid** if its value does not conform to its said [attribute type](#); that is, if the value of the attribute is [in error](#) given the processing rules for that type of attribute.

Boolean attribute

A boolean attribute is a [keyword attribute](#) that can only be used with a valid boolean value. A **valid boolean value** is a [keyword](#) that case-sensitively matches `true` or `false`. Unless specified otherwise, the default behavior, which is used when the attribute is omitted or has a value other than the two allowed values, is `false`. The way a user agent interprets a boolean attribute is defined as part of an attribute's definition (see, for example, the [feature](#) element's [required](#) attribute).

String attribute

The value of a string attribute is any string that conforms to [XML](#) as a valid string for an XML attribute. The purpose of this attribute type is to classify strings that are not affected by the [dir](#) attribute, such as the [email](#) attribute (i.e., these attributes will not be treated as [displayable-strings](#) during [Step 7](#)).

Displayable-string attribute

An attribute whose primary purpose is to convey human readable information, such as the [name](#) element's [short](#) attribute and the widget element's [version](#) attribute.

Keyword attribute

A **keyword** is a string that is [reserved](#) for the purpose of this specification. The value of a keyword attribute is a [keyword](#) that is one of a finite set specified in the attribute's definition in the case given in this specification.

Keyword list attribute

An attribute defined as taking one or more [keywords](#) as a value, which are separated by [space characters](#).

Media type attribute

An attribute whose value is defined as containing a valid media type. A **valid media type** is string that matches the production for [valid-MIME-type](#) in the following [ABNF]:

```
valid-MIME-type = type "/" subtype *(";" parameter)
```

The *type*, *subtype*, and *parameter* tokens are defined in the [MIME] specification.

Language attribute

An attribute whose value is defined as containing a **valid language tag** (see the [IANA Language Subtag Registry](#) for an authoritative list of possible values, see also the [Maintenance Agency for ISO 3166 country codes](#)). A valid language tag is a string that conforms to the production of a Language-Tag, as defined in the [BCP47] specification.

Numeric attribute

The value of a numeric attribute is a string containing a valid non-negative integer. A **valid non-negative integer** is a string that consists of one or more code points in the range U+0030 DIGIT ZERO (0) to U+0039 DIGIT NINE (9). For example, the strings "002", "323", "23214", and so on.

Path attribute

An attribute defined as containing a valid path. A **valid path** is one that matches the production of a [Zip-rel-path](#) or a Zip-abs-path.

Authoring Guidelines:

A valid path is not a URI: a valid path represents a hierarchical path to a file inside a [Zip archive](#), which exactly matches the value of a [file name field](#) of a [local file header](#) of a [file entry](#). This means that valid paths need not be URL encoded.

IRI attribute

An attribute defined as containing a valid IRI. A **valid IRI** is one that matches the [IRI](#) token of the [IRI] specification.

Authoring Guidelines:

Because of the risk of confusion between IRIs that would be equivalent if dereferenced, the use of %-escaped characters in feature names is strongly discouraged.

Version attribute

A [displayable-string attribute](#) whose value is any arbitrary string value (possibly empty) within the constraints allowed for [XML] attributes. This specification does not mandate any specific format, semantics, or special processing rule for the format of a [version attribute](#).

Authoring Guidelines:

For the purpose of this specification, the structure of version tags has no semantics; they are just treated as arbitrary strings (e.g. '1.0' is not less than '2.0', but is simply different). However, for the sake of consistency, one can choose to use the following [ABNF]:

```
rec-version-tag = 1*DIGIT "." 1*DIGIT [ "." 1*DIGIT ]
                 *[ 1*ALPHA / SP / 1*DIGIT ]
```

Example version tags:

- Version 1.0 Beta
- 1.0 RC1
- 1.0-Build/1580
- Joey the dog [5.1.2100]

Example of [rec-version-tag](#):

- 1.0
- 1.10.1 beta1
- 1.02.12 RC1

7.5 Global Attributes

This section describes the behavior and expected usage of other relevant attributes that are part of the [XML](#) specification and this specification. In this specification, these attributes are referred to as **global attributes** because they can be used on any element in a [configuration document](#).

Although [global attributes](#) can be used on any element in this specification, they sometimes have no effect on the element on which they are used. For example, applying [dir](#) attribute on an [icon](#) element will have no effect on the [icon](#) elements or any of its attributes. What effect specifying a global attribute has on an elements is determined by [Step 7](#) of this specification.

Authoring Guidelines:

Although it is optional for an author to use any global attributes, their usage is recommended when appropriate (e.g., when declaring the language will help with legibility and when directional information will assist the user agent render text correctly).

7.5.1 The `xml:lang` Attribute

A [language attribute](#) that specifies, through a [language tag](#), the language of the contents and attribute values of XML elements (see the [IANA Language Subtag Registry](#)). The [XML](#) specification specifies the `xml:lang` attribute and its influence on child nodes.

Authoring Guidelines:

Although [BCP47](#) recommends that language tags be casefolded in a particular way for presentation, case has no meaning in a language tag. As a reminder to authors that user-agents map all language tags to lowercase, all examples in this document use lowercase. See also [folder-based localization](#), which also requires authors to use language tags in lowercase form as the names of folders.

Avoid subtags from the [IANA Language Subtag Registry](#) marked as deprecated, grandfathered, or redundant. The intended purpose of the `xml:lang` attribute is to declare the primary language of an element, its attributes, and its descendant nodes; as such, it has no implication with regards to the directionality of the text in the user agent. To specify the directionality of text, see the [dir](#) attribute.

7.5.2 The `dir` Attribute

A [keyword attribute](#) used to specify the directionality in which human-readable text is to be represented by a user agent (e.g., the text content of the [name](#) element, the [description](#) element, and the [license](#) element). The directionality set by the [dir](#) attribute applies to the text content and any [displayable string](#) attributes of the element where it is used, and to child elements in its content unless overridden with another instance of [dir](#) (i.e., in this specification, the [dir](#) attribute only affects the [short](#) attribute of the [name](#) element and to the [version](#) attribute of the [widget](#) element).

The possible value of a [dir](#) attribute is one of the **valid directional indicators**:

ltr

Left-to-right text. Request that the Unicode [\[BIDI\]](#) algorithm treat characters within an element as embedded left-to-right.

rtl

Right-to-left text. Request that the Unicode [BIDI] algorithm treat characters within an element as embedded right-to-left.

lro

Left-to-right override. Forces the Unicode [BIDI] algorithm to treat characters within an element as strong left-to-right characters.

rlo

Right-to-left override. Forces the Unicode [BIDI] algorithm to treat characters within an element as strong right-to-left characters.

Note:

For security reasons, implementations intending to display [IRIs](#) and [IDNA](#) addresses found in the configuration document are strongly encouraged to follow the security advice given in [\[UTR36\]](#). This could include, for example, behaving as if the [dir](#) attribute had no effect on any [IRI attributes](#), [path attributes](#), and the [author](#) element's [email](#) attribute.

The base direction of a [dir](#) attribute is either set explicitly by the nearest parent element that uses the [dir](#) attribute; or, in the absence of such an attribute, the base direction is inherited from the **default direction of the document**, which is left-to-right ("[ltr](#)").

7.5.3 Examples of Usage

The following example demonstrates the [dir](#) attribute being applied globally to a [configuration document](#).

```
<widget xmlns="http://www.w3.org/ns/widgets" dir="rtl" xml:lang="fa">
  <name short="آب و هوا">
    آب و هوا برنامه
  </name>
  <description>
    این نرم افزار به شما اجازه می دهد تا برای دیدن آب و هوا در منطقه محلی تان.
  </description>
</widget>
```

The following example shows the [dir](#) attribute applied to [localized content](#).

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name short="Weather">
    Weather Application
  </name>
  <name short="آب و هوا" xml:lang="fa" dir="rtl">
    آب و هوا برنامه
  </name>
</widget>
```

The following example shows the [dir](#) attribute used with mixed language content:

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name short="Weather">
    Weather! a totally awesome application!
  </name>
  <name short="آب و هوا" xml:lang="fa" dir="rtl">
    <span dir="ltr" xml:lang="en">Weather!</span> برنامه واقعا بزرگ
  </name>
```

```
</widget>
```

7.6 The widget Element and its Attributes

The [widget](#) element serves as a container for the other elements of the [configuration document](#).

Context in which this element is used:

The [widget](#) element is the root element of a [configuration document](#).

Occurrences:

Exactly one, at the root element of the [\[XML\]](#) document.

Expected children (in any order):

[name](#): zero or more ([one element is allowed per language](#)).

[description](#): zero or more ([one element is allowed per language](#)).

[author](#): zero or one.

[license](#): zero or more ([one element is allowed per language](#)).

[icon](#): zero or more.

[content](#): zero or one.

[feature](#): zero or more.

[preference](#): zero or more.

Localizable via xml:lang:

No. Inheritance of the value of this attribute by [author](#), [preference](#), [icon](#), and [content](#) will have no effect during processing in [Step 7](#).

Attributes:

[Global attributes](#), [id](#), [version](#), [height](#), [width](#), [viewmodes](#).

7.6.1 The id Attribute

An [IRI attribute](#) that denotes an identifier for the widget.

Authoring Guidelines:

It is optional for authors to use the [id](#) attribute with a [widget](#) element.

7.6.2 The version Attribute

A [version attribute](#) that specifies the version of the widget.

Authoring Guidelines:

It is optional for authors to use the [version](#) attribute with a [widget](#) element.

7.6.3 The height Attribute

A [numeric attribute](#) greater than 0 that indicates the preferred [viewport](#) height of the instantiated [custom start file](#) in [CSS pixels \[CSS\]](#).

Authoring Guidelines:

It is optional for authors to use the [height](#) attribute with a [widget](#) element.

7.6.4 The width Attribute

A [numeric attribute](#) greater than 0 that indicates the preferred [viewport](#) width of the instantiated [custom start file](#) in [CSS pixels \[CSS\]](#).

Authoring Guidelines:

It is optional for authors to use the [width](#) attribute with a [widget](#) element.

7.6.5 The `viewmodes` Attribute

A [keyword list attribute](#) that denotes the author's preferred view mode, followed by the next most preferred view mode and so forth. When the attribute is missing, or is left empty, it implies that the author expects the user agent to select an appropriate viewmode for the widget.

The concept of a *viewport* is defined in [\[CSS\]](#), but is essentially a window or other viewing area on the screen (see section [9.1.1 The viewport](#) of [\[CSS\]](#)). The concept of a *view mode* is defined in the [\[View-Modes\]](#) specification.

Authoring Guidelines:

It is optional for authors to use the [viewmodes](#) attribute with a [widget](#) element.

7.6.6 The `defaultlocale` attribute

A [language attribute](#) that specifies, through a [language tag](#), the author's preferred locale for the widget. Its intended use is to provide a fallback in case the user agent cannot match any of the widget's [localized content](#) to the [user agent locales](#) list or in case the author has not provided any unlocalized content.

Authoring Guidelines:

It is optional for authors to use the [defaultlocale](#) attribute with a [widget](#) element.

7.6.7 Example of Usage

The following example shows how the [widget](#) element can be used.

```
<widget xmlns      = "http://www.w3.org/ns/widgets"
      id           = "http://example.org/exampleWidget"
      version      = "2.0 Beta"
      height       = "200"
      width        = "200"
      viewmodes    = "windowed floating"/>
```

7.6.8 Example of Usage of the `defaultlocale` attribute

The following example shows how the [widget](#) element's [defaultlocale](#) attribute can be used:

```
<widget xmlns = "http://www.w3.org/ns/widgets"
      defaultlocale = "en-us">
  <name short="Weather" xml:lang="en-us">
    The Ultimate Weather Widget
  </name>
  <name short="Boletim" xml:lang="pt">
    Boletim Metereológico
  </name>
</widget>
```

7.7 The `name` Element and its Attributes

The [name](#) element represents the full human-readable name for a [widget](#) that is used, for example, in an application menu or in other contexts.

Context in which this element is used:

In a [widget](#) element.

Content model:

Any.

Occurrences:

Zero or more ([one element is allowed per language](#)).

Expected children:

[span](#): zero or more.

Localizable via [xml:lang](#):

Yes.

Authoring Guidelines:

The value of the [xml:lang](#) attribute needs to be unique for any subsequent element of this type.

Attributes:

[Global attributes](#), [short](#).

7.7.1 The short Attribute

A [displayable-string attribute](#) intended to represent a condensed name for a widget (e.g., a name that could be used in context where only limited space is available, such as underneath an icon).

Authoring Guidelines:

It is optional for authors to use the [short](#) attribute with an [name](#) element.

7.7.2 Example of Usage

The following example shows the usage of the [name](#) element.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name short="Weather">
    The Ultimate Weather Widget
  </name>

  <name short="Boletim" xml:lang="pt">
    Boletim Meteorológico
  </name>
</widget>
```

7.8 The description Element and its Attributes

The [description](#) element represents a human-readable description of the widget.

Context in which this element is used:

In a [widget](#) element.

Content model:

Any.

Occurrences:

Zero or more ([one element is allowed per language](#)).

Expected children:

[span](#): zero or more.

Localizable via [xml:lang](#):

Yes.

Authoring Guidelines:

The value of the [xml:lang](#) attribute needs to be unique for any subsequent element of this type. If two or more elements with the same [xml:lang](#) attribute value are encountered, the user agent will

[ignore](#) all but the matching first element. See [Step 7](#) for more details.

Attributes:

[Global attributes](#).

7.8.1 Example of Usage

An example usage of the description element.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name>Dahut Chaser</name>
  <description>
    Combining the latest weather info with your GPS position,
    this widget alerts you of any significant dahut activity in your
    area. When a big one walks by, the widget plots the best route on a map based
    on the dahut's trajectory so you can chase it! With support for
    built-in cameras, you can quickly upload all the Alpine action to
    your blog or to the insane dahut chaser web site! Awesome!
  </description>
</widget>
```

7.9 The author Element and its Attributes

An [author](#) element represents people or an organization attributed with the creation of the widget.

Context in which this element is used:

As a child of the [widget](#) element.

Content model:

Any.

