Widgets 1.0: The Widget Landscape (Q1 2008)

W3C Working Draft 14 April 2008

This version:
http://www.w3.org/TR/2008/WD-widgets-land-20080414/
Latest version:
http://www.w3.org/TR/widgets-land/
Latest Editor's Draft:
http://dev.w3.org/2006/waf/widgets-land/
Previous version:
none.
Version history:
Twitter messages (non-editorial changes only): http://twitter.com/widgetspecs (RSS)
Editor:
Marcos Caceres, Invited Expert

Copyright © 2008 W3C® (MIT, ERCIM, Keio), All Rights Reserved. W3C liability, trademark and document use rules apply.

Abstract

This document surveys a group of market-leading widget user agents with the aim to inform the requirements of the [Widgets 1.0: Requirements] document. The survey exposes commonalities and fragmentation across widget user agents, and discusses how fragmentation currently affects, amongst other things, authoring, security, distribution and deployment, internationalisation and the device-independence of widgets. The document concludes by making a set of recommendations on what aspects of widgets require standardization to reduce fragmentation to ultimately standardize a cross-platform widget solution.

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 at http://www.w3.org/TR/.

This is the W3C First Public Working Draft of the *Widgets 1.0: Landscape*. Once all the comments about this document will have been addressed, the Working Group intends to publish a final version of this document as a W3C Working Group Note.

The W3C Membership and other interested parties are invited to send comments to public-appformats@w3.org, the W3C's public email list for issues related to Web Application Formats. Archives of the list are available.

This document is produced by the Web Application Formats WG, part of the Rich Web Clients Activity in the W3C Interaction Domain. It is expected that this document will become a Working Group Note. Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1. Introduction
    1.1 Purpose
2. Terms
3. Widgets and Widget User Agents
    3.1 Differences from Web Widgets
    3.2 Differences from Java Applets
4. Packaging for Distribution and Deployment
    4.1 Packaging Formats, file extensions and Media Types
5. Metadata and Configuration
    5.1 Metadata
        5.1.1 Fragmentation Issues
6. Authoring and Scripting
    6.1 APIs
    6.2 Widget object: properties and events
7. User interface and Accessibility
8. Instantiation
9. Internationalization and Localization
10. Digital Signatures
11. Automatic Updates
12. Device Independence

Tableau EX1065
IPR2024-01389
U.S.Patent No. 8,520,000

13. Security Models
14. Icons
15. Standardizable Aspects of Widgets
Acknowledgments
References
Related Sources

1. Introduction

This document surveys the widget landscape by examining how market-leading widget user agents address issues around:

- distribution and deployment,
- metadata and configuration,
- user interface and accessibility,
- authoring,
- internationalization and localization,
- device-independence,
- Initialization,
- automatic updates,
- and security.

The ***market-leading widget user agents*** that are included in the survey are listed below. The widget user agents were subjectively chosen because of their perceived prevalence in the market place. This survey was conducted independently of any vendor and no vendor explicitly requested they be included in the survey.

Market-Leading Widget User Agents

Widget User Agent	Vendor	Version	Platform
Konfabulator	Yahoo!	4.5	Windows XP, Windows Vista, MacOS
Windows Sidebar	Microsoft	1.0	Windows Vista
Google Desktop Gadgets	Google	1.x	Windows XP, Windows Vista
Opera Widgets	Opera	9.5 Beta	Mac OS 10.5, Windows XP, Windows Vista
Dashboard	Apple	1.1	Mac OS 10.5
Web-Runtime	Nokia	1.0 Beta	S60 3rd Edition, Feature Pack 2 (emulator)
Joost Widgets	Joost	1.0 Beta	Mac OS 10.5, Windows XP, Windows Vista
Widget User Agent	Vendor	Version	Platform

The inclusion of the widget user agents in this survey does not indicate endorsement of the standardization process by any particular vendor (in other words, just because a widget user agent appears should not be taken to mean that they will implement any part of the Widgets 1.0 specifications).

1.1 Purpose

The purpose of this document is to provide a holistic overview of the ***widget space*** and provide information to aid in standardization process of widgets by the Web Application Formats Working Group. As such, this document provides:

- A consistent set of terms that can be used throughout the standardization process, including in specifications.
- A definition of what constitutes a widget and a widget user agent for the sake of standardization (and what does not constitute a widget).
- A discussion of what role various technologies play in the lifecycle of a widget.
- Comparison matrices that clearly demonstrate fragmentation and interoperability across the widget landscape.
- A list of standardizable aspects of widgets.

2. Terms

This section defines some of the key terms related to widgets.

Widget:
: For the purpose of this standardization process, a widget is 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 or mobile device. A widget may run as a stand alone application (meaning it can run outside of a Web browser), or may be embedded into a Web document. In this document, the runtime environment on which a widget is run is referred to as a widget user agent and a running widget is referred to as an instantiated widget. Prior to instantiation, a widget exists as a widget resource.

Widget user agent:
: The user agent (software application) that hosts an instantiated widget. Generally speaking, widget user agents are either directly built on, or provide similar functionality to a Web browser. An increasing number are actually built directly on top of Web browsers so they are able to process/render [HTML] documents, while others incorporate Web browser components like ECMAScript interpreters. widget user agents are built for many different software platforms and devices, including Microsoft's Windows, Apple's MacOS, Symbian, Linux, and so on; and some widget user agents, such as the Konfabulator and Opera Widgets, run on multiple platforms.

Instantiated widget:
: The runtime manifestation of a decompressed widget resource whose start file has been instantiated on a widget user agent. The instantiated widget may have been configured through a configuration document. The ability for an instantiated widget to be programmed and behave interactively is provided via a widgets API.

Icon:
: An image or symbolic representation of an instantiated widget. Icons are usually used to represent the widget in non-running context, such as menus and docks. Some widget user agents, such as Konfabulator, allow authors to dynamically change the icon at runtime. For example, a weather widget might update its icon as the weather or time of day changes.

Widget resource:
: A resource created from some packaging format that encapsulates the resources of a widget for the purposes of distribution and deployment. On the wire, a widget resource is identified by an arbitrary widget media type.

Media type:
An [media type] that formally associates a widget resource with some proprietary widget user agent. For example, Joost's widget engines requires that widgets be served over HTTP with a `application/vnd.joost.joda-archive` media type.

Packaging format:
The physical data format used to create a widget resource. For example, the flat-file format described in the [Konfabulator Reference] or the [Zip] file format supported by Opera Widgets and Microsoft's Vista Sidebar.

Resource:
Any file or directory used by an instantiated widget that resides either inside a widget resource or is accessible over [HTTP]. In a widget resource, resources may be organized in directories and may have versions of those directories tailored for localization purposes. Examples of resources include images, text, markup, style sheets, executable scripts, and sounds.

Start file:
A resource either inside the widget resource or on the Web that when instantiated represents the widget. If an instantiated widget contains a configuration document, the widget user agent may configure the start file through that configuration document.

Configuration document:
A distinguished resource where authors can declare metadata and/or configuration parameters for a widget. A widget user agent uses a configuration document to configure a widget upon instantiation. The configuration document may also define the relationship between resources in the widget resource. The configuration document usually takes the form of an [XML] file; for example, the `config.xml` resource bundled with an Opera widget.

Metadata:
Data declared in the configuration document about a widget that relates to authorship or classification, but does not affect the behavior of the widget at runtime (eg. the author's name and email).

Configuration parameter:
Any declaration in the configuration document that affords the widget with functionality beyond its default behavior (eg. declaring that the widget will require network access).

Bootstrap:
A mechanism that either declaratively or automatically finds the start file in an instantiated widget.

Widget API:
A set of programming interfaces that provide functionality specific to and instantiated widget. Current APIs range extensively in the level of functionality they provide an author; see for example Microsoft's API for accessing the operating system in the [Sidebar Reference].

3. Widgets and Widget User Agents

Since around 2003, a relatively new kind of application has seen significant proliferation onto desktop computers and, more recently, web-enabled portable devices like mobile phones. This kind of application is generally referred to by developers as a *widget engine*: a piece of software that is able to run other smaller applications known as *widgets* or *gadgets*. On the user's desktop, widgets have gradually taken the place of some traditional single-purpose applications. Typical examples of widgets range from simple clocks, CPU gauges, post-it notes, games, battery-life indicators, to more sophisticated web-enabled widgets like weather forecasters, news readers, email checkers, photo albums and currency converters.

There are literally thousands of unique widgets available for download on the web, which users generally collect to create personal widget inventories. These widget inventories provide users with access to online services that they commonly use. This means that, in a lot of instances, users don't need to open up a web browser to get the information that they want (eg. to check the weather). This is an aspect of widgets that makes them particularly attractive on mobile devices, where the monetary cost of downloading web pages is currently an issue for many users.

For developers, some widgets differ from traditional binary applications in that they are created using the same open technologies used to create web pages. Widget engines mimic, in many ways, the behavior of web browsers: an increasing number are actually built directly on top of web browsers so they are able to render web pages, while others incorporate web browser components such as ECMAScript interpreters. To developers and vendors, this means that most widgets are significantly easier to create than applications developed with lower-level programming languages such as Java and C#.

Amidst the popular rise of widgets and widget engines lay a number of issues for users, developers, current vendors and new vendors wanting to enter the market. By surveying various aspects that pertain to widget user agents, this document discusses these issues so they may be resolved through the W3C standardization process.

As shown in figure 1, a widget is instantiated on a widget user agent and can make use of a number of technologies that serve a different role (eg. distribution and deployment, etc). However, some of those technologies have not yet been formally standardized (items marked with an asterisk), which has contributed to fragmentation across the widget space.

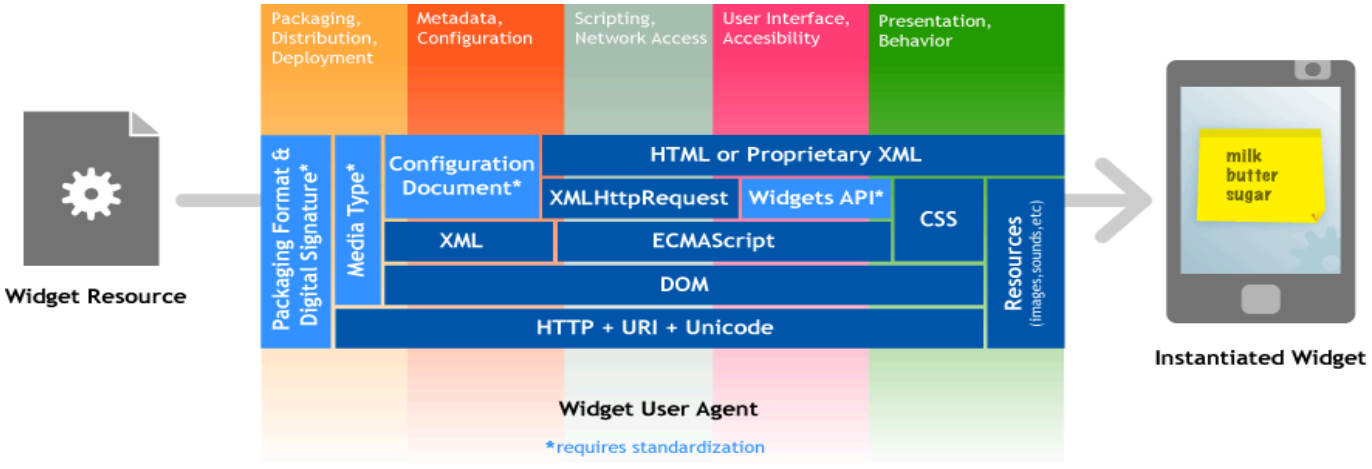

Figure 1. A typical widget technology stack and aspects that have require standardization. Please note that this technology stack is intended as a guide, and does not represent the technology stack of any particular user agent.

3.1 Differences from Web Widgets

Web widgets (also known as *modules* or *badges*) are fragments of [HTML], [CSS], and [ECMAScript] (or possibly an Adobe Flash movie) that are either declaratively or dynamically included into a Web document. A common example of Web widgets is one that downloads a set of icon-sized images from a photo-sharing Web site and displays those images as a slide-show based on a set of user preferences (eg. the images tagged 'vacation Italy'); such Web widgets are commonly seen embedded into social networking Web sites and blogs. Popular providers of Web widgets include:

- Google Homepage Gadgets (iGoogle Gadgets),
- WordPress Widgets,
- and Microsoft Live Gadgets.