Occurrences:

Zero or one.

Expected children:

[span](#): zero or more.

Localizable via [xml:lang](#):

No. Only the first instance of this element in [document order](#) will be used, regardless of the value of [xml:lang](#) (if any).

Attributes:

[Global attributes](#), [href](#), [email](#).

7.9.1 The href Attribute

An [IRI attribute](#) whose value represents an IRI that the author associates with himself or herself (e.g., a homepage, a profile on a social network, etc.).

Authoring Guidelines:

It is optional for authors to use the [href](#) attribute with an [author](#) element.

7.9.2 The email Attribute

A [string attribute](#) that represents an email address associated with the author.

Authoring Guidelines:

It is optional for authors to use the [email](#) attribute with an [author](#) element.

7.9.3 Example of Usage

The following example shows the expected usage of the [author](#) element.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name>Café Finder</name>
  <author href = "http://dahut.example.org/developers/"
          email = "cafefinder@example.org">
    Mr. Jo and Julia Bacalhau
  </author>
</widget>
```

7.10 The license Element and its Attributes

The [license](#) element represents a **software license**, which includes, for example, a usage agreement, redistribution statement, and/or a copyright license terms under which the content of the widget package is provided.

Context in which this element is used:

As a child of the [widget](#) element.

Content model:

Any.

Expected children:

[span](#): zero or more.

Occurrences:

Zero or more ([one element is allowed per language](#)).

Localizable via [xml:lang](#):

Yes.

Authoring Guidelines:

The value of the [xml:lang](#) attribute needs to be unique for any subsequent element of this type. The content of localized [license](#) elements shouldn't be used to present different versions of a license, just translations of the same license.

Attributes:

[Global attributes](#), [href](#).

7.10.1 The href Attribute

A [valid IRI](#) or a [valid path](#) that points to a representation of a software and/or content license.

Authoring Guidelines:

It is optional for authors to use the [href](#) attribute with a [license](#) element.

7.10.2 Example of Usage

This example shows the expected usage of the [license](#) element's [href](#) attribute.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <license href="http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231">
    Distributed under the W3C Software License.
  </license>
</widget>
```

This example shows the expected usage of the [license](#) element when the [href](#) attribute is omitted.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <license>
    ...
    3.3.1 Widgets can use any APIs or libraries, prescribed by anyone.
    Widgets are a free and open Web technology, so can be produced for free
```



```
and sold anywhere. Widgets can be written in JavaScript
so can run on any conforming engine (without the annoying restrictions of
C, C++, or Objective-C). You can even "cross-compile" them, if you want.
```

```
...
</license>
</widget>
```

7.11 The `icon` Element and its Attributes

The `icon` element represents a [custom icon](#) for the widget.

Context in which this element is used:

As a child of the `widget` element.

Content model:

Empty.

Occurrences:

Zero or more.

Localizable via `xml:lang`:

No. Relies on [folder-based localization](#).

Attributes:

[Global attributes](#), `src`, `width`, `height`.

7.11.1 The `src` Attribute

A [path attribute](#) that points to a [file](#) inside the widget package.

Authoring Guidelines:

When an `icon` element is used, it is required for authors to use the `src` attribute.

7.11.2 The `width` Attribute

A [numeric attribute](#) greater than 0 that represents, in [CSS pixels \[CSS\]](#), the author's preferred width for the icon. A user agent MAY ignore this value when changing the height icon to fit a rendering context or for accessibility reasons.

Authoring Guidelines:

It is optional for authors to use the `width` attribute of an `icon` element.

7.11.3 The `height` Attribute

A [numeric attribute](#) greater than 0 that represents, in [CSS pixels \[CSS\]](#), the author's preferred height for the icon. A user agent MAY ignore this value when changing the height icon to fit a rendering context or for accessibility reasons.

Authoring Guidelines:

It is optional for authors to use the `height` attribute of an `icon` element.

7.11.4 Example of Usage

This example shows the expected usage of the `icon` element. The example declares three icon elements, two of which are raster and one of which is an [\[SVGTiny\]](#) file. The raster graphics would be used for display contexts smaller than 256x256 pixels. [Document order](#) of the elements is irrelevant.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <icon src="icons/medium.png"/>
```

```

<icon src="icons/big.svg" width="256" height="256"/>
<icon src="icons/tiny.png"/>
</widget>

```

7.12 The content Element and its Attributes

The [content](#) element is used by an author to declare which [custom start file](#) the user agent is expected to use when it instantiates the widget.

Context in which this element is used:

As a child of the [widget](#) element.

Content model:

Empty.

Occurrences:

Zero or one.

Localizable via [xml:lang](#):

No. Relies on [folder-based localization](#).

Attributes:

[Global attributes](#), [src](#), [type](#), [encoding](#).

7.12.1 The src Attribute

A [path attribute](#) that allows an author to point to a [file](#) within the [widget package](#).

Authoring Guidelines:

When a [content](#) element is used, it is required for authors to use the [src](#) attribute.

7.12.2 The type Attribute

A [media type attribute](#) that indicates the [media type](#) of the file referenced by the [src](#) attribute.

Authoring Guidelines:

It is optional for authors to use the [type](#) attribute with a [content](#) element. When the value is absent, the user agent will assume the value `text/html`.

7.12.3 The encoding Attribute

A [keyword attribute](#) that denotes the character encoding of the file identified by the [src](#) attribute. The value is the "name" or "alias" of any "Character Set" listed in [\[IANA-Charsets\]](#).

Authoring Guidelines:

It is optional for authors to use the [encoding](#) attribute with a [content](#) element. Where aliases are given by the [\[IANA-Charsets\]](#) registry, authors are encouraged to use the value of the "preferred MIME name" (if any) from the registry.

The **default encoding** is [\[UTF-8\]](#), which has the name "[UTF-8](#)" in the [\[IANA-Charsets\]](#) registry.

Aside from the [default encoding](#), it is OPTIONAL for a [user agent](#) to [support](#) other character encodings.

7.12.4 Example of Usage

This example shows the expected usage of the [content](#) element:

```

<widget xmlns="http://www.w3.org/ns/widgets">
  <content src="myWidget.html"/>

```

```
</widget>
```

This example shows the [content](#) element being used with a `encoding` attribute to override the default value of the `encoding` attribute ([UTF-8](#)) with the GB2312 character set, which the author has used to encode simplified Chinese characters:

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name xml:lang="zh-cn">古老瓷地图</name>
  <name>Ancient Chinese Maps</name>
  <content src="china-maps.html" encoding="GB2312"/>
</widget>
```

This example shows the [content](#) element being used with a `type` attribute to instantiate a widget created with a proprietary [media type](#):

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name>Location-Based Games Finder</name>
  <content src="lbg-maps.swf" type="application/x-shockwave-flash"/>
  <feature name="http://example.org/api.geolocation"
    required="false"/>
</widget>
```

7.13 The feature Element and its Attributes

A **feature** is a URI identifiable runtime component (e.g. an Application Programming Interface or video decoder). The act of an author requesting the availability of a feature through a [feature](#) element is referred to as a **feature request**. The [feature](#) element serves as a standardized means to request the binding of an IRI identifiable runtime component to a widget for use at runtime. Using a [feature](#) element denotes that, at runtime, a widget can attempt to access the [feature](#) identified by the [feature](#) element's `name` attribute. How a user agent makes use of [features](#) depends on the user agent's security policy, hence activation and authorization requirements for features are beyond the scope of this specification. A feature has zero or more [parameters](#) associated with it.

Context in which this element is used:

As a child of the [widget](#) element.

Content model:

Zero or more [param](#) elements.

Occurrences:

Zero or more.

Localizable via `xml:lang`:

No.

Attributes:

[Global attributes](#), [name](#), [required](#).

7.13.1 The name Attribute

An [IRI attribute](#) that identifies a [feature](#) that is needed by the widget at runtime (such as an [API](#)).

Authoring Guidelines:

When the [feature](#) element is declared, it is required for authors to use the [name](#) attribute.

7.13.2 The required Attribute

A [boolean attribute](#) that indicates whether or not this [feature](#) has to be available to the widget at runtime.

When set to `true`, the [required](#) attribute denotes that a [feature](#) is absolutely needed by the widget to function correctly, and without the availability of this [feature](#) the widget serves no useful purpose or won't execute properly.

When set to `false`, the [required](#) attribute denotes that a widget can function correctly without the [feature](#) being [supported](#) or otherwise made available by the user agent.

Authoring Guidelines:

It is optional for authors to use the [required](#) attribute with an [feature](#) element. However, authors need to be aware that a user agent will behave as if the [required](#) attribute had been set to `true` when the [required](#) attribute is absent, meaning that the named [feature](#) needs to be available at runtime or the widget will be treated as an [invalid widget package](#).

7.13.3 Example of Usage

This example demonstrates a widget that would like to use a fictional geo-location API feature, but would still be able to function if the feature cannot be made available to the widget by the user agent.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <feature name      = "http://example.org/api/geolocation"
          required = "false"/>
</widget>
```

7.14 The param Element and its Attributes

The [param](#) element defines a [parameter](#) for a [feature](#). A **parameter** is a name-value pair that is [associated](#) with the corresponding [feature](#) for which the parameter is declared for. A author establishes the relationship between a [parameter](#) and feature by having a [param](#) element as a direct child of a [feature](#) element in [document order](#).

Context in which this element is used:

As a child of the [feature](#) element.

Content model:

Empty.

Occurrences:

Zero or more.

Localizable via [xml:lang](#):

No.

Attributes:

[Global attributes](#), [name](#), [value](#).

7.14.1 The name Attribute

A [string attribute](#) that denotes the name of this [parameter](#).

Authoring Guideline:

When a [param](#) element is declared, it is required for authors to use the [name](#) attribute.

7.14.2 The value Attribute

A [string attribute](#) that denotes the value of this [parameter](#).

Authoring Guidelines:

When a [param](#) element is used, it is required for authors to use the [value](#) attribute.

7.14.3 Example of Usage

This example demonstrates a widget that makes use of a fictional geo-location feature where by its accuracy is set to "low" via a [param](#) element.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <feature name="http://example.org/api/geolocation">
    <param name="accuracy" value="low"/>
  </feature>
</widget>
```

7.15 The preference Element and its Attributes

The [preference](#) element allows authors to declare one or more preferences: a **preference** is a persistently stored name-value pair that is [associated](#) with the widget the first time the widget is initiated.

Note:

A user agent that [supports](#) the [\[Widgets-APIs\]](#) specification will expose any declared [preference](#) at runtime in the manner described in the [\[Widgets-APIs\]](#) specification.

Context in which this element is used:

As a child of the [widget](#) element.

Occurrences:

Zero or more.

Expected children:

none.

Localizable via [xml:lang](#):

No.

Attributes:

[Global attributes](#), [name](#), [value](#), [readonly](#).

7.15.1 The name Attribute

A string that denotes the name of this [preference](#).

Authoring Guidelines:

When a [preference](#) element is used, it is required for authors to use the [name](#) attribute.

7.15.2 The value Attribute

A string that denotes the value of this [preference](#).

Authoring Guidelines:

It is optional for authors to use the [value](#) attribute with a [preference](#) element.

7.15.3 The readonly Attribute

A [boolean attribute](#) indicating whether this preference can, or cannot, be overwritten at runtime (e.g., via an author script). When set to `true`, it means that the [preference](#) cannot be overwritten. When set to `false`, it means that the [preference](#) can be overwritten.

Authoring Guidelines:

It is optional for authors to use the `readonly` attribute with a [preference](#) element. If the `readonly` attribute is absent, the user agent will act as if the `readonly` attribute had been set to `false` (meaning that the [preference](#) can be overwritten at runtime).

7.15.4 Example of Usage

This example shows a widget where two preferences are set. The second [preference](#) is set as "read only" via the `readonly` attribute, which means the values of that preference cannot be changed at runtime.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <preference name      = "skin"
             value     = "alien"/>

  <!-- The preference below will be protected
       from modification and deletion at runtime -->

  <preference name      = "api-key"
             value     = "f6d3a312f9d742"
             readonly  = "true"/>
</widget>
```

7.16 The `span` Element and its Attributes

The [span](#) element is a wrapper for text content; on its own it serves no useful function.

It is expected that authors will use the [span](#) element with the [global attributes](#). When combined with the [dir](#) attribute, the [span](#) element can indicate the textual directionality of text content. In other words, it allows authors to set the base direction for the Unicode bidirectional algorithm [BIDI]. When combined with the `xml:lang` attribute, the [span](#) element allows the author to indicate the particular language used for a subset of text content within another element.

Context in which this element is used:

As a child of the [name](#), [author](#), [license](#), and/or [description](#) element.

Occurrences:

Zero or more.

Expected children:

Any.

Localizable via `xml:lang`:

No, meaning that declaring `xml:lang` on a [span](#) element within a parent element (e.g., a [name](#) element) will not affect the behavior of [element-based localization](#) (see Example of Usage below to see how this works).

Authoring Guidelines:

Authors are encouraged to use `xml:lang` to indicate the language information for text content contained within a [span](#) element. See Example of Usage for more information.

Attributes:

[Global attributes](#).

7.16.1 Example of Usage

This section is informative.

This example shows the [span](#) elements being used to indicate directionality of text as well as language information. Note that the [name](#) element's `xml:lang` attribute is set to an empty string to allow it to be used as [default content](#) in the process of [element-based localization](#):

```
<widget
  xmlns="http://www.w3.org/ns/widgets"
  xml:lang="he" dir="rtl">
  <name xml:lang="" dir="ltr">
    <span xml:lang="en">GPS Weather!</span>
  </name>
  <description>
    הייטשומן ה-
    <span dir="ltr" xml:lang="en">GPS Weather!</span> מאפסר
```

```

לך לבדוק את מזג הא
ברחבי העולם GPS וויר בכל נקודה.
</description>

<description xml:lang="" dir="ltr">
  <span xml:lang="en">The GPS Weather! widget lets you check
    the weather at any point around the world with GPS.</span>
</description>
</widget>

```

8 Internationalization and localization

Internationalization, or i18n, is the design and development of a product, application or document content that enables localization for target audiences that vary in culture, region, or language. **Localization** refers to the adaptation of a product, application or document content to meet the language, cultural and other requirements of a specific target market (a "locale").

Note:

See also the [Web Services Internationalization Usage Scenarios](#) and the [Unicode Locale Data Markup Language](#) for an informative discussion on the term locale.

Localized content is content an author has explicitly [localized](#): that is, a [widget package](#) that contains content localized using [folder-based localization](#) or a [configuration document](#) that contains content localized via [element-based localization](#).

Default content is content included in the [widget package](#) or in the [configuration document](#) that has not been explicitly localized or content explicitly indicated by the author to be used as default content via the `defaultLocale` attribute of the [widget](#) element. In the case of a [widget package](#), this means content outside the [container for localized content](#). In the case of a [configuration document](#), this means any element without an explicitly declared or inherited `xml:lang` attribute.

A **localized file** is any [file](#) that has been placed inside a [locale folder](#) (i.e., [localized content](#) that makes use of [folder-based localization](#)). A widget package contains zero or more localized files. All files and folders, except for the [digital signatures](#) and the [configuration document](#), can be localized via [folder-based localization](#).

For example, a locale folder "locales/ja/" (Japanese) might contain an HTML document translated into Japanese.

Authoring Guidelines:

[Locale folders](#) need to be placed in the [container for localized content](#) (a locale folder not in a [container for localized content](#) will be treated as an [arbitrary](#) folder). In addition, authors making use of localization features provided by this specification should translate, localize, or alter [localized content](#) for a the given locale, and test their widgets thoroughly.

8.1 Bidirectional text

In conjunction to mandating that user agents support [Unicode](#), this specification provides the [dir](#) attribute and [span](#) element as a markup-based means of influencing the directionality for bidirectional Unicode text [\[BIDI\]](#). See the definitions of the [dir](#) attribute and [span](#) element for usage examples.

8.2 Localization Model

This specification provides two means that authors can use to explicitly localize the content of a widget:

1. By placing localized file content in [locale folders](#), a process referred to as [folder-based localization](#).

- By explicitly marking XML elements in the [configuration document](#) as localized, a process referred to as [element-based localization](#).

Both forms of localization are described below. Folder-based and element-based localization rely on the user agent to match the language ranges held by the [user agent locales](#) to the appropriate locale folders and/or localized XML elements in the widget's [configuration document](#).

8.3 Folder-based localization

This specification defines the concept of **folder-based localization**, which is the process whereby an author places [files](#) inside [folders](#) that are named in a manner that conforms to a language-range ABNF rule of this specification. That is, by naming folders in lower-case using values derived from the [IANA Language Subtag Registry](#) such as "en-us", "en-gb", and so on, but avoiding subtags marked as *deprecated*, *grandfathered*, or *redundant* in the [IANA Language Subtag Registry](#). These [locale folders](#) are then placed inside the [container for localized content](#).

Authoring Guidelines

Although BCP 47 recommends a particular case-folding convention, the use of upper or lowercase letters has no meaning in a language tag. Because folders inside a widget package are treated by the user-agent in a case-sensitive manner, the names of the folders inside a 'locale' folder must be all lowercase. All language tags are mapped to lowercase for matching purposes (although they can appear in any form in the configuration file or elsewhere).

The **container for localized content** is a [reserved folder](#) at the [root of the widget package](#) whose [folder-name](#) case-sensitively matches the string 'locales'. A container for localized content contains zero or more [locale folders](#).

A **locale folder** is a [folder](#) whose [file name field](#) matches the production of [locale-folder](#) and is a direct descendant of the [container for localized content](#) (e.g., "/locales/en-us", "/locales/fr", etc). A [locale folder](#) contains zero or more [arbitrary](#) folders and/or files.

Authoring Guidelines

Authors need to avoid region, script or other subtags except where they add useful distinguishing information to a [locale folder](#). In addition, avoid including empty [locale folders](#) in a [widget package](#) (unless there is a good reason to include them).

An example of a widget that uses folder-based localization:

```

widget.wgt
  locales/zh-hans-cn/a.gif
  locales/zh-hans-cn/f.gif
  locales/zh-hans/a.gif
  locales/zh-hans/b.gif
  locales/zh/a.gif
  locales/zh/b.gif
  locales/zh/c.gif
  a.gif
  b.gif
  c.gif
  d.gif
  index.html
  config.xml

```

Authors can further facilitate the localization process by grouping [files](#) into [folder](#) hierarchies made up of matching subtags, as is shown in the example.

Assuming the widget's locale is "zh-hans-cn", a reference to:

- a.gif would resolve to locales/zh-hans-cn/a.gif
- b.gif would resolve to locales/zh-hans/b.gif
- c.gif would resolve to locales/zh/c.gif
- d.gif would resolve to d.gif, as it is not associated with any locales and is hence available to all locales.

This works at all sub-levels, so long as the parent subtag matches the child subtags. So, for example, the "cn" region can make use of the localized files in the "zh-hans" folder level, the "zh" folder level, and the unlocalized files at the [root of the widget package](#). The user agent always prioritizes files in sub-folders over files in locale folders closer to the [root of the widget package](#). Conversely, if the widget's locale were "en-us", references to a.gif, b.gif, c.gif and d.gif would all resolve to the files in the root of the widget package.

Note also that the user agent treats any [file](#) or [folder](#) outside the container for localized content as [default content](#).

8.4 Element-Based Localization

This specification defines the concept of **element-based localization**, which allows authors to use the `xml:lang` attribute to explicitly indicate that an [\[XML\]](#) element in the [configuration document](#) has been [localized](#).

The following is an example of element-based localization:

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name short="Weather">
    The Ultimate Weather Widget
  </name>

  <name short="Boletim" xml:lang="pt">
    Boletim Metereológico
  </name>
</widget>
```

Some of the elements in the widgets namespace are defined to be **localizable via `xml:lang`** as part of the element's definition (with either "yes" or "no"). See, for example, the [name](#) element. When "yes", it means that an author can utilize `xml:lang` to achieve [element-based localization](#) either directly or indirectly through the inheritance of the value of `xml:lang`. How element-based localization is handled is specified in detail in [Step 7](#).

Note:

The `xml:lang` attribute can be used on any element in order to indicate which language is used in the content and attribute values of that element. As specified in the XML Specification, its value is inherited, such that if an element has an `xml:lang` attribute, all of its descendants are considered to be in that language as well, unless they specify their own `xml:lang` attribute. Note that an element can indicate that it is in no specific language by setting `xml:lang` to the empty string, irrespective of whether any of its ancestors has an `xml:lang` attribute.

For example, if an author uses the `xml:lang` attribute on the [widget](#) element, then all child elements inherit the value `xml:lang`. This means that the first [name](#) element below behaves as `xml:lang="en"` had been explicitly used. However, in the second [name](#) element, the declaration of `xml:lang="pt"` overrides `xml:lang="en"` inherited from the [widget](#) element. Finally, the last name element overrides `xml:lang` with an empty string, so that element will be treated by the user agent as [default content](#).

```
<widget xmlns="http://www.w3.org/ns/widgets"
  xml:lang="en">

  <name short="I'm in english, though not explicitly marked as such!">
    Behaves as if xml:lang="en"
  </name>

  <name xml:lang="pt">
    The declaration of xml:lang="pt" overrides
    xml:lang="en" inherited from the widget element.
  </name>

  <name xml:lang="">
    The user agent will treat this as unlocalized content.
  </name>
```

```
</widget>
```

If an element is marked as being localizable via [xml:lang](#) with "yes", the specification indicates that only **one element is allowed per language**:

One element is allowed per language means that only one element of a type is allowed to be used per language (e.g., although many [name](#) elements can be present in a [configuration document](#), only one [name](#) element will be selected by the user agent for the English language). During processing ([Step 7](#)), the user agent will only match the first element, in [document order](#), that matches a language range in the [user agent locales](#) and [ignore](#) any subsequent repetitions of the element that contain a matching [xml:lang](#) value (even if that element's content is different).

For example, assume the [user agent locales](#) only contains the following language range: "en-us" (English as used in the United States). As only one instance of the [description](#) element is allowed per language, in the following code the user agent would match the first [description](#) element but would [ignore](#) the second and third [description](#) elements.

```
<widget xmlns="http://www.w3.org/ns/widgets">

  <description xml:lang="en">
    This element would be used.
  </description>

  <description xml:lang="en">
    This element would be ignored because there is already
    a description element that uses xml:lang="en".
  </description>

  <description>
    This element is unlocalized, and would be used if the user agent's
    locale does not match any localized description elements.
  </description>

  <description xml:lang="">
    This element would be ignored because there is already an unlocalized
    description element! Using xml:lang="" makes this element behave as if
    it is unlocalized.
  </description>

</widget>
```

However, if the [user agent locales](#) only contained "*", or did not match any of the localized [description](#) elements, then the user agent would match the third [description](#) element above.

In the case whereby the author does not use an [xml:lang](#) attribute, and no element of a particular type with [xml:lang](#) matches the [user agent locales](#), the user agent will use the first element that is [default content](#), in [document order](#), that matches the element type being sought.

For example, now assume that the [user agent locales](#) only contains the following language range: "jp" (Japanese). As only one instance of the [description](#) element is allowed per language, in the following code the user will agent [ignore](#) the first two [description](#) elements, but would match the third (unlocalized) [description](#) element.

```
<widget xmlns="http://www.w3.org/ns/widgets">

  <description xml:lang="en">
    This element would be ignored.
  </description>

  <description xml:lang="en">
    This element would be ignored.
  </description>
```

```

</description>

<description>
  In this case, this unlocalized element would be used.
</description>

</widget>

```

8.5 Localization Examples

This section is non-normative.

This section presents three examples of how widgets can be localized. Each example is intended to showcase how the localization algorithm is intended to work.

8.5.1 Simple Example

This example shows a widget that displays the days of the week based on the language ranges held by the [user agent locales](#). If the user agent is unable to match a language range to any [locale folder](#), the widget displays /index.html at the [root of the widget package](#).

 config.xml

```

<widget xmlns="http://www.w3.org/ns/widgets">
  <name>What day is it?</name>
  <description>
    This widget highlights the current day
    of the week.
  </description>
</widget>


```
















 locale/es/index.html

```

<!doctype html>
<title>¿Qué día es?</title>
<script src="scripts/dayfinder.js"></script>
<body>
<p>Hoy es: lunes, martes, miércoles,
  jueves, viernes, sábado,
  domingo

```

 weekdays.wgt

-  index.html /*English*/
-  config.xml
-  scripts
 -  dayfinder.js
-  locales
 -  fr/
 -  index.html /*French*/
 -  es/
 -  index.html /*Spanish*/
 -  pt/
 -  index.html /*Portuguese*/
 -  fi/
 -  index.html /*Finnish*/
 -  it/
 -  index.html /*Italian*/


8.5.2 Complex Example

The following is an example of a localized widget with multiple localized [icons](#), [start files](#) and [configuration documents](#). Some relevant things to note from the example:

- The Spanish version (`locales/es/`) has its own localized icon.
- The Spanish (`locales/es`) and English (`locales/en/`) versions of the widget use the un-localized script in `/scripts/engine.js` folder.
- On the other hand, the English-Australia (`en-au`) version uses a localized script (`en-au/scripts/engine.js`).

 /config.xml

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name xml:lang="ko">웃기는 고양이</name>
  <content src="cats.html"/>
</widget>
```

 /locales/en-au/cats.html

```
<!doctype html>
<title>G'day! LOL Cats!</title>
<script src="scripts/engine.js">
...

```

 /locales/es/cats.html

```
<!doctype html>
<title>Gatos Graciosos!</title>
<script src="scripts/engine.js">
...

```

```

🔧 LOLcats.wgt
  📄 cats.html /*written in Korean*/
  📄 config.xml
  🖼 icon.png /*default icon*/
  🛡 signature.xml /*not localizable*/
  ▼ 📁 scripts/
    📄 engine.js /*default functionality*/
  ▼ 📁 locales/ /*container for localized content*/
    📁 en-au/
      📄 cats.html
      📁 scripts/
        📄 engine.js /*custom localized functionality*/
    📁 en/
      📄 cats.html /*international english version*/
    📁 es/
      📄 cats.html /*Spanish version*/
      🖼 icon.png /*localized icon*/

```

8.5.3 Fallback Behavior Example

This specification allows authors to place files and folders they don't wish to localize at the root of the widget package. At runtime, if the user agent fails to find a file in a [locale folder](#), it will always search at the root of the widget package for that missing file. The purpose of this 'fallback' model is to reduce the number of [files](#) that need to be created in order to [localize a widget package](#).

The example below demonstrates how a user agent attempts to locate a file that is absent in a localized scenario. Assume the user agent's locale is 'en-gb' and the zip relative path being sought is "images/mast.png":

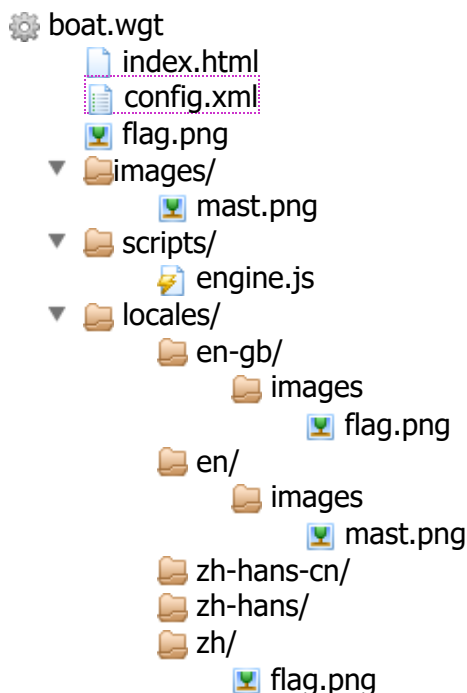
1. Firstly, the user agent will search for a folder that matches the user agent's locale as closely as possible for the desired file. In this case, the user agent would attempt to locate the relative path 'images/mast.png' in '/locales/en-gb/', but would fail.
2. Secondly, if the file is absent, the user agent will search for the absent file in any other subfolder that is in the language range of the current locale folder. So the next place the user agent would look is in the 'en/' folder, where it would match the zip relative path 'images/mast.png'.
3. Lastly, if the above fails, the user agent will search for the absent file at the root of the widget package. So, if the user agent's locale did not find the zip relative path in one of the locale folders, then 'images/mast.png' file at the root of the widget would be matched and this [default content](#) would be used.

Now consider the for various Chinese variants: 'zh-hans-cn', 'zh-hans', and 'zh' below. In this case, to find the 'flag.png' file for Mainland Chinese in simplified script 'zh-hans-cn', the user agent would first look in 'zh-hans-cn', followed by 'zh-hans', then in 'zh' where the file is located.

To find the 'mast.png' file, the user agent would look in 'zh-hans-cn', followed by 'zh-hans', followed 'zh', and finally at the root of the widget package where the absent file is actually located.

 /index.html