Unlike widgets, Web widgets a hosted on the server-side and are embedded into HTML documents prior to being served to the client. The creation of a Web widget usually involves having an author specify, in XML or some other format (eg. PHP), what the widget does and which APIs the Web widget depends on. This document is then uploaded to a server, where when it is served to a client, it is transformed into HTML, CSS, and ECMAScript. For example, the input column of the table below shows a typical iGoogle gadget specification:

iGoogle Gadget Transformation Example

Input	Output

`<?xml version="1.0" encoding="UTF-8"?>`
`<Module>`
`<ModulePrefs title="hello world example" />`
`<Content type="html"><![CDATA[`
`<h1>Hello, world!</h1>`
`]]></Content>`
`</Module>`
	Hello World!

Input	Output

After being processed on the server-side, the code in the input column is transformed to HTML, CSS, and ECMAScript and inserted into the served document as either an `iframe` or as HTML elements (see the the Output column above). The actual code that Google generates from the example is too large to be included in this document.

Functional Differences

Because Web widgets and a widget have a reliance of Web technologies, their offer much of the same functionality. However, differences exist in:

- the packaging format,
- the security model,
- and the APIs.

In relation to the packaging format , Web widgets are generally not packaged or downloaded as a single file (except in the case of Adobe Flash movies). Instead, Web widgets are commonly dynamically instantiated through a mix of [ECMAScript], [HTML] elements, and [CSS]. However, similar to a widget as described in this document, some Web widgets make use of a dynamically loaded RSS file or [JSON] as a configuration document format.

In relation to security models, unlike a widget, Web widgets are generally part of a [HTML] document's DOM and so are bound to all the security constraints imposed by Web browsers.This means that Web widgets cannot make cross-domain requests, cannot autonomously access resources on an end-user's device, access system-level properties like the make, model, or usage percentage of the CPU, or execute system level commands like creating or deleting files, while widgets that run on most market-leading widget user agents generally can. In other words, some widget user agents provide a more relaxed security model than the one afforded to Web widgets by Web browsers.

The ability for a widget to perform actions beyond the security scope of Web widgets is partially afforded by widget-specific APIs. For example, on Windows Vista's Sidebar, a widget can be scripted to create a new folder on the end-user's hard drive by calling `'System.Shell.Folder.newFolder(strNewFolderName)'`. See Microsoft's [Sidebar Reference] or [Konfabulator Reference] for more examples of API functionality that is beyond the scope of Web widgets.

Another difference is how widget user agents handle internationalization when compared to Web widgets (Web Browsers). On the Web, internationalization is sometimes handled through [HTTP]'s `Accept-Language` header: this works by allowing the Web Browser to send the end-user's preferred language to a server (eg. `Accept-Language: en-us`). If the server contains a version of the desired resource in the end-user's language of choice, then the localized resource may be returned to the end-user. A widget, on the other hand, may sometimes contained all localized resources inside the widget resource in folders named using the common language-region pattern (eg. /en-us/). When the widget is instantiated, the widget user agent attempts to match one of these specially named folders to user's language preferences. See the Internationalization and Localization section for more information.

3.2 Differences from Java Applets

Widgets and Java applets share many commonalities. For instance, both widgets and applets rely on a pre-installed runtime engine for execution: java applets rely on the presence of the Java Runtime Environment (JRE), while widgets rely on the presence of their target widget engine. Widget and Java applets also share many similar functional aspects, like being able to do asynchronous HTTP requests to download resources from the Web.

It is argued that the most notable difference between them is that widgets are easier for authors to create than Java applets. This argument is made because widgets are created using HTML, CSS, and ECMAScript, which have very forgiving error handling and a short learning curve

compared to Java. Another difference is that Java Applets are intended to run inside Web pages, while widgets as described in this document generally serve the purpose of stand-alone applications that run outside of a Web browser.

4. Packaging for Distribution and Deployment

Packaging refers to encapsulating all the necessary resources and metadata required by the widget into a single file for the purpose of distribution and deployment. ***Distribution and deployment*** refers to getting a widget from the Web to run on an user's device as easily as possible.

4.1 Packaging Formats, file extensions and Media Types

The *de facto* standard for packaging widgets is the Zip file format, but with vendors requesting that their developers use a vendor specified file extension (ie. not .zip, but .widget, or .gadget, etc) when packaging their widgets.

Once a widget has been packaged for distribution, it is put onto a web server and served with an appropriate media type. The purpose of a media type is to allow a browser, for instance, to automatically associate a widget resource with the appropriate widget user agent. For example, widgets served for Operas widget engine are served with the `application/x-opera-widgets` media type and associated with the Opera browser. If a widget engine has correctly registered itself with the operatig system to be the program of choice to deal with a particular media type media type and/or file extension with, the web browser should automatically pass widgets to the widget engine without the end-user having to select the widget resource manually.

End-users generally acquire widget resources directly from vendors (eg. Apple, Yahoo!) who often host dedicated online galleries where users can search for widgets and where developers can submit or update widgets they have created. However, authors are free to distribute their widgets from their own web sites.

Packaging Format
: The packaging format is supported widget user agent.

Compression
: The compressions algorithm supported by the widget user agent.

File extension
: The file extensions that associates a widget with a widget user agent.

Media type
: As widgets are generally distributed via the Web, vendors usually assign them an arbitrary MIME type. The MIME type, which is usually used in conjunction with the file extension, helps a widget user agent associate the a widget with the appropriate widget user agent.

Packaging Formats, file extensions and Media Types

Widget User Agent	Packaging Format	Compression	File Extension	Media type
Konfabulator	Proprietary flat-file, Zip	Deflate (Zip)	.widget (not enforced)	`application/vnd.yahoo.widget`
Windows Sidebar	Zip, Cab, directory	Deflate	.gadget	`application/x-windows-gadget`
Google Desktop	Zip	Deflate	.gg	`app/gg`
Opera Browser	Zip	Deflate	.zip	`application/x-opera-widgets`
Apple Dashboard	Zip	Deflate	.wdgt or .zip	`application/x-macbinary`
Nokia Web-Runtime	Zip	Deflate	.wgz	`application/x-nokia-widgets`
Joost Widgets	Zip	Deflate	.joda (not enforced)	`application/vnd.joost.joda-archive`
Widget User Agent	Packaging Format	Compression	File Extension	Media type

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Interoperable Aspects

- Zip format
- Deflate compression

Fragmentation Issues

- There is no consistent file extension for widgets; each vendor uses their own unique file extension.
- Nowhere is it defined exactly which parts of the Zip specification should be supported by a widget user agent. Zip is a large specification that is constantly evolving. Zip supports multiple compression algorithms and features. If a widget is packaged using the wrong compression algorithm, it might not be interoperable with another widget engine or device. Although the issue of compression is not currently a major issue, it has the potential to become one as widgets become more prevalent on mobile devices and computers start to work natively in 64-bit.
- There is no standardized Media Type, which results in widgets served over HTTP being associated with only one kind of widget engine. One media type per widget engine results in vendor lock-in.

5. Metadata and Configuration

A widget resource will typically include a configuration document, in which an author declares metadata and/or some configuration parameters that a widget user agent may use to configure a widget upon instantiation. All market leading widget engines use XML as the preferred format for storing metadata.

XML vocabulary
: The proprietary specification that defines the semantics of the elements and attributes that authors should use when marking up a configuration document.

File name
: The name of the file as

Required?
: Is the configuration document required for the widget user agent to instantiate the widget. This was tested by attempting to instantiate a widget without a configuration document present inside the widget resource.

Uses XMLNS
Does the configuration document require authors to declare a namespace. Also tested using a bogus namespace on the root element xmlns=""
Conforming Parser?
Is the XML parser used by the widget user agent conformant to the XML specification and XMLNS aware?

Configuration documents

Widget User Agent	XML vocabulary	File Name	Required?	Uses XMLNS?	Conforming Parser?
Konfabulator	Konfabulator Reference	widget.xml	no	no	
Windows Sidebar	Microsoft Gadgets	gadget.xml	yes	no	
Google Desktop	Google Gadgets	gadget.gmanifest		no	
Opera Widgets	Opera Spec	config.xml	yes	yes, but not required. If a bogus namespace is given, the widget will not work.	
Apple Dashboard	Apple plist	Info.plist	Yes	no	
Nokia Web-Runtime	Apple plist	Info.plist	Yes	no	
Joost Widgets	[Joost Joda]	config.xml	yes	yes, but not required (will work with any given namespace).	
Widget User Agent	XML vocabulary	File Name	Required?	Uses XMLNS?	

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authorities information about any particular widget user agent, please visit the vendor's web site.*

Interoperable Aspects

- XML is the preferred format for structuring metadata.

Fragmentation Issues

- The XML dialects are all different.
- XML parsers are generally non-conforming to the XML and XMLNS specifications.

5.1 Metadata

Metadata refers to how authors store information about their widget inside the widget, and how that data is made accessible to people or machines.

In practice, the inclusion of any metadata elements is generally considered optional but is nonetheless recommended by vendors. Widget user agents generally make use of metadata elements in different application contexts such as menus, lists, and about-boxes. The most common metadata elements captured in configuration documents include:

Root element:
The element found at the root of each configuration document, which contains all other elements in the document.
Name:
The human readable name of the widget as it appears in menus or other contexts.
Unique identifier:
An author assigned unique identifier for the widget.
Version identifier:
Some string that identifies the version of the widget.
Description:
A human readable description of the widget, generally describing what it does.
Copyright:
Copyright information.
Authorship:
information about the authorship of the widget, including the author's name, email, and the organization that they may be affiliated with

Metadata elements and their roles

Widget User Agent	Root Element	Name	Unique Identifier	Version Identifier
Konfabulator	`<metadata>`	`<name>text</name>`	`<identifier>UUID or RD</identifier>`	`<version>text</version>`
Windows Vista Sidebar	`<gadget>`	`<name>text</name>`	Uses `<name>` and `<version>`.	`<version>text(n.n.n.n)</version>`
Google Desktop	`<gadget>`	`<name>text</name>`	`<id>UUID</id>`	`<version>text(n.n.n.n)</version>`
Opera Widgets	`<widget>`	`<widgetname>text</widgetname>`	`<id><host>URI</host> <name>text</name><id>`	`<id><revised>W3CDTF</revised></id>`
Apple Dashboard	`<plist version="1.0">`	`<key>CFBundleDisplayName</key> <string>text</string>`	`<key>CFBundleIdentifier</key> <string>RD</string>`	`<key>CFBundleVersion</key> <string>string</string>`
Nokia Web-Runtime	`<plist version="1.0">`	`<key>CFBundleDisplayName</key> <string>text</string>`	`<key>Identifier</key> <string>RD</string>`	`<key>Version</key> <string>string</string>`
Joost Widgets	`<widget-manifest>`	`<name>text</name>`	`<id>URI</id>`	none.
Widget User Agent	Root Element	Name	Unique Identifier	Version Identifier

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Metadata elements and their roles (continued)

Widget User Agent	Description	Authorship	Copyright
Konfabulator	`<description>Text</description>`	`<author name="" organization="" href=""/>`	`<copyright>text</copyright>`
Widget User Agent	Description	Authorship	Copyright

Widget User Agent	Description	Authorship	Copyright
Windows Vista Sidebar	`<description>Text</description>`	`<author name=""> <info url="URL"/> <logo src="rel-path"/> </author>`	`<copyright>text</copyright>`
Google Desktop	`<description>Text</description>`	`<author>text</author> <authorEmail>text</authorEmai> <authorWebsite>URL </authorWebsite>`	`<copyright>text</copyright>`
Opera Widgets	`<description>Text</description>`	`<author> <name>text</name> <email>text</email> <link>text</link> <organization>text</organization> </author>`	none.
Apple Dashboard	none.	none.	none.
Nokia Web-Runtime	none.	none.	none.
Joost Widgets	none.	`<web site.>URI</web site.>`	none. Often included as an xml comment inside the configuration document.
Widget User Agent	Description	Authorship	Copyright

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Interoperable Aspects

- Although element names differ, the semantics captured by elements is relatively consistent.

5.1.1 Fragmentation Issues

- There some fragmentation in regards to recording the description, authorship information and copyright.

Configuration parameters

The most common configuration parameters include:

Bootstrap:
A way to identify the start file (or main content), including a way to identify the content type of the start file (eg. type="HTML").

Network:
The need for a widget to access the network.
Width and height:
The initial rendering dimensions (width, height).

Plugins:
The intention to use plugins (eg. Flash and Java).
Platform:
The minimum version of the widget user agent required to run the widget.