```
<!doctype html>
<title>Patriotic Boat</title>
<script src="scripts/engine.js">
</script>
<body>
  
  
```



9 Steps for Processing a Widget Package

The **steps for processing a widget package** involves nine steps that a [user agent](#) follows in sequential order, responding accordingly if any of the steps result in an error or if the specification asks for the user agent to skip a step. The procedures for what to do when an error is encountered are described as part of each step; however, there are times when it will not be possible for the user agent to recover from an error and it will be forced to treat the widget as an [invalid widget package](#).

In the event that a user agent encounters an [invalid widget package](#) during the [steps for processing a widget package](#), a [user agent](#) MUST abort all processing of the [steps for processing a widget package](#).

Note:

A user agent can optimize [steps for processing a widget package](#) and associated [processing rules](#), or perform the steps in a different order, but the end result needs to be indistinguishable from the result that would be obtained by following the specification.

9.1 Processing Rules

This section defines various **processing rules**, which are algorithms used during the [steps for processing a widget package](#).

These algorithm makes use of a few special concepts defined below:

Text node

Any Text node, including CDATASection nodes (any Node with node type TEXT_NODE or CDATA_SECTION_NODE) as defined in the [DOMCore](#) specification. For example, the worlds "hello world!" in the following [name](#) element: `<name>hello world!</name>`.

Localizable string

A data structure containing a sequence of one or more strings, each having some associated directional information and language information (if any). The purpose of an localizable string is to assist user agent in correctly applying the Unicode [BIDI](#) algorithm when displaying text.

For example, the string "Internationalization [نشاط التدويل](#) Activity." The string contains both Arabic and English that mixes left-to-right text, right-to-left text, and a directionality-neutral punctuation mark that could not be correctly displayed without directional information.

null

A special symbol to indicate that a value has no data. For example, "let *x* be [null](#)" or "if *y* is empty, then return [null](#)".

Note:

Note: Although languages such as ECMA Script and Java support [null](#) as a native value type, there are some programming languages that have no notion of [null](#) or where null is problematic (e.g. C++). Implementations in these languages need to substitute a language-specific value or symbol which is functionally equivalent to [null](#), or if no equivalent exists, to have no value at all. For example, the value 0 may represent [null](#) for the height of a widget, since the height of a widget is defined as a non-negative integer greater than 0. In such a case, 0 could be treated as if it were [null](#).

9.1.1 Rule for Verifying a Zip Archive

The **rule for verifying a zip archive** is described in this section. The algorithm returns either true or an error.

This specification does not provide the technical details of how to actually perform the checks, for which implementers need to refer to the [ZIP](#) specification.

1. If the [Zip archive](#) is *split into multiple files* or spans *multiple volumes*, as defined in the [ZIP](#) specification, then return an error and terminate this algorithm.
2. If the [Zip archive](#) is encrypted, as defined in [Zip](#), return an error and terminate this algorithm.
3. Otherwise, return true.

9.1.2 Rule for Extracting File Data from a File Entry

The rule for extracting file data from a file entry is as follows:

1. Let *path* be the [zip relative path](#) that identifies the [file entry](#) being sought.
2. Let [file entry](#) be the file entry identified by the *path*.
3. Let [file](#) be the result of decompressing (or extracting) the [file data](#) from [file entry](#) using [\[Zip\]](#).
4. Return [file](#).

Note:

For efficiency, a user agent can extract specific [files](#) as they are needed for processing rather than extracting all the [files](#) at once. As a security precaution, implementations are discouraged from extracting [file entries](#) from untrusted widgets directly onto the file system. Instead, implementations could use, for example, a virtual file system or mapping to access files inside a [widget package](#).

9.1.3 Rule for Finding a File Within a Widget Package

The rule for finding a file within a widget package is given in the following algorithm. The algorithm returns either a [processable file](#), [null](#), or an error.

For the sake of comparison and matching, it is RECOMMENDED that a [user agent](#) treat all [Zip relative paths](#) as [\[UTF-8\]](#).

Note:

This specification does not define how links in documents other than the [configuration document](#) are to be dereferenced. For handling links in other documents, such as (X)HTML, CSS, SVG, etc., please refer to the [\[Widgets-URI\]](#) specification.

1. Let *path* be the path to the [file entry](#) being sought by the user agent.
2. If *path* is not a valid path, return an error and terminate this algorithm.
3. If the *path* starts with a U+002F SOLIDUS (e.g., `"/style/master.css"`), then remove the first U+002F SOLIDUS from *path*.
4. Let *path-components* be the result of splitting *path* at each occurrence of a U+002F SOLIDUS character, removing that U+002F SOLIDUS character in the process.
5. if the first item in *path-components* case-sensitively matches the string "locales", then:
 - A. If the *path-components* does not contain a second item, then return [null](#).
 - B. If the second item in *path-components* is not a valid language-range, then return [null](#) and terminate this algorithm.
 - C. Otherwise, continue.
6. For each *lang-range* in the [user agent locales](#):
 - A. Let *path* be the concatenation of the string "locales/", the *lang-range*, a U+002F SOLIDUS character, and the *path* (e.g., `locales/en-us/cats.png`, where "en-us" is the *lang-range* and "cats.png" is the *path*).
 - B. If *path* case-sensitively matches the [file name field](#) of a [file entry](#) within the [widget package](#) that is a [folder](#), then return an error and terminate this algorithm.
 - C. If *path* case-sensitively matches the [file name field](#) of a [file entry](#) within the [widget package](#) that is a [file](#), let [file](#) be the result of applying the [rule for extracting file data from a file entry](#) to *path*.

- D. If *file* is a [processable file](#), then return *file* and terminate this algorithm.
 - E. If the *path* points to a [file entry](#) that is not a [processable file](#), then return an error and terminate this algorithm.
7. If every *lang-range* in the [user agent locales](#) have been searched, then search for a [file entry](#) whose [file name field](#) matches *path* from the [root of the widget package](#):
- A. If *path* points to a [file entry](#) within the [widget package](#) that is a [folder](#), then return an error and terminate this algorithm.
 - B. If *path* points to a [file entry](#) within the [widget package](#) that is a [file](#), let *file* be the result of applying the [rule for extracting file data from a file entry](#) to *path*.
 - C. If *file* is a [processable file](#), then return *file* and terminate this algorithm.
 - D. If the *path* points to a [file entry](#) that is not a [processable file](#), then return an error and terminate this algorithm.
8. Otherwise, return [null](#).

9.1.4 Rule for Determining Directionality

The **rule for determining directionality** is given in the following algorithm. The algorithm always returns one of the [valid directional indicators](#) as a string.

1. Let *element* be the element to be processed.
2. If *element* is the root element of the [configuration document](#):
 - A. If it does not contain a [dir](#) attribute, return "[ltr](#)" and terminate this algorithm.
 - B. If it does contain a [dir](#) attribute, let *value* be value of the [dir](#) attribute of *element*.
 - C. Remove any leading or trailing [space characters](#) from *value*.
 - D. If *value* of the attribute case-sensitively matches one of the [valid directional indicators](#), return the value of the attribute and terminate this algorithm. if the value does not case-sensitively match one of the [valid directional indicators](#), return "[ltr](#)" and terminate this algorithm.
3. If *element* does not contain a [dir](#) attribute, recursively apply [rule for determining directionality](#) to the direct parent element of *element* and return the result.
4. If *element* contains a [dir](#) attribute, let *direction* be the result of applying the [rule for getting a single attribute value](#) to the [dir](#) attribute of *element*:
 - A. If *direction* case-sensitively matches one of the [valid directional indicators](#), return *direction* and terminate this algorithm.
 - B. If *direction* did not case-sensitively match one of the [valid directional indicators](#), apply the [rule for determining directionality](#) to the direct parent element of *element* and return the result.

9.1.5 Rule for Getting a Single Attribute Value

The **rule for getting a single attribute value** is given in the following algorithm. The algorithm always returns either a string or a [localizable string](#), which can be empty.

1. Let *attribute* be the attribute to be processed.
2. Let *value* be the value of the attribute to be processed.

3. In *value*, replace any sequences of [space characters](#) (in any order) with a single U+0020 SPACE character.
4. Remove any leading or trailing U+0020 SPACE characters from *value*.
5. If the attribute is not a [displayable-string attribute](#), then let *result* be a string that contains the value of *value*.
6. Otherwise, if and only if the attribute is a [displayable-string attribute](#):
 - A. Let *result* be a [localizable string](#) that contains the value of *value*.
 - B. Let *element* be the element that owns *attribute*.
 - C. Let *direction* be the result of applying the [rule for determining directionality](#) to *element*.
 - D. Associate *direction* with *result*.
 - E. Let *lang* be the [language tag](#) derived from having processed the [xml:lang](#) attribute on either *element*, or in *element*'s ancestor chain as per [\[XML\]](#). If [xml:lang](#) was not used anywhere in the ancestor chain, then let *lang* be an empty string.
 - F. Associate *lang* with *result*.
7. Return *result*.

9.1.6 Rule for Getting a List of Keywords From an Attribute

The **rule for getting a list of keywords from an attribute** is given by the following algorithm. The algorithm takes a string as input, and returns a list of strings which can be empty.

1. Let *result* be the value of the attribute to be processed.
2. In *result*, replace any sequences of [space characters](#) (in any order) with a single U+0020 SPACE character.
3. In *result*, remove any leading or trailing U+0020 SPACE character.
4. In *result*, split the string at each occurrence of a U+0020 character, removing that U+0020 character in the process.
5. Return *result*.

9.1.7 Rule for Verifying a File Entry

The **rule for verifying a file entry** is given in the following algorithm. The algorithm always returns a boolean value.

For the [file entry](#), check the following data in the [local file header](#).

1. If the value of the **CRC-32** field (defined in the [\[ZIP\]](#) specification) fails a CRC-32 check, return `false` and terminate this algorithm.
2. The [file name field](#) is an empty string, return `false` and terminate this algorithm.
3. The [file name field](#) contains [Zip forbidden characters](#), return `false` and terminate this algorithm.
4. The [file name field](#) is a sequence exclusively composed of (one or more) [space characters](#) or a mixed sequence of [space characters](#) and U+002E FULL STOP (".") (e.g. ". ."), return `false` and terminate this algorithm.

5. The *file name field* is an invalid [Zip relative path](#), return false and terminate this algorithm.
6. return true.

9.1.8 Rule for Getting Text Content

The **rule for getting text content** is given in the following algorithm. The algorithm always returns a list of [localizable strings](#), which can be empty.

1. Let *input* be the `Element` to be processed.
2. Let *bidi-text* be an empty list of [localizable strings](#).
3. If *input* has no child nodes, return an *bidi-text* and terminate this algorithm.
4. Let *lang-strings* be an empty list (it will hold localizable strings).
5. Let *lang* be the [language tag](#) derived from having processed the `xml:lang` attribute on either *input*, or in *input*'s ancestor chain as per [\[XML\]](#). If `xml:lang` was not used anywhere in the ancestor chain, then let *lang* be an empty string.
6. Let *direction* be the result of applying the [rule for determining directionality](#) to *input*.
7. Associate *lang* and *direction* with *bidi-text*.
8. For each *child node* of *input*:
 - A. If *child node* is not an `Element` or `TextNode`, move onto the next child element and ignore the following sub-steps.
 - B. If the *child node* is an `Element`, let *lstring* be the result of recursively applying the [rule for getting text content](#) using this *current child element* as the argument.
 - C. If the *child node* is a [text node](#), then:
 - A. Create a new localizable string called *lstring*, using the text content of the *current child node* as the text value, *direction* as the direction, and *lang* as the language.
 - D. Append *lstring* to the *lang-string* list.
9. Take all the text nodes, in order, from the returned [localizable string](#) and associate them with *bidi-text*.
10. return *bidi-text*.

For example, the following configuration document:

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns="http://www.w3.org/ns/widgets" dir="rtl" xml:lang="fr">
  <name dir="rlo">
    <span xml:lang="jp">Hello <span dir="rtl"><span dir="rlo"/>backwards</span></span>
  </name>
</widget>
```

When the [rule for getting text content](#) is applied could be logically represented as pseudo code:

```
{dir: rlo,
 lang: fr,
 textNodes:
   [{data: "Hello ",
    direction: "rlo",
    lang: "jp"}],
```

```

    {data: "backwards",
      direction: "rtl",
      lang: "jp"]}
  }

```

9.1.9 Rule for Getting Text Content with Normalized White Space

The **rule for getting text content with normalized white space** is given in the following algorithm. The algorithm always returns a string, which can be empty.

1. Let *input* be the `Element` to be processed.
2. Let *result* be the result of applying the [rule for getting text content](#) to *input*.
3. In *result*, convert any sequence of one or more [space characters](#) into a single U+0020 SPACE.
4. In *result*, remove any leading or trailing U+0020 SPACE character.
5. Return *result*.

For example, the user agent would [ignore](#) the `author` and `blink` elements, but their Text nodes would be extracted (together with directional and language information):

```

<widget xmlns="http://www.w3.org/ns/widgets"
        xmlns:x="http://x.x.example/x" xml:lang="en">
  <name>
    The <blink xml:lang="en-us">Awesome</blink>
    <author email="dude@example.com">Super <x:blink dir="rtl">Dude</x:blink></author>
    Widget</name>
</widget>

```

The resulting [widget name](#) would be "The Awesome Super eduD Widget" (please note that "Dude" is rendered right-to-left).