Please note that some configuration parameters have a close relationship to the security model of widgets.

	Configuration documents				
Widget User Agent	Bootstrap	Width and Height	Network	Plugins	Platform
Konfabulator	Not declared. First *.kon file encountered is treated as the start file.	none.	`<security> <http name="someSite">exemple.com</http> </security>`	none.	`<platform minVersio os="macintosh|windo`
Windows Sidebar	`<host> <base type="HTML" src="rel-path/file" /> </host>`	none.	none. Allowed by default.	none. Allowed by default.	`<host name="sidebar minPlatformVersion`
Google Desktop	Not declared. Root of container must have a "main.xml" file which serves as the start file.	none.	none. Allowed by default.	none.	`<gadget minimumGoogleDeskto <platform> <windows minimumGadgetHostVe <mac minimumGadgetHostVe </platform> </gadget>`
Opera Widgets	None. The start file must be called "index.html" and must be at the root of the archive.	`<width>n</width> <height>n</height>`	`<security> <access> <protocol>http|ftp</protocol> <host>IP Address|domain name</host> <port>integer|integer-range(eg 1200-1500)</port> </access> </security>`	`<content> <plugins>yes|no</plugins> <java>yes|no</java></content>`	
Apple Dashboard	`<key>MainHTML</key> <string>rel-path/any.html</string>`	`<key>Width</key> <integer>n</integer> <key>Height</key> <integer>n</integer>` When not present, the width and height of Default.png is used.		`<key>AllowFullAccess</key> <true|false/>`	
Nokia Web-Runtime	`<key>MainHTML</key> <string>rel-path/any.html</string>`		`<key>AllowNetworkAccess</key> <true|false/> <key>AllowFullAccess</key> <true|false/>`	none.	
Widget User Agent	Bootstrap	Width and Height			

Widget User Agent	Bootstrap	Width and Height	Network	Plugins	Platform
Joost Widgets	`<main-file>rel-path/a.[jwl,html,svg]</main-file>`		`<key>AllowNetworkAccess</key>` `<true|false/>`	`<key>AllowInternetPlugins</key>` `<true|false/>`	
Widget User Agent	Bootstrap	Width and Height			

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

TO DO: TBW

6. Authoring and Scripting

Authoring refers to how widgets are created, marked-up and scripted. In terms of authoring, there is a fairly congruent set of commonalities that most widget user agents share, and which authors exploit when authoring a widget: mainly their reliance on Web standards and protocols, and a strong focus on rapid development. Most widget user agents will typically support [HTTP], [IRI]s, and [Unicode], as well as [ECMAScript], the DOM, and the ability to render markup languages, like [HTML] and [CSS]. Widget user agents also generally support multimedia resources, such as images, sounds, and some even video.

To make authoring of widgets possible, widget user agents provide authors with Application Programming Interfaces (APIs) that are mostly identical to those found in Web browsers, as well as APIs that provide functionality that is specific to widgets. Also, because of the rise in popularity of [Ajax]-style development, many widget user agents now implement the [XMLHttpRequest] object or some similar mechanism for making asynchronous data requests over [HTTP].

	Supported technologies													
Widget User Agent	png	gif87	gif89	jpeg	png	svg	mp3	wav	swf	css	js	html4	canvas	XHR
Konfabulator	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes
Windows Vista Sidebar	yes	yes	yes	yes	yes	no			yes	yes	yes	yes	no	yes
Google Desktop	yes	yes	yes	yes	yes	no	yes		no	no	yes	no	no	yes
Opera Widgets	yes	yes	yes	yes	yes	yes			yes	yes	yes	yes	yes	yes
Apple Dashboard	yes	yes	yes	yes	yes	yes			yes	yes	yes	yes	yes	yes
Nokia Web Runtime	yes	yes	yes	yes	yes	no			no	yes	yes	yes	no	yes
Joost Widgets	yes	yes	yes	yes	yes	yes			?	yes	yes	yes	yes	yes
Widget User Agent	png	gif87	gif89	jpeg	png	svg	mp3	wav	swf	css	js	html4	canvas	XHR

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

TO DO: explain functionality exclusively available to widgets (run programs, cross domain requests)

Interoperable Aspects

- All widget user agents support XMLHttpRequest.
- All but one support HTML and CSS as layout

Fragmentation Issues

- There is some limited fragmantation is regards to support of proprietary technologies, such as Flash.

6.1 APIs

TO DO: TBW

Open page in browser
: Open a web document in the standard system browser

Preferences
: Get, set and delete user preferences

Close widget
: Destroy instance of currently running widget

	Supported technologies		
Widget User Agent	Open page in browser	Preferences	Close widget
Konfabulator	`void openURL(url)`		
Widget User Agent			

Widget User Agent	Open page in browser	Preferences	Close widget

Windows Vista Sidebar	no method, use <a href=""> element, or using javascript within the document: `window.open(url)`	`System.Gadget.Settings.write(String name, Object Value)`
		`System.Gadget.Settings.writeString(String name, String Value)`
		`System.Gadget.Settings.read(strName)`
		`System.Gadget.Settings.readString(strName)`
			`System.Gadget.close()`

Google Desktop	no method, use <a href=""> element		

Opera Widgets	`openURL(String url)`	`widget.setPreferenceForKey(preference, key)`
		`widget.preferenceForKey(key)`
		`setPreferenceForKey(String preference, null)`	

Apple Dashboard	`openURL(String url)`	`widget.setPreferenceForKey(preference, key)`
		`widget.preferenceForKey(key)`
		`setPreferenceForKey(String preference, null)`	

Nokia Web Runtime	`openURL(String url)`	`widget.setPreferenceForKey(preference, key)`
		`widget.preferenceForKey(key)`
		`setPreferenceForKey(String preference, null)`	

Joost Widgets	`navigate(String url);`	widget.setString(String name, String vlaue);	

Widget User Agent

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

TO DO: TBW

6.2 Widget object: properties and events

6.2.1 Properties of the Widget object

Properties of the widget object

Property	Konfabulator	Windows Sidebar	Google Desktop	Opera Widgets	Apple Dashboard	Nokia Web-Runtime	Joost Widgets
locale information	locale			window.navigator.language			window.navigator.lang
Engine version needed to run	requiredEngineVersion	System.Gadget.platformVersion					
If the widget is visible	visible	System.Gadget.visible					
The version of the widget as specified in the configuration document	version	System.Gadget.version					
The name of the widget as specified in the configuration document	name	System.Gadget.name					
The details of the author as specified in the configuration document	widget.author widget.company						
The copyright declaration as in the configuration document	widget.copyright						
Access the unique identifier for the widget					**String widget.identifier** This read-only property contains a string value that is unique among all of the instances of a single widget. This value is assigned by	string widget.identifier;	

Property	Konfabulator	Windows Sidebar	Google Desktop	Opera Widgets	Apple Dashboard	Nokia Web-Runtime	Joost Widgets
					Dashboard and persists between instantiations of each widget instance.		
	requiredPlatform	Document, opacity, path, settingsUI, docked, background					

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

6.2.2 Events

TO DO: TBW

Events of the widget object

Event	Konfabulator	Windows Sidebar	Google Desktop	Opera Widgets	Apple Dashboard	Nokia Web-Runtime	Joost Widgets
Widget has loaded	widget.onLoad						
WUA has focus					widget.onshow		
WUA lost focus					widget.onhide		
Widget focus	widget.onGainFocus				window.onfocus		
Widget lost focus	widget.onLoseFocus				window.onblur		
Widget drag start	widget.onMouseDrag				widget.ondragstart		
Widget is being dragged							
Widget drag end					widget.ondragend		
Widget is removed from WUA	widget.onUnload				widget.onremove		
Cross widget communication	widget.onTellWidget = function(){ }						
Other	dockOpen onDockClosed onDockOpened onPreferencesChanged onRunCommandInBgComplete onScreenChanged onTellWidget onWakeFromSleep onWillChangePreferences						

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Accessing the file system

TO DO: TBW

Execute system level commands or open applications.

```
//If no callbackHandler is present, the command is executed synchronously.
//If callbackHandler is present, command is executed asynchronously. callbackHandler needs to except an argument.
CommandObject widget.system(String command, [Function callbackHandler])

interface CommandObject{
        String property outputString; //the standard out
        String errorString; //any error that was thrown
        String status. //the command's exit status

        // allows handler to receive incremental output (eg ping.)
        EventListener onreadoutput(function handler);
        function cancel(); // cancel the command
        function write(String value); //write a string to standard in
        function close(); //send (EOF) to standard in
}


void widget.openApplication(HexNum Uid, String param)

opens an S60 application in stand-alone mode. Values cannot passed back from the opened application to the widget.
```

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

TO DO: TBW

7. User interface and Accessibility

Accessibility refers to how end-users, and those with special needs, can access the content and use the interactive elements of an instantiated widget. Most market-leading widget user agents allow widgets to be authored using [HTML], [CSS], and [ECMAScript]. [HTML], when authored with care, is generally regarded to be an accessible technology whose structure and semantics are generally well understood and correctly implemented by most market-leading widget user agents. To extend It is also therefore theoretically possible for authors to make their widgets fairly accessible by applying, for example, the relevant sections of the *Web Content Accessibility Guidelines* ([WCAG]).

Language used to declare the user interface of a widget

Widget User Agent	UI Markup	HTML Renderer
Konfabulator	HTML + CSS (through Webkit), Proprietary XML	webkit
Windows Vista Sidebar	HTML + CSS + Proprietary XML	Internet Explorer
Google Desktop	Proprietary XML	none
Opera Widgets	(X)HTML + CSS + SVG	Opera
Apple Dashboard	HTML + CSS + SVG	Webkit/safari
Nokia Web Runtime	HTML + CSS	Webkit
Joost Widgets	XHTML + CSS + SVG + Proprietary XML	Gecko
Widget User Agent	UI Markup	

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

TO DO: TBW

8. Instantiation

What addressing mechanism does the widget user agent support at runtime to get to resources inside the package? (ie. how does a developer address and include things like images or sounds in their widget?)

The details of configuration document

Widget User Agent	Other expected files
Konfabulator	at least one *.kon somewhere in the archive
Windows Sidebar	at least one html file somewhere in the archive
Google Desktop	
Opera Widgets	(any folder)* index.html. The folder is selected in alphabetical order, not by the order in which it appears physically in the archive.
Apple Dashboard	some [name].html file identified as the start file by the `MainHTML` key in the Info.plist file. Icon.png and Default.png. Icon.png is used in the Dashboard bar. Default.png is shown while the widget loads and used to set the render context of the widget if width and height keys were not set in Info.plist.
Nokia Web-Runtime	some [name].html file identified as the start file by the `MainHTML` key in the Info.plist file. Icon.png is optional.
Joost Widgets	
Widget User Agent	Other expected files

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

TO DO: TBW: discusses different ways that the start file of widget is located: 1. having a preset name (eg. index.html ala Opera Widgets). Having it declared in the configuration document (ala Joost), Having it match a search pattern (ala Konfabulator)

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

TO DO: TBW

9. Internationalization and Localization

internationalization refers to the technical aspects that allow a widget to work in “multilingual contexts without requiring an author to make significant engineering changes to a widget.” ***Localization*** is the processes where by a widget is adapted to use the local conventions of particular end-users (eg. using a particular dialect)“.” To allow authors to distribute a widget to multiple locales, the majority of vendors provide guidelines explain to authors how to make effective use of internationalization mechanisms built into widget user agents. When authors follow localization guidelines, a widget user agent can use automated mechanisms to select the appropriate localized resources for a widget based on a system's locale information; thus making it easier for authors to create localized widgets.

In the widget space, internationalization is commonly achieved in one of two ways:

- One way is to have authors place localized resources into specifically named folders that identify the language and region (eg. "`/en-au/`" for English Australian) of localized content. If the widget user agent can identify the end-user's locale, and the widget package contains matching localized content, then a localized widget will be presented to the end-user.
- Another way is to have authors place localized textual content inside a specifically named text file, which is in turn placed inside specially named folders (eg. "`/en-au/Localizable.strings`"). Inside the text file, localized strings are identified by an unique identifier and must be formatted using a special syntax. For instance, Konfabulator uses a special syntax for localizing strings, for example:

```
"greeting" = "g'day mate!";
"greeting_gfx" = "/en-au/images/greet.png";
```

Note that this method only localizes textual content, but can be used to also identify the path or URI to localized resources. At runtime, the widget user agent makes those localized strings available via a scripting interface:

```
//would set the button's label to "g'day mate!"
myButton.label = widget.getLocalizedString("greeting");

myButton.style.background = widget.getLocalizedString("greeting_gfx");
```

When automated internationalization is not provided by a widget user agent, authors usually result to using arbitrary solutions to achieve localization. Such is the case for Opera widgets.