9.1.10 Rule for Parsing a Non-negative Integer

The **rule for parsing a non-negative integer** is given in the following algorithm. This algorithm returns the number zero, a positive integer, or an error.

1. Let *input* be the string being parsed.
2. Let *result* have the value 0.
3. If the length of *input* is 0, return an error.
4. Let *position* be a pointer into *input*, initially pointing at first character of the string.
5. Let *nextchar* be the character in *input* at *position*.
6. If the *nextchar* is one of the [space characters](#), increment *position*. If *position* is past the end of *input*, return an error and terminate this algorithm. Otherwise, go to step 5 in this algorithm.
7. If the *nextchar* is not one of U+0030 (0) .. U+0039 (9), then return *result*.
8. If the *nextchar* is one of U+0030 (0) .. U+0039 (9):
 - A. Multiply *result* by ten.
 - B. Add the value of the *nextchar* to *result*.
 - C. Increment *position*.

D. If *position* is not past the end of *input*, go to 5 in this algorithm.

9. Return *result*.

9.1.11 Rule for Identifying the Media Type of a File

The **rule for identifying the media type of a file** is given by the following algorithm. The algorithm always returns a string.

Note:

This rule is only to be applied when explicitly instructed to by the specification (e.g., during [Step 7](#)). There are situations where alternative means are defined by the specification to identify the media type of a file (e.g., by the presence of the [content](#) element's [type](#) attribute or deriving the media type of a [default start file](#) from the [default start files table](#)).

1. Let *file* be the [file](#) to be processed.
2. Let *content-type* be an empty string.
3. Let *extension* be an empty string.
4. Let *name* be the [file-name](#) component of the [zip relative path](#) that identifies the *file*.

||For example, the *name* for "some/zip/re1/path/hello.png" would be "hello.png".

5. If the first character of the *name* is a U+002E 'FULL STOP' character, and the file name contains no other U+002E 'FULL STOP' character, then go to step 10 of this algorithm.

||For example, if the name is ".htaccess", jump to step 10 and derive the media type using the [\[SNIFF\]](#) specification.

6. If the first character or last character of the *name* is not a U+002E 'FULL STOP' character, but *name* contains one or more U+002E 'FULL STOP' characters, then let *extension* be the sequence of characters from the last U+002E 'FULL STOP' (inclusive) to the end of *name*.

||The value of *extension* for the file name "cat.html" would be ".html". The value of *extension* for "...html" would be ".html". And the value of *extension* for "hello." would be an empty string.

7. If the first character of the *name* is a U+002E 'FULL STOP' character, and the file name contains another U+002E 'FULL STOP' character, then let *extension* be the sequence of characters from the last U+002E 'FULL STOP' (inclusive) to the end of *name* (if any).

||For example, if the *name* is ".myhidden.html", then the extension would be ".html".

8. If *extension* is an empty string, go to step 10 in this algorithm.
9. Check that each character in the *extension* is either in the U+0041-U+005A range or in the U+0061-U+007A range (ASCII uppercase and lowercase characters, respectively) or in the U+0030-U+0039 range (ASCII numbers 0 to 9):
 - A. If any character in the *extension* is outside the U+0041-U+005A range or the U+0061-U+007A range or the U+0030-U+0039 range, then go to step 10 in this algorithm.

||For example, if the extension is ".png", the go to step 10 in this algorithm.

- B. If all characters in the *extension* are in any of the U+0041-U+005A range or in the U+0061-U+007A range or the U+0030-U+0039 range (e.g., "mp3"), then attempt to case-insensitive match the value of *extension* to one of the values in the file extension column in the [file identification table](#). If there is a match, then return let *content-type* be the corresponding value from the [media type](#) column.

C. Go to step 11.

10. Let *content-type* be the result of processing *file* through the [SNIFF] specification.

11. Return the value of *content-type*.

File Identification Table

| file extension | media type |
|----------------|--------------------------|
| .html | text/html |
| .htm | text/html |
| .css | text/css |
| .js | application/javascript |
| .xml | application/xml |
| .txt | text/plain |
| .wav | audio/x-wav |
| .xhtml | application/xhtml+xml |
| .xht | application/xhtml+xml |
| .gif | image/gif |
| .png | image/png |
| .ico | image/vnd.microsoft.icon |
| .svg | image/svg+xml |
| .jpg | image/jpeg |
| .mp3 | audio/mpeg |

It is OPTIONAL for a [user agent](#) to [support](#) the [media types](#) given in the [file identification table](#).

9.1.12 Rule for Deriving the *user agent locales*

The rule for deriving the *user agent locales* is as follows:

1. Let *unprocessed locales list* be a list that contains the [end-user's language ranges](#).
2. For each *range* in the *unprocessed locales list*:
 - A. If this *range* begins with the [subtag](#) "*" (e.g. "*-us" or just "*"), or contains any [space characters](#), skip all the steps in this algorithm below, and move onto the next *range*.
 - B. If this *range* begins with the [subtag](#) "i" or the *range* is marked as "deprecated" in the [IANA Language Subtag Registry](#), or is or is null, skip all the steps in this algorithm below, and move onto the next *range*.
 - C. If this *range* contains any [subtag](#) "*", remove the "*" and its preceding hyphen (U+002D) (e.g., 'en-*-us' becomes 'en-us').
 - D. While *range* contains [subtags](#):
 1. Add the value of the *range* to the [user agent locales](#).
 2. Remove the right most [subtag](#) from *range* and append the resulting value to [user agent locales](#). Continue removing the right most [subtag](#) and adding the result to [user agent locales](#) until there are no [subtags](#) left in *range*.

||For example, if the *range* was "zh-hans-cn", then the [user agent locales](#) become "zh-hans-cn, zh-hans, zh".
 3. Move onto the next *range* and go to step 1 in this algorithm.
3. Append the value "*" to the end of [user agent locales](#).

For example, an *unprocessed locales list* that contains "en-us, en-au, en, fr-ca, zh-hans-cn" would result in a [user agent locales](#) that contains "en-us, en, en-au, en, en, fr-ca, fr, zh-hans-cn, zh-hans, zh, *".

For example, an *unprocessed locales list* that contains "en-us, en, fr-ca, en, en-ca" would result in a [user agent locales](#) that contains "en-us, en, en, fr-ca, fr, en, en-ca, en, *".

9.1.13 Rule for Determining if a Potential Zip Archive is a Zip Archive

The rule for determining if a potential Zip archive is a Zip archive is given by the following algorithm.

1. Let *potential archive* be the acquired resource.
2. Check if the first four bytes of *potential archive* match the [magic numbers for a Zip archive](#) (50 4B 03 04).
3. If the first four bytes do not match the [magic numbers for a Zip archive](#), then return an error.
4. Otherwise, return true.

Note:

A user agent can inspect the *potential archive* once it has acquired the first four bytes of the [potential Zip archive](#) or can wait until all the data of the [potential Zip archive](#) has been completely acquired.

Step 1 - Acquire a Potential Zip Archive

Step 1 involves acquiring a [potential Zip archive](#) and confirming that it is a [Zip archive](#) by applying the [rule for determining if a potential Zip archive is a Zip archive](#). A user agent will acquire a [potential Zip archive](#) from a data transfer protocol that either labels resources with a [media type](#) (e.g. [HTTP](#)) or from a data transfer protocol that does not label resources with a media type (e.g., BitTorrent or Bluetooth).

9.1.1 Acquisition of a Potential Zip archive Labeled with a Media Type

It is RECOMMENDED that a user agent [support acquisition of a potential Zip archive from a protocol that labels resources with a media type](#) (e.g., getting a widget package over [HTTP](#)). If a user agent [supports](#) acquisition of a potential Zip archive from a protocol that labels resources with a media type, then the [user agent](#) needs to process resources labeled with the [valid widget media type](#) (`application/widget`), regardless of whether the resource contains a file extension or not, by applying the [rule for determining if a potential Zip archive is a Zip archive](#). During the [acquisition of a potential Zip archive labeled with a media type](#), unless the user agent [supports](#) legacy or proprietary [media types](#), [unsupported media types](#) are in error and the [user agent](#) MUST treat the [potential Zip archive](#) as an [invalid widget package](#). {ta-GVVlvsdEUo}

If the result of the user agent applying the [rule for determining if a potential Zip archive is a Zip archive](#) is true, meaning that the [potential Zip archive](#) is a [Zip archive](#), then the [user agent](#) MUST proceed to [Step 2](#). Otherwise, if an error is returned, the [user agent](#) MUST treat the [potential Zip archive](#) as an [invalid widget package](#). {ta-qxLSCRCHIN}

For example, in [HTTP](#), where the Content-Type header matches `application/widget`.

If the protocol used for acquisition of a [potential Zip archive](#) does not provide, or otherwise include, a [media type](#), then a [user agent](#) SHOULD treat the acquired [potential Zip archive](#) as if it has been [acquired from a protocol that does not label resources with a media type](#).

In this example, the [media type](#) of the Content-Type is not one [supported](#) by the user agent, so the user agent would treat the [potential Zip archive](#) as an [invalid widget package](#):

Request

```
GET /foo.wgt HTTP/1.1
Host: www.example.com
Accept: application/widget
```

Response

```
HTTP/1.1 200 OK
Date: Tue, 04 Sep 2007 00:00:38 GMT
Last-Modified: Mon, 03 Sep 2007 06:47:19 GMT
Content-Length: 1337
Content-Type: application/x-gadget
```

9.1.2 Acquisition of Potential Zip Archive not Labeled with a Media Type

When acquiring a potential Zip archive that has not been labeled with a media type (e.g., from a file system), a [user agent](#) SHOULD attempt to process the resource regardless of the file extension (including situations when the file extension is absent) by applying the [rule for determining if a potential Zip archive is a Zip archive](#). If the [rule for determining if a potential Zip archive is a Zip archive](#) return true, proceed to [Step 2](#). Otherwise, if an error was returned, the [user agent](#) MUST treat the [potential Zip archive](#) as an [invalid widget package](#). {ta-FDQBROtzW}

Step 2 - Verify the Zip Archive

To verify that a [Zip archive](#) and its [file entries](#) conform to this specification, a [user agent](#) MUST apply the [rule for verifying a zip archive](#). If the [rule for verifying a zip archive](#) returns true, then the [user agent](#) MUST go to [Step 3](#). Otherwise, if the [rule for verifying a zip archive](#) returns an error, then the [user agent](#) MUST treat the [Zip archive](#) as an [invalid widget package](#). {ta-uLHyIMvLwz}

Step 3 - Set the Configuration Defaults

In [Step 3](#), a [user agent](#) MUST set the following variables and default values as defined in the [table of configuration defaults](#).

Note:

When a [null](#) value is assigned to a variable in the [table of configuration defaults](#), a user agent needs to treat the value as [null](#) (i.e., not as an empty string and not as the text string "[null](#)").

Table of Configuration Defaults

| Variable | Type | Overridden in | Description |
|-------------------------------------|----------------------|--|--|
| <i>author email</i> | String | Step 7 | The value of the author element's email attribute (if any). |
| <i>author href</i> | IRI | Step 7 | The value of the author element's href attribute (if any). |
| <i>author name</i> | String | Step 7 | A localizable string that represents the content of the author element (if any). |
| <i>feature list</i> | List | Step 7 | A list of features that correspond to features that were requested via feature elements (if any). Each item in the list corresponds to a feature element's name attribute, whether it is required, and any associated parameters (if any). |
| <i>icons</i> | List of file entries | Step 7 , Step 9 | The icons of the widget as they correspond to the default icons and to the occurrence of custom icons that are supported by the widget package (if any). |
| <i>start file encoding Variable</i> | String | Step 7 | The character encoding of the custom start file , corresponding to either the content element's encoding attribute (if any), or the default encoding . |
| | Type | Overridden by | Description |

| | | | |
|--------------------------------|------------------------------------|--|---|
| <i>start file content-type</i> | String | Step 7 | The media type of the start file , corresponding to the content element's type attribute or to a media type derived from the default start files table . |
| <i>widget config doc</i> | File | Step 6 | The file that is the configuration document for the widget package . |
| <i>widget description</i> | String | Step 7 | The text content of the description element in the configuration document . |
| <i>widget height</i> | positive number | Step 7 | The value of the widget element's height attribute in the configuration document (if any). |
| <i>widget id</i> | String | Step 7 | The value of the widget element's id attribute in the configuration document (if any). |
| <i>widget license</i> | String | Step 7 | The text content of the license element in the configuration document (if any). |
| <i>widget license file</i> | File | Step 7 | A file derived if the value of the license element's href is a Zip relative path to a file within the widget package . |
| <i>widget license href</i> | IRI | Step 7 | The value of the license element's href attribute in the configuration document (if any). |
| <i>widget name</i> | Localizable String | Step 7 | The text content of the name element in the configuration document (if any). |
| <i>widget preferences</i> | List | Step 7 | The widget's preferences , corresponding to the preference elements in the configuration document (if any). Unless an end-user explicitly requests that these values be reverted to the values as declared in the configuration document , a user agent MUST NOT reset the value of the widget preferences variable on subsequent initializations of the widget. |
| <i>widget short name</i> | Localizable String | Step 7 | The value of the name element's short attribute in the configuration document (if any). |
| <i>widget version</i> | Localizable String | Step 7 | The value of the widget element's version attribute in the configuration document (if any). |
| <i>widget width</i> | positive number | Step 7 | The value of the widget element's width attribute in the configuration document (if any). |
| <i>widget window modes</i> | List of strings | Step 7 | The value of the widget element's viewmodes attribute in the configuration document (if any). |
| <i>widget start file</i> | File entry | Step 7 , Step 8 | The start file for the widget package , corresponding to either one of the default start files table or the file identified by the content element's src attribute. |

| Variable | Type | Overridden by | Description |
|----------|------|---------------|-------------|
|----------|------|---------------|-------------|

| <i>user agent locales</i> | List of strings | Step 5 | A list of language tags . |
|---------------------------|-----------------|------------------------|---|
| Variable | Type | Overridden by | Description |

Step 4 - Locate and Process the Digital Signature

If the user agent does not [support \[Widgets-DigSig\]](#), then the user agent MUST skip [Step 4](#) and go to [Step 5](#). Otherwise, the user agent MUST apply the algorithm to locate digital signatures, which is defined in the [\[Widgets-DigSig\]](#) specification under the section named [Locating signature files in a widget package](#).