Localization strategies

Widget User Agent	Localization Strategy	Automatic	Convention followed	Example	Comments
Konfabulator	Directory-based + JS	yes		`/en-au/Localizable.strings`	
Windows Sidebar	Directory-based + JS	yes			
Google Desktop	Directory-based + JS	yes			
Opera Widgets	none	no	none		Does not support any automated localization strategies.
Apple Dashboard	Directory-based + JS	yes		/en-au/	
Widget User Agent	Localization Strategy	Automatic	Convention followed	Example	Comments

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

TO DO: TBW

10. Digital Signatures

A ***digital signature*** is the product of applying a secret numeric key (known as a *private key*) and an encryption algorithm over some data to produce a unique *hash value*. The only way to validate a digital signature is to use a corresponding *public key* in a process known as *asymmetric cryptography*.

Although not widely supported by market-leading widget user agents, some widget user agents allow an author to digitally sign a widget resource. Digitally signing a widget resource involves calculating the hash values of all the resources inside the widget resource and encrypting those values using a unique private key that is known only to the author. The resulting data is bundled inside the widget resource with a digital certificate, which an author can obtain from a certification authority (such as VeriSign) for a charge.

A digital certificate is therefore a trust mechanism intended to verify to a user that an author really did sign the widget resource. A widget user agent can use the digital certificate to verify the authenticity and data integrity of the widget resource. In the rare case where a widget damages the end-user's device, the digital certificate provides a user with a legal means to prove that a widget resource was signed by a particular author or publisher.

Because digital certificates can come from untrusted sources, widget user agents will include *root certificates* from sources that it trusts. Root certificates are explicitly trusted and are considered impossible to forge. For example, the Konfabulator is bundled with the Yahoo! root certificate which it uses to verify widgets signed by Yahoo! Inc. are in fact signed by Yahoo! Inc.

Digital Signatures

Widget User Agent	Signature	Certificate Format
Konfabulator	yes	x509

Windows Sidebar	yes	x509
Google Desktop	no	-
Opera Widgets	no	-
Apple Dashboard	no	-
Nokia Web Runtime	Planned	-
Joost	Planned	-
Widget User Agent	**Signature**	**Certificate Format**

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

Interoperable Aspects

- The use of X509 certificates.

Fragmentation Issues

- General lack of support for digital signatures.

11. Automatic Updates

Automatic updates refers to a service that allows a widget user agent to automatically check if a new or different version of an installed widget resource has become available, and somehow acquire the new version and install it. Automatic updates models work by allowing authors to assign a unique identifier and version identifier to a widget resource (eg. `id="http://example.com/my.wdgt" version="1.2"`).

To keep a widget resource up to date, the widget user agent periodically checks if a different version of a widget resource has become available on a remote server. In the case of the Konfabulator, it does this by sending an [XML] document that identifies the widget via an unique identifier, and what version the end-user currently has installed. If a new version of the widget resource is available on the server, the server sends back an [XML] file that contains a URL from where the widget user agent can acquire the latest version. The widget user agent then informs the end-user that an update to a widget has become available and if they want to perform the update. If the end-user agrees, then the widget user agent downloads the latest version of the widget resource, archives the old version, installs the new one in its place. The updated widget is re-instantiated with its preexisting preferences (eg. an updated weather forecaster widget will generally not require the end-user to re-input their preferred city after an update).

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

- General lack of support for automatic updates.

12. Device Independence

TO DO: Need to read up on how Nokia Web Runtime recommends authors deal with migration of Dashboard widgets, etc.

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

TO DO: TBW

13. Security Models

Security model refers to the security policies under which a widget user agent operates that either prevents or allows an instantiated widget from performing particular actions (eg. accessing the network or reading/writing files). When compared to Web browsers, some market-leading widget user agents have a comparatively relaxed security model that allows an instantiated widget to read, write, modify, and/or delete files, automatically upload files, automatically download files, execute local applications, and even perform cross-domain request to "mash-up" data from multiple different sources. All without the end-user having any indication that their privacy and security might be at risk.

The ability to perform most of the aforementioned actions is strictly forbidden in documents running in Web Browsers. Such a relaxed security model has been generally positive in the widget space with very useful and powerful widgets being developed. However, this has created an environment where users are left extremely vulnerable to malicious attacks, and serious security risks have been demonstrated. Some market-leading widget user agents, such as Opera Widgets, have a much tighter security model that adheres more closely to the security model of a Web Browser; as such, they may be less prone to serious security issues.

TO DO: Need to discuss how security declarations are made using pInfo list or Opera's security element.

Interoperable Aspects

TO DO: TBW

Fragmentation Issues

TO DO: TBW

14. Icons

			Icons	
Widget User Agent	Element in configuration file	Supported Types	Preferred	Comments
Konfabulator	`<image usage="dock|security" src="images/some.png"/>`	GIF, PNG.		Two images may be declared, depending on the `usage` attribute. The `usage` attribute allows either values `dock|security` to indicate where the image should be used. Security image is shown as part of the widget security warning when the widget is instantiated.
Windows Vista Sidebar	<icons> <icon src="rel-path" [width="" height=""]> </icons> <hosts> <host> <defaultImage src=""/> </host>	any GDI+ 1.0 supported format.		Having multiple icon elements allows the engine to select the icon most appropriate for communication based on size. Preferred size is 64px*64px, but any size is ok. TO DO: The documentation contradicts itself in regards to icon and defaultImage. Need to verify which one is actually used!
Google Desktop	<small-icon>rel-path/some.png<small-icon> <icon>rel-path/some.png<small-icon>	PNG	PNG	TO DO: Need to test other formats
Opera Widgets	<icon>relative-path/some.png</icon>	GIF, PNG.		
Apple Dashboard	none.	PNG		Not declared. An optional icon must appear in the root of the archive and must be called icon.png. If the icon is missing, then the runtime will use a default icon.
Nokia Web-Runtime	none.	PNG	PNG	include an "icon.png" file at the root of the widget.
Joost Widgets	<icon>relative-path/some.svg</icon>	SVG, PNG, JPEG, GIF		SVG or PNG are preferred.
Widget User Agent	Element	Supported Types		Comments

**Although care has been take to ensure the accuracy of the information contained in this table, there is no guarantee that the information is complete, correct, or up-to-date. To obtain authoritative information about any particular widget user agent, please visit the vendor's web site.*

TO DO: This section will describe how icons are used by different widget engines. It will discuss static icons (images, pngs), and dynamic icons, such as Yahoo!'s icons. Might also talk briefly about iPod/Phone icons here too, as they are dynamic.