Step 5 - Derive the User Agent's Locales

The **end-user's language ranges** represents the end-user's preferred languages and regional settings, which are derived from the operating system or directly from the user agent. As there are numerous ways a user agent can derive the end-user's preferred languages and regional settings, the means by which those values are derived are beyond the scope of this specification and left up to the implementation.

During the [rule for deriving the user agent locales](#) defined below, the user agent will need to construct a list *unprocessed locales*. Each item in the *unprocessed locales* is a string in lowercase form, that conforms to the production of a Language-Tag, as defined in the [\[BCP47\]](#) specification. A string that conforms to the production of a Language-Tag is referred to as a *language range* [\[BCP47\]](#) (e.g. 'en-au', which is the range of English as spoken in Australia, and 'fr-ca', which is the range of French as spoken in Canada, etc.). A language range is composed of one or more **subtags** that are delimited by a U+002D HYPHEN-MINUS ("-").

The first item of the *unprocessed locales* represents the user's most preferred language range (i.e., the language/region combination the user agent assumes the end-user most wants to see content in), followed by the next most preferred language range, and so forth.

For example, in an *unprocessed locales list* that contains 'en-us, en, fr, es', English as spoken in the United States is preferred over English, and English is preferred over French, and French is preferred over Spanish, and Spanish is preferred over [default content](#).

For example, the end-user may have specified her preferred languages and regional settings at install time by selecting a preferred language, or languages from a list, or a list of preferred languages and regional settings could have been dynamically derived from the end-user's geographical location, etc.

In [Step 5](#), the [user agent](#) MUST apply the [rule for deriving the user agent locales](#).

Step 6 - Locate the Configuration Document

This step involves searching within the Zip archive for a [configuration document](#).

In [Step 6](#), a [user agent](#) MUST apply the algorithm to [locate the configuration document](#). {ta-ZjcdAxFMSx}

The algorithm to **locate the configuration document** is as follows:

1. Search at the [root of the widget package](#) for a [file entry](#) whose *file name field* case-sensitively matches the [valid configuration document file name](#) (config.xml).
2. If a match is made, then let *widget config doc* to the result of applying [rule for extracting file data from a file entry](#) to the matching [file entry](#). If no match is made (meaning that *widget config doc* is `null`), then treat the [Zip archive](#) as an [invalid widget package](#).

Step 7 - Process the Configuration Document

The purpose of processing the [configuration document](#) is to override the values in the [table of configuration defaults](#), which are used during [initialization](#) and at runtime, and to select the appropriate localized content (if any) to be presented to the end user.

In conjunction to the algorithm for processing a [configuration document](#) given below, this section firstly defines some terminology used by the processing algorithm and describes how localized elements are processed.

During [Step 7](#), a [user agent](#) MUST apply the [algorithm to process a configuration document](#).

9.1.1 Terminology Used in Processing Algorithm

In the [algorithm to process a configuration document](#), the term **in error** is used to mean that an element, or attribute, or [file](#) in a [configuration document](#) is non-conforming to the rules of this specification. How an element or an attribute is to be treated when it is in error is always given when the term is used; but will generally require the user agent to [ignore](#) any element, attribute, or [file](#) that is in error.

To **ignore** means to act as if the element, attribute, or file that is [in error](#) is absent (i.e., not declared or included by the author) in the widget package or configuration document. A user agent MUST, however, keep a record of all element types it has attempted to process even if they were ignored (this is to determine if the user agent has attempted to process an element of a given type already).

In the case the user agent is asked to [ignore](#) an [\[XML\]](#) element or node, a [user agent](#) MUST :

1. Stop processing the current *element*, [ignoring](#) all of the *element's* attributes and child nodes (if any), and proceed to the next *element* in the *elements list*.
2. Make a record that it has attempted to process an element of that type.

In the following example, the user agent ignores both [content](#) elements. The user agent ignores the first because it lacks a [src](#) attribute. The user agent ignores the second because it is not the first content element to be encountered by the user agent.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <!-- User agent ignores the first, but records
  that it has attempted to process a content element
  -->

  <content/>

  <!-- The use agent knows that it has previously attempted
  to process a content element, hence this content element
  is ignored.
  -->

  <content src="cats.html"/>
</widget>
```

To **associate** means that two or more pieces of information are bound and stored together for the purpose of later processing (e.g., the name of a [feature](#), and if it is [required](#), and any associated [parameters](#)). How associated data is represented is left up-to the implementation (e.g., a user agent could use an array, an object, a hash map, etc.).

The [lookup](#) algorithm is defined in [\[BCP47\]](#). It is used in this Step to match [localized content](#) in the [configuration document](#) to the language ranges held by the [user agent locales](#) (if any).

9.1.2 Algorithm to Process a Configuration Document

The [algorithm to process a configuration document](#) is as follows:

1. Let *doc* be the result of loading the [widget config doc](#) as a [DOMCore] Document using an [XML] parser that is both [XMLNS]-aware and `xml:lang` aware.
2. If *doc* is not namespace well-formed [XML], then the [user agent](#) MUST terminate this algorithm and treat this [widget package](#) as an [invalid widget package](#). {ta-kILDaEgJeU}
3. Let *root element* be the `documentElement` of *doc*.
4. If the *root element* is not a [widget](#) element in the [widget namespace](#), then the [user agent](#) MUST terminate this algorithm and treat this [widget package](#) as an [invalid widget package](#). {ta-ACCJfDGwDQ}
5. Otherwise, the *element* is a [widget](#) element:

A. If the `defaultLocale` attribute is used, then let *default locale* be the result of applying the [rule for getting a single attribute value](#) to the `defaultLocale` attribute:

A. If the *default locale* is [in error](#) or an empty string or already contained by the [user agent locales](#) list, then the [user agent](#) MUST [ignore](#) the `defaultLocale` attribute. {ta-defaultlocale-ignore}

B. If *potential default locale* is a [valid language tag](#) and the [user agent locales](#) does not contain the value of *default locale*, the [user agent](#) MUST prepend the value of *potential default locale* into the the [user agent locales](#) list as the second-last item (i.e., at position `length - 1`). {ta-defaultlocale-process}

For example, if the *default locale* is the value "fr", and the [user agent locales](#) contains the values "jp,us,*", then the [user agent locales](#) list becomes "jp,us,fr,*".

For example, if the *default locale* is the value "en", and the [user agent locales](#) only contains the value "*", then the [user agent locales](#) list becomes "en,*".

For example, if the *default locale* is the value "en", and the [user agent locales](#) already contains the values "en,*", then the user agent would ignore the *default locale* because it is already contained by the [user agent locales](#) list.

- B. If the `id` attribute is used, then let *id* be the result of applying the [rule for getting a single attribute value](#) to the `id` attribute. If *id* is a [valid IRI](#), then let [widget id](#) be the value of the *id*. If the *id* is not a [valid IRI](#) or is an empty string, then the [user agent](#) MUST [ignore](#) the attribute. {ta-RawAIWHoMs}
- C. If the `version` attribute is used, then let *version value* be the result of applying the [rule for getting a single attribute value](#) to the `version` attribute. If the *version* is an empty string, then the [user agent](#) MUST [ignore](#) the attribute; otherwise, let [widget version](#) be the value of *version value*. {ta-VerEfVGeTc}
- D. If the `height` attribute is used, then let *normalized height* be the result of applying the [rule for parsing a non-negative integer](#) to the value of the attribute. If the *normalized height* is not [in error](#) and greater than 0, then let [widget height](#) be the value of *normalized height*. If the `height` attribute is [in error](#), then the [user agent](#) MUST [ignore](#) the attribute. {ta-BxjoiWHaMr}
- E. If the `width` attribute is used, then let *normalized width* be the result of applying the [rule for parsing a non-negative integer](#) to the value of the attribute. If the *normalized width* is not [in error](#) and greater than 0, then let [widget width](#) be the value of *normalized width*. If the `width` attribute is [in error](#), then the [user agent](#) MUST [ignore](#) the attribute. {ta-UScJfQHPPy}
- F. If the `viewmodes` attribute is used, then the [user agent](#) MUST let *viewmodes list* be the result of applying the [rule for getting a list of keywords from an attribute](#): {ta-

viewmodes}

- A. From the *viewmode list*, remove any [unsupported](#) items.
- B. From the *viewmode list*, remove any duplicated items from right to left.

For example, *viewmode list* with a value of "windowed fullscreen windowed floating fullscreen windowed" would become "windowed fullscreen floating".

- C. Let [widget window modes](#) be the value of *viewmodes list*.

6. If the [widget](#) element does not contain any child elements, then the [user agent](#) MUST terminate this algorithm and go to [Step 8](#). {ta-MFcsScFEaC}
7. Otherwise, let *element list* be an empty list.
8. For each *range* in the [user agent locales](#), starting from the first and moving to the last:

- A. If the value of *range* is not "*", then retaining [document order](#), let *matching elements* be the result of applying [lookup](#) to the child elements that are defined as being [localizable via xml:lang](#) (i.e., the [name](#), [description](#), and [license](#) elements) that are direct descendants of the *root element* and whose [xml:lang](#) attribute matches the current *range*. Append *matching elements* to the [element list](#).

Note:

In the context of this specification, the above conformance requirement is intended to match the [name](#), [description](#), and [license](#) elements. However, it is written in an abstract manner to provide a hook for future specifications that want to define elements that also support being [localizable via xml:lang](#).

- B. If the value of *range* is "*", retaining [document order](#), let *unlocalized elements* be all child elements that are direct descendants of the *root element* that do not have an implicit or explicit [xml:lang](#) attribute (i.e., match [default content](#)). Append *unlocalized elements* to the [element list](#).

For example, consider the following configuration document.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <name>E1 Widget!</name>
  <name xml:lang="fr">Le Widget</name>
  <name xml:lang="en">The Widget</name>
</widget>
```

For a use agent whose [user agent locales](#) contains "en,fr,*", the *matching elements* would be in the following order:

1. <name xml:lang="en">The Widget</name>
2. <name xml:lang="fr">Le Widget</name>
3. <name>E1 Widget!</name>

For a use agent whose [user agent locales](#) contains "en,*", the *matching elements* would be in the following order:

1. <name xml:lang="en">The Widget</name>
2. <name>E1 Widget!</name>

For a use agent whose [user agent locales](#) contains "jp,*", the *matching elements* would be in the following order:

1. <name>E1 Widget!</name>

9. For each *element* in the *elements list*, if the *element* is one of the following:

A [name](#) element:

If this is not the first [name](#) element encountered by the user agent, then the [user agent](#) MUST [ignore](#) this *element*. {ta-LYLMhryBBT}

If this is the first [name](#) element encountered by the user agent, then the [user agent](#) MUST: {ta-AYLMhryBnD}

1. Record that an attempt has been made by the user agent to process a [name](#) element.
2. Let *widget name* be the result of applying the [rule for getting text content with normalized white space](#) to this element.
3. If the [short](#) attribute is used, then let *widget short name* be the result of applying the [rule for getting a single attribute value](#) to the [short](#) attribute.

A [description](#) element:

If this is not the first [description](#) element encountered by the user agent, then the [user agent](#) MUST [ignore](#) this *element*. {ta-UEMbyHERkl}

If this is the first [description](#) element encountered by the user agent, then the [user agent](#) MUST: {ta-VdCEyDVSA}

1. Record that an attempt has been made by the user agent to process a [description](#) element.
2. Let *widget description* be the result of applying the [rule for getting text content](#) to this element.

A [license](#) element:

If this is not the first [license](#) element encountered by the user agent, then the [user agent](#) MUST [ignore](#) this *element*. {ta-vcYJAPVEym}

If this is the first [license](#) element used, then the [user agent](#) MUST: {ta-YUMJAPVEgl}

1. Record that an attempt has been made by the user agent to process a [license](#) element.
2. Let *license text* be the result of applying the [rule for getting text content](#) to this element. [Associate](#) *license text* with [widget license](#).
3. If the [href](#) attribute is used, then let *potential license href* be the result of applying the [rule for getting a single attribute value](#) to the [href](#) attribute.
4. If *potential license href* is not a [valid IRI](#) or a [valid path](#), then the [href](#) attribute is [in error](#) and the [user agent](#) MUST [ignore](#) the attribute.
5. If *potential license href* is a [valid IRI](#), then let [widget license href](#) be the value of *potential license href*.
6. If [license href](#) is a [valid path](#), then let *file* be the result of applying the [rule for finding a file within a widget package](#) to [license href](#).
7. If *file* is not a [processable file](#), as determined by applying the [rule for identifying the media type of a file](#), then [ignore](#) this *element*.
8. Otherwise, let *widget license file* be the value of [file](#).

An [icon](#) element:

If the [src](#) attribute of this [icon](#) element is absent, then the [user agent](#) MUST [ignore](#) this *element*. {ta-iipTwNshRg}

Let *path* be the result of applying the [rule for getting a single attribute value](#) to the [src](#) attribute of this [icon](#) element. If *path* is not a [valid path](#) or is an empty string, then the [user agent](#) MUST [ignore](#) this *element*. {ta-roCaKRxZhS}

Let *file* be the result of applying the [rule for finding a file within a widget package](#) to *path*. If *file* is not a [processable file](#), as determined by applying the [rule for identifying the media type of a file](#), or already exists in the [icons](#) list, then the [user agent](#) MUST [ignore](#) this element. {ta-iuJHnskSHq}

Otherwise,

1. If the [height](#) attribute is used, then let *potential height* be the result of applying the [rule for parsing a non-negative integer](#) to the attribute's value. If the *potential height* is not [in error](#) and greater than 0, then [associate](#) the *potential height* with *file*. Otherwise, the [height](#) attribute is [in error](#) and the [user agent](#) MUST [ignore](#) the attribute. {ta-eHUaPbgfKg}
2. If the [width](#) attribute is used, then let *potential width* be the result of applying the [rule for parsing a non-negative integer](#) to the attribute's value. If the *potential width* is not [in error](#) and greater than 0, then [associate](#) the *potential width* with *file*. Otherwise, the [width](#) attribute is [in error](#) and the [user agent](#) MUST [ignore](#) the attribute. {ta-nYAcofihvj}
3. Add *file* and any [associated potential width](#) and/or *potential height* to the list of [icons](#).