Interoperable Aspects

PNG, GIF87/89

TO DO: TBW

Fragmentation Issues

TO DO: TBW

15. Standardizable Aspects of Widgets

The following list represents the aspects of a widget that members of the working group have identified as requiring standardization to reduce fragmentation in the widget space. Aspects that are currently outside the scope of the working group charter are proceeded by the text "***out of scope***". To address aspects beyond the scope of the working group, the working group will require liaison with other working groups at the W3C and possibly other related consortia such as the OMA and the OAA.

Standardizable aspects of widgets include:

- Finding a suitable packaging format capable of encapsulating and structuring resources for distribution and deployment, including:
  - The relevant technical aspects of the physical packaging format that make the format interoperable across multiple platforms and mobile devices.

    - The abstract container, including required parts and hierarchies (eg. required files and folders, if any).
    - The model by which the internal structure of a widget resource can be exploited by an instantiated widget for localization purposes in internationalized contexts.
    - A means for an instantiated widget to address resources in a widget resource at runtime.
    - A file extension.
    - A MIME type to formally denote that a widget resource distributed over [HTTP] conforms to the Widgets 1.0: Packaging specification.
    - A widely supported digital signature format that adequately provides data security, authenticity and non-repudiation.
- The configuration document language including:
    - The structure, semantics, and processing model for a vocabulary that would make up the configuration document format.
    - Metadata elements pertaining to authorship (eg. author's name, email, etc).
    - Metadata elements pertaining to the widget (eg. id, title, description, etc).
    - Configuration parameters (eg. width, height, network access, etc).
    - A bootstrap mechanism that allows the widget user agent to find the start file of a widget resource.
    - A model for finding a start file of a widget resource when a bootstrap is unavailable or is in error.
    - A means for distinguishing the configuration document from other resources.
    - A means to declare an alternative representation of a widget (eg. an image icon) for when a widget has not been instantiated.
- A widgets API that could be implemented by a widget user agent and made available to an instantiated widget that would allow authors to:
    - Access preferences particular to each instantiated widget.
    - Access runtime configuration properties and other relevant platform properties (**out of scope**).
    - Access to services particular to the device on which the widget has been instantiated (eg. camera, short message service, address book, etc) (**out of scope**).
    - The ability to instantiate other applications on an end-user's device (**out of scope**).
    - Access metadata values that the author declared in the configuration document.
    - Capture events and access properties that are particular to each instantiated widget.
    - Control alternative runtime representations of a widget (eg. the docked representation).
- The security model/policies that determines what an instantiated widget can access while running on a end-user's device.
- A pre-existing language to declare the user interface in an accessible manner (eg. mandating the use of [HTML] or some other [XML] vocabulary).
- A [HTTP]-based model for a widget user agent to check if an updated version of a widget resource has become available for download.

Standardizable aspects of widget engines include:

- TO DO: TBW

Acknowledgments

The editor would particularly like to thank Corin Edwards for his help on improving the design on of figure 1.

The editor would like to thank to the following people who have contributed to this document (ordered by first name):

- Alexander Dreiling
- Anne van Kesteren
- Arthur Barstow
- Arun Ranganathan
- Benoit Suzanne
- Bert Bos
- Bradford Lassey
- Cameron McCormack
- Cliff Schmidt
- Claudio
- Coach Wei
- Corin Edwards
- Dan Brickley
- David Pollington
- Dean Jackson
- Debra Polson
- Doug Schepers
- Ed Voas
- Gene Vayngrib
- Guido Grassel
- Jay Sweeney
- Jim Ley
- Jose Manuel Cantera Fonseca
- Kevin Lawver
- Krzysztof Maczyński
- Lachlan Hunt
- Marc Silbey
- Mark Baker
- Mikko Pohja
- Philipp Heltewig
- Stephen Paul Weber
- Thomas Landspurg
- Yang Wong
- Zachary Fitz-Walter

References

[*Ajax*]

Ajax: A New Approach to Web Applications. J. J. Garrett. February 18, 2005. Adaptive Path. Available at http://www.adaptivepath.com/publications/essays/archives/000385.php

[*Apple pList*]
Introduction to Property List Programming Topics for Core Foundation, Apple Computer Inc, 7 February 2006. Available at http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFPropertyLists/index.html

[*CSS*]
Cascading Style Sheets, level 2, revision 1, B. Bos, T. Çelik, I. Hickson, and H. Wium Lie. W3C Candidate Recommendation 19 July 2007. Available at http://www.w3.org/TR/CSS21

[*DOM*]
Document Object Model (DOM) Level 1 Specification, L. Wood et al., 1 October 1998. Available at http://www.w3.org/TR/REC-DOM-Level-1

[*Dashboard Reference*]
Dashboard Reference, Apple Computer, Inc, May 2006. Available at http://developer.apple.com/documentation/AppleApplications/Reference/Dashboard_Ref/index.html

[*Google Gadgets*]
Google Desktop Sidebar Scripting API, Google Inc., 2006. Available at http://desktop.google.com/script.html

[*JSON*]
The application/json media type for ECMAScript Object Notation. D. Crockford. July 2006. Available at http://www.ietf.org/rfc/rfc4627.txt

[ ]

[*Opera Spec*]
Opera Widgets Specification 1.0, A. Bersvendsen (Editor), Opera Software, 30 Apr, 2007. Available at http://oxine.opera.com/widgets/documentation/widget-configuration.html

[*Sidebar Reference*]
Windows Sidebar Reference, Microsoft Corporation, 2006. Available at http://msdn2.microsoft.com/en-us/library/aa965853.aspx

[*XML Internationalization and Localization*]
XML Internationalization and Localization. Savourel, Y. Sams Publishing, Indiana. June 2001.

[*Konfabulator Reference*]
Konfabulator Reference