An [author](#) element:

If this is not the first [author](#) element encountered by the user agent, then the [user agent](#) MUST [ignore](#) this *element*. {ta-sdwhMozwlc}

If this is the first [author](#) element used, then the [user agent](#) MUST: {ta-argMozRiC}

1. Record that an attempt has been made by the user agent to process a [author](#) element.
2. If the [href](#) attribute is used, then let *href-value* be the value of applying the [rule for getting a single attribute value](#) to the [href](#) attribute.
3. If *href-value* is a [valid IRI](#), then let [author href](#) be the value of the [href](#) attribute. Otherwise, if *href-value* is not a [valid IRI](#), then [ignore](#) the [href](#) attribute.
4. If the [email](#) attribute is used, then let [author email](#) be the result of applying the [rule for getting a single attribute value](#) to the [email](#) attribute.
5. Let [author name](#) be the result of applying the [rule for getting text content with normalized white space](#) to this element.

A [preference](#) element:

If a [value](#) attribute of the [preference](#) element is used, but the [name](#) attribute is absent, then this [preference](#) element is [in error](#) and the [user agent](#) MUST [ignore](#) this *element*. Otherwise, the user agent MUST: {ta-DwhJBJRQN}

1. Let *name* be the result of applying the [rule for getting a single attribute value](#) to the [name](#) attribute.
2. If the *name* is an empty string, then this *element* is [in error](#); [ignore](#) this *element*.
3. If [widget preferences](#) already contains a preference whose name case-sensitively matches the value of *name*, then this *element* is [in error](#); [ignore](#) this *element*.

4. If *name* was not [in error](#), let *preference* be an empty object.
5. Associate *name* with *preference*.
6. Let *value* be the result of applying the [rule for getting a single attribute value](#) to the *value* attribute.
7. [Associate](#) *value* with *preference*.
8. If a [readonly](#) attribute is used, then let *readonly* be the result of applying the [rule for getting a single attribute value](#) to the *readonly* attribute. If *readonly* is not a [valid boolean value](#), then let the value of *readonly* be the value 'false'.
9. [Associate](#) *readonly* with the *preference*.
10. Add the *preference* and the [associated](#) *name*, *value* and *readonly* variables the list of [widget preferences](#).

A [content](#) element:

If this is not the first [content](#) element encountered by the user agent, then the [user agent](#) MUST [ignore](#) this *element*. {ta-hkWmGJgfv}

If this is the first [content](#) element, then the [user agent](#) MUST:

1. Record that an attempt has been made by the user agent to process a [content](#) element.
2. If the [src](#) attribute of the [content](#) element is absent or an empty string, then the [user agent](#) MUST [ignore](#) this *element*. {ta-LTUJGJFCOU}
3. Let *path* be the result of applying the [rule for getting a single attribute value](#) to the value of the [src](#) attribute.
4. If *path* is not a [valid path](#), then the [user agent](#) MUST [ignore](#) this *element*. {ta-plffQywZin}
5. If *path* is a [valid path](#), then let *file* be the result of applying the [rule for finding a file within a widget package](#) to *path*. If *file* is [null](#) or in error, then the [user agent](#) MUST [ignore](#) this *element*. {ta-LQcjNKBLUZ}
6. If the [type](#) attribute of the [content](#) element is absent, then check if *file* is [supported](#) by the user agent by applying the [rule for identifying the media type of a file](#). If the *file* is [supported](#), then let the [widget start file](#) be the file referenced by the [src](#) attribute and let [start file content-type](#) be the [supported media type](#) as was derived by applying the [rule for identifying the media type of a file](#).
7. If the [encoding](#) attribute is used, then let *content-encoding* be the result of applying the [rule for getting a single attribute value](#) to the value of the [encoding](#) attribute. If the character encoding represented by the value of *content-encoding* is [supported](#) by the user agent, then let the [start file encoding](#) be the value of *content-encoding*. If *content-encoding* is an empty string or [unsupported](#) by the user agent, then a [user agent](#) MUST [ignore](#) the [encoding](#) attribute. {ta-dPOgiLQKNK}
8. If the [type](#) attribute is used, then let *content-type* be the result of applying the [rule for getting a single attribute value](#) to the value of the [type](#) attribute. If the value of *content-type* is a [media type supported](#) by the user agent, then let the [start file content-type](#) be the value of [content type](#). If value of *content-type* is [invalid](#) or [unsupported](#) by the user agent, then a [user agent](#) MUST treat the [widget package](#) as an [invalid widget package](#). {ta-palabGIIMC}

9. If the [start file encoding](#) was set in step 7 of this algorithm as a result of processing a valid and [supported](#) value for the [content](#) element's [encoding](#) attribute, then the [user agent](#) MUST skip this step in this algorithm. Otherwise, if the value of *content-type* is a [media type supported](#) by the user agent and if *content-type* contains one or more [\[MIME\] parameter](#) components whose purpose is to declare the character encoding of the [start file](#) (e.g., the value "text/html; charset=Windows-1252", where charset is a parameter component whose purpose is to declare the character encoding of the start file), then let [start file encoding](#) be the value of the last [supported parameter](#) components whose purpose is to declare the character encoding of the [start file](#). {ta-aaaaaaaa}

In the following example, the user agent would set the start file encoding to ISO-8859-1 and would ignore the charset parameter used in the type attribute.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <content src      = "start.php"
          type      = "text/html; charset=Windows-1252"
          encoding  = "ISO-8859-1" />
</widget>
```

In the following example the user agent would set the [start file encoding](#) to Windows-1252, if the user agent [supports](#) that character encoding.

```
<widget xmlns="http://www.w3.org/ns/widgets">
  <content src = "start.php"
          type = "text/html; charset=Windows-1252"/>
</widget>
```

A [param](#) element:

If this [param](#) element is not a direct child of a [feature](#) element, then the [user agent](#) MUST [ignore](#) this [param element](#). {ta-KNiLPOKdgQ}

Note:

How a [param](#) element is to be processed when it is inside a [feature](#) element is defined below.

A [feature](#) element:

The [user agent](#) MUST process a [feature](#) element in the following manner: {ta-rZdcMBExBX}

1. If the [name](#) attribute of this [feature](#) element is absent, then the [user agent](#) MUST [ignore](#) this [element](#).
2. Let *required-feature* be the value true.
3. If a [required](#) attribute is used, then let *potentially-required-feature* be the result of applying the [rule for getting a single attribute value](#) to the [required](#) attribute. If the value of *potentially-required-feature* is the value "false", then let *required-feature* be the value 'false'.
4. Let *feature-name* be the result of applying the [rule for getting a single attribute value](#) to the value of the [name](#) attribute.

Note:

This specification allows [feature](#) elements with the same [name](#) attribute value to be declared more than once. Handling of duplicate [feature requests](#) is left up to the implementation or the specification that defines the feature.

5. If *feature-name* is not a [valid IRI](#), and *required-feature* is true, then the [user agent](#) MUST treat this widget as an [invalid widget package](#). {ta-paWbGHyVrG}

6. If *feature-name* is not a [valid IRI](#), and *required-feature* is `false`, then the [user agent](#) MUST [ignore](#) this element. {ta-ignore-unrequired-feature-with-invalid-name}
7. If *feature-name* is not [supported](#) by the user agent, and *required-feature* is `true`, then the [user agent](#) MUST treat this widget as an [invalid widget package](#). {ta-vOBaOcWfl}
8. If *feature-name* is not [supported](#) by the user agent, and *required-feature* is `false`, then the [user agent](#) MUST [ignore](#) this *element*. {ta-luyKMFABLX}
9. Let *feature* be an object that represents this [feature](#). [Associate](#) *required-feature* and *feature-name* with the [feature](#) object.
10. If the [feature](#) element contains any [param](#) elements as direct descendants, then, for each child [param](#) element that is a direct descendants of this [feature](#) element, starting from the first moving to the last in [document order](#):
 - A. If either the a [name](#) attribute or the [value](#) attribute is missing, then this [param](#) element is [in error](#) and the [user agent](#) MUST [ignore](#) this *element*. {ta-EGkPfcBOz}
 - B. Let *param-name* be the result of applying the [rule for getting a single attribute value](#) to the [name](#) attribute.
 - C. If *param-name* is an empty string, then this [param](#) element is [in error](#) and the [user agent](#) MUST [ignore](#) this *element*. {ta-CEGwkNQcWo}
 - D. Let *param-value* be the result of applying the [rule for getting a single attribute value](#) to the [value](#) attribute.
 - E. Let *param* be an object that represents this parameter.
 - F. [Associate](#) *param-name* and *param-value* with *param*.
 - G. [Associate](#) *param* with [feature](#).
11. Append [feature](#), to the [feature list](#).

Any other type of element:

If the user agent [supports](#) the element, then the [user agent](#) MUST process it in accordance with whatever specification defines that element (if any). Otherwise, the [user agent](#) MUST [ignore](#) the element. {ta-bbbbbbbbbbb}

Step 8 - Locate the Start File

If [widget start file](#) of the [table of configuration defaults](#) contains a [file](#) (i.e. [widget start file](#) is not `null`), then a [user agent](#) MUST skip [Step 8](#) and go to [Step 9](#). {ta-BnWPqNvNVo}

If [widget start file](#) does not contain a [file](#), the [user agent](#) MUST apply the [algorithm to locate a default start file](#). {ta-RGNHRBWNZV}

The [algorithm to locate a default start file](#) is as follows:

1. For each *file name* in the [default start files table](#) (from top to bottom) that has a [media type](#) that is [supported](#) by the user agent:
 - A. Let *potential-start-file* be the result of applying the [rule for finding a file within a widget package](#) to *file name*.
 - B. If *potential-start-file* is `null` or [in error](#), [ignore](#) this *file name* and move onto the next *file name* in the [default start files table](#).

C. If *potential-start-file* is a [file](#), then:

- A. Let *widget start file* be the value of *potential-start-file*.
- B. Let *start file content-type* be the [media type](#) given in the media type column of the [default start files table](#).
- C. Terminate this algorithm and go to [Step 9](#).

2. If after searching for every *file* in the [default start files table](#) no default start file is found, then treat this widget as an [invalid widget package](#).

Step 9 - Process the Default Icons

This step describes how to locate the [default icons](#).

In [Step 9](#), a [user agent](#) MUST apply the [algorithm to locate the default icons](#). {ta-FAFYMEGELU}

The **algorithm to locate the default icons** is as follows:

1. For each *file name* in the [default icons table](#) (from top to bottom) that has a [media type](#) that is [supported](#) by the user agent:
 - A. Let *potential-icon* be the result of applying the [rule for finding a file within a widget package](#) to *file name*.
 - B. If the *potential-icon* is a [processable file](#), determined by the [media type](#) given in the media type column of the [default icons table](#), and the *potential-icon* does not already exist in the [icons](#) list of the [table of configuration defaults](#), then append the value of *potential-icon* to the [icons](#) list of the [table of configuration defaults](#).
 - C. Move onto the next *file name* in the [default icons table](#).

Appendix

Media Type Registration for application/widget

This appendix is the MIME media type registration for "application/widget". This registration has been [approved](#) by the Internet Assigned Numbers Authority (IANA) .

Registration with IANA was conducted in conformance with [BCP 13](#) and [W3CRegMedia](#).

Type name:

application

subtype name:

[widget](#)

Required parameters:

None.

Optional parameters:

None.

Encoding considerations:

Widget packages are binary data and thus are encoded for MIME transmission. As a widget package is binary, it requires encoding on transports not capable of handling binary. The

same guidelines that apply to `application/octet-stream` apply to widget packages (see [\[MIME\]](#)).

Security considerations:

In addition to the security considerations specified for Zip files in the [\[Zip-MIME\]](#) registration, there are a number of security considerations that need to be taken into account when dealing with [widget packages](#) and [configuration documents](#).

As the [configuration document](#) format is [\[XML\]](#) and will commonly be encoded using [\[Unicode\]](#), the security considerations described in [\[XML-MIME\]](#) and [\[UTR36\]](#) apply. In addition, implementers need to impose their own implementation-specific limits on the values of otherwise unconstrained attribute types, e.g. to prevent denial of service attacks, to guard against running out of memory, or to work around platform-specific limitations.

The configuration document allows authors, through the [feature](#) element, to request permission to enable third-party runtime components and APIs. As these features are outside the scope of this specification, significant caution needs to be taken when granting a widget the capability to use a feature. Features themselves define their own security considerations.

[Widget packages](#) will generally contain ECMAScript, HTML, CSS files, and other media, which are executed in a sand boxed environment. As such, implementers need to be aware of the security implications for the types they [support](#). Specifically, implementers need to consider the security implications outlined in the [\[CSS-MIME\]](#) specification, the [\[ECMAScript-MIME\]](#) specification, and the [\[HTML-MIME\]](#) specification.

As widget packages can contain content that is able to simultaneously interact with the local device and a remote host, implementers need to consider the privacy implications resulting from exposing private information to a remote host. Mitigation and in-depth defensive measures are an implementation responsibility and not prescribed by this specification. However, in designing these measures, implementers are advised to enable user awareness of information sharing, and to provide easy access to interfaces that enable revocation of permissions.

As this specification relies on the standardized heuristics for determining the content type of files defined in the [\[SNIFF\]](#) specification, implementers need to consider the security considerations discussed in the [\[SNIFF\]](#) specification.

As this specification allows for the declaration of IRIs within certain elements of a configuration documents, implementers need to consider the security considerations discussed in the [\[IRI\]](#) specification. Implementations intending to display IRIs and IDNA addresses found in the [configuration document](#) are strongly encouraged to follow the security advice given in [\[UTR36\]](#). This could include, for example, behaving as if the `dir` attribute had no effect on any [IRI attributes](#), [path attributes](#), and the `author` element's `email` attribute.

In addition, user agents need to be careful about trusting path components found in the widget package. Such path components might be interpreted by operating systems as pointing at security critical files outside the widget environment proper, and naive unpacking of widget packages into the file system might lead to undesirable and security relevant effects, such as overwriting of system files.

Interoperability considerations:

Some issues can arise with regards to character encodings of file names, the length of zip relative paths, and the use of certain strings as file names.

This specification does not put a restriction on the byte length of a [Zip relative path](#), so a user agent SHOULD to be able to deal with [Zip relative paths](#) that have lengths longer than 250 bytes. As some operating systems have restrictions on how long a path length can be, authors need to keep the lengths of relative paths at less than 250 bytes. Unicode code points may require more than one byte to encode a character, which can result in a path whose length is less than 250 characters but whose size is greater than 250 bytes.

Authors need to be aware that, at the time of publication, there are interoperability issues with regards to using characters outside the `safe-chars` range for file or folder names in a Zip archive when using Zipping tools bundled with operating systems. The interoperability issues have arisen from non-conforming implementations of the [ZIP](#) specification across operating systems: very few, if any, correctly support encoding file names in Unicode.

In the case where the Zip relative path is encoded using [\[UTF-8\]](#), the *language encoding flag (EFS)* needs to be set.

If an author chooses to use the `utf8-chars`, they need to thoroughly test their widgets on various platforms prior to distribution; otherwise it is suggested that authors restrict file and folder names to the `safe-chars` (characters in the US-ASCII range). In addition, having excessively long path names (e.g. over 120 characters) can also result in interoperability issues on some operating systems.

Authors need to avoid using [Zip forbidden characters](#) when naming the files used by a widget. These characters are reserved to maintain interoperability across various file systems and with [\[URI\]](#)s.

Authors need to avoid using the following words as either a [folder](#) or a [file-name](#) in a [Zip relative path](#) as they are reserved by some operating systems (case-insensitive): CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, LPT9, CLOCKS\$. For example, the following names are ok: "CON-tact.txt", "printer.lpt1", "DCOM1.pdf". However, "com3.txt" "Lpt1", "CoM9.gif" would not be.

In addition, authors need to avoid having a "." U+002E FULL STOP as the last character of a file or folder name as some operating systems will remove the character when the file is extracted from the Zip archive onto the device. Furthermore, avoid having the space character ([SP](#)) at the start or end of a file name; and take caution when using the "+" U+002B PLUS SIGN, as it might cause issues on some operating systems.

Published specification:

<http://www.w3.org/TR/widgets/>

Applications that use this media type:

User agents that claim conformance to this specification.

Magic number(s):

50 4B 03 04

File extension(s):

wgt

Macintosh file type code(s):

None.

Person & email address to contact for further information:

Steven Pemberton, member-webapps@w3.org

Intended usage:

Common.

Restrictions on usage:

None.

Author:

The W3C's Web Applications Working Group

Linking To a Widget Package From a HTML Document

This section is non-normative.

This section only applies to HTML user agents [HTML].

Auto-discovery enables a user agent to identify and install a [widget package](#) that is associated with an HTML page. When a page points to a [widget package](#), user agents SHOULD expose the presence of the [widget package](#) to the end-user and allow the end-user to install the [widget](#).

The link type "widget" indicates that a link of this type references a document that is a [widget package](#). In HTML, it may be specified for the `a`, `area` and `link` elements to create a hyperlink.

For example:

```
<a rel="widget"
  href="http://example.org/exampleWidget">
  The Example Widget
</a>
```

Table of Elements and Their Attributes

This section is non-normative.

This table lists all elements and respective attributes, as well as child-parent relationships, that make up the language of the [configuration document](#) format defined in this specification.

| Element | Description | Parent | Expected Children | Attributes | Type |
|-----------------------------|--|------------------------|--|---|---|
| widget | The root element of a configuration document . | none. | <ul style="list-style-type: none"> name description author license icon content feature preference | xml:lang dir id version height width viewmodes defaultlocale | language keyword IRI version numeric numeric Keyword list language |
| name | The name of the widget. | widget | <ul style="list-style-type: none"> span, text node | xml:lang dir short | language keyword displayable-string |
| description | Some text that describes the purpose of the widget. | widget | <ul style="list-style-type: none"> span text node | xml:lang dir | language keyword |
| author | The person or person that created the widget. | widget | <ul style="list-style-type: none"> span text node | xml:lang dir href email | language keyword IRI string |
| license | The license under which the widget is distributed. | widget | <ul style="list-style-type: none"> span text node | xml:lang dir href | language keyword IRI or path |
| icon | An iconic | widget | none. | xml:lang | language |

| | | | | | |
|----------------------------|--|--|---|--------------------------|----------------------------|
| | representation of the widget. | | | dir | keyword |
| | | | | src | path |
| | | | | width | numeric |
| | | | | height | numeric |
| content | The means to point to the "main file" of a widget; serves as a bootstrapping mechanism. | widget | none. | xml:lang | language |
| | | | | dir | keyword |
| | | | | src | numeric |
| | | | | type | media type |
| | | | | encoding | keyword |
| feature | A means to request the availability of a feature , such as an API, that would not normally be part of the default set of features provided by the user agent at runtime. | widget | <ul style="list-style-type: none"> param. | xml:lang | language |
| | | | | dir | keyword |
| | | | | name | keyword |
| | | | | required | boolean |
| preference | A means to declare a name-value pair that is made available to the widget at runtime. | widget | none. | xml:lang | language |
| | | | | dir | keyword |
| | | | | name | keyword |
| | | | | value | keyword |
| | | | | readonly | boolean |
| param | A means of declaring a parameter that can be used with a feature . | feature | none. | xml:lang | language |
| | | | | dir | keyword |
| | | | | name | string |
| | | | | value | string |
| span | A generic text container which is mainly used for internationalization purposes. | <ul style="list-style-type: none"> name description author license | <ul style="list-style-type: none"> span text node | xml:lang | language |
| | | | | dir | keyword |

Acknowledgements

Huge thanks to everyone who contributed their time and sent feedback to our public mailing, particularly the i18n WG.

This specification would not exist without the contribution of the following individuals:

Aaron Boodman, Adam Barth, Addison Phillips, Alexander Dreiling, Andrew Sledd, Andrew Welch, Arun Ranganathan, Arthur Barstow, Bárbara Barbosa Neves, Bil Corry, Brian Wilson, Bjoern Hoehrmann, Benoit Suzanne, Bert Bos, Boris Zbarsky, Bradford Lasse, Bryan Sullivan, Cameron McCormack, Cliff Schmidt, Claudio Venezia, Coach Wei, Corin Edwards, Cynthia Shelly, Cyril Concolato, Dan Brickley, Dan Connolly, Daniel Silva, David Clarke, Dean Jackson, David Poehman, David Pollington, David Rogers, Dominique Hazael-Massieux, Doug Schepers, Ed Voas, Felix Sasaki, Francois Daoust, Frederick Hirsch, Gautam Chandna, Geir Pedersen, Gene Vayngrib, Gorm Haug Eriksen, Guido Grassel, Guenter Klas, Hans S. Tømmerholt, Hari Kumar G, Henri Sivonen, Henry Story, Ian Hickson, Ivan Demarino, Jay Sweeney, Jean-Claude Dufourd, Jeff Decker, Jere Käpyaho, Jim Ley, Jo Rabin, Jon Ferraiolo, Jonas Sicking, Jose Manuel Cantera Fonseca, Josh Soref, Jouni Hakala, Joey Bacalhau, Julian Reschke, Kevin Lawver, Kai Hendry, Krzysztof Maczyński, Lachlan Hunt, Larry Masinter, Laurens Holst, Mark

Priestley, Marc Silbey, Marcin Hanclik, Mark Baker, Martin J. Dürst, Michael Cooper, Max Froumentin, Mikko Pohja, Mohamed Zergaoui, Ms2ger, Najib Tounsi, Noah Mendelsohn, Oguz Kupusoglu, Ola Andersson, Olli Immonen, Paddy Byers, Paul Libbrecht, Philipp Heltewig, Philip Taylor, Rainer Hillebrand, Robert Sayre, Rune F. Halvorsen, Samuel Santos, Scott Wilson, Sean Mullan, Sigbjorn Finne, Simon Pieters, Stephen Paul Weber, Stephen Jolly, Stephane Sire, Steven Faulkner, Thomas Landspurg, Thomas Roessler, Tiago Neves, William Edney, Yoan Blanc, Yves Savourel.

Special thanks go to Arve Bersvendsen, Robin Berjon, Anne van Kesteren, and Charles McCathieNeville who helped edit various versions of this specification. Big thanks also to Richard Tibbett.

Special thanks also to [David Håsäther](#) for creating and maintaining the [\[Widgets-Relax NG Schema\]](#) for the configuration document format.

Parts of this document reproduce text and behavior from the [\[HTML\]](#) specification and from the [XBL 2.0 specification](#) (as permitted by both specifications by their copyright statements).

Graphic icons used some examples of this specification were created by [Yusuke Kamiyamane](#) and are available for use under a [Creative Commons Attribution 3.0 license](#).

This specification is dedicated to the children of India.

Normative References

[ABNF]

[Augmented BNF for Syntax Specifications: ABNF](#). RFC5234. D. Crocker and P. Overell. January 2008.

[BIDI]

[Unicode Standard Annex #9: Unicode Bidirectional Algorithm](#). M. Davis.

[BCP47]

[Tags for Identifying Languages](#), A. Phillips and M. Davis. September 2009.

[CP437]

[IBM CPGID 00437](#). Code Page 47, Character Encoding Scheme. 1984.

[CSS]

[Cascading Style Sheets Level 2 Revision 1 \(CSS 2.1\) Specification](#). B. Bos, I. Hickson, T. Çelik, H. Wium Lie. W3C Recommendation 07 June 2011.

[CSS-MIME]

[The text/css Media Type](#). RFC 2318. H. Lie, B. Bos, and C. Lilley. IETF. March 1998.

[Deflate]

[DEFLATE Compressed Data Format Specification version 1.3](#). P. Deutsch, The Internet Society, May 1996.

[DOMCore]

[Document Object Model \(DOM\) Level 3 Core Specification](#). A. Le Hors, P. Le Hégarret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, editors. W3C Recommendation 07 April 2004.

[IANA-Charsets]

[IANA Character Set Registry](#).

[IRI]

[Internationalized Resource Identifiers \(IRIs\)](#). RFC3987, M. Duerst, M. Suignard. January 2005.

[MIME]

[Multipurpose Internet Mail Extensions \(MIME\) Part One: Format of Internet Message Bodies](#). RFC2045, N. Freed and N. Borenstein, IETF, November 1996.

[Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types](#). RFC2046, N. Freed and N. Borenstein, IETF, November 1996.

[RFC2119]

[Key words for use in RFCs to Indicate Requirement Levels](#). RFC2119, S. Bradner. March 1997.

[SNIFF]

[Media Type Sniffing](#). A. Barth and I. Hickson. IETF (Work in Progress).

[Unicode]

[The Unicode Standard](#). Unicode Consortium.

[URI]

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

[UTF-8]

[UTF-8: A Transformation format of ISO 1064](#). RFC 3629, F. Yergeau. IETF, November 2003.

[Widgets-DigSig]

[XML Digital Signatures for Widgets](#). M. Cáceres, Paddy Byers, Stuart Knightley, F. Hirsch, and M. Priestley. W3C Proposed Recommendation.

[View-Modes]

[The 'view-mode' Media Feature](#). R. Berjon and M. Cáceres. W3C Recommendation.

[XML]

[Extensible Markup Language \(XML\) 1.0 \(Fifth Edition\)](#). T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. W3C Recommendation 26 November 2008.

[XML-MIME]

[XML Media Types](#). RFC3023. M. Murata, S. St. Laurent, D. Kohn. IETF. January 2001.

[XMLNS]

[Namespaces in XML \(Second Edition\)](#). T. Bray, D. Hollander, A. Layman, R. Tobin. W3C Recommendation, August 2006.

[ZIP]

[Zip File Format Specification](#). PKWare Inc.

[ZIP-MIME]

[IANA Media Type Assignment](#).

Informative References

[ECMAScript-MIME]

[Scripting Media Types](#). RFC4329. B. Hoehrmann. IETF. April 2006.

[HTML]

[HTML5](#). Ian Hickson. W3C Working Draft 25 May 2011.

[HTML - Living Standard](#). I. Hickson. WHATWG.

[HTML-MIME]

[The 'text/html' Media Type](#). D. Connolly and L. Masinter. IETF. June 2000.

[HTTP]

[Hypertext Transfer Protocol -- HTTP/1.1](#). RFC 2616, R. Fielding, et al. June 1999.

[SVGTiny]

[Scalable Vector Graphics \(SVG\) Tiny 1.2 Specification](#). O. Andersson, R. Berjon, E. Dahlström, A. Emmons, J. Ferraiolo, A. Grasso, V. Hardy, S. Hayman, D. Jackson, C. Lilley, C. McCormack, A. Neumann, C. Northway, A. Quint, N. Ramani, D. Schepers, A. Shellshear. W3C Recommendation, 22 December 2008.

[UTR36]

[UTR #36: Unicode Security Considerations](#), M. Davis, M. Suignard. Unicode Consortium.

[Widgets-Landscape]

[The Widget Landscape \(Q1 2008\)](#). M. Cáceres. W3C Note.

[Widgets-Requirements]

[Widgets Requirements](#), M. Cáceres and M. Priestley. W3C Note.

[Widgets-Relax NG Schema]

[Relax NG Schema for the Widgets Family of Specifications](#). D. Håsäther, R. Berjon, and M. Cáceres.

[P&C-Test-Suite]

[Test Suite for Widget Packaging and XML Configuration](#). M. Cáceres.

[Widgets-APIs]

[The Widget Interface](#). M. Cáceres. (Work in progress).

[Widgets-URI]

[Widget URI Scheme](#). M. Cáceres. W3C Note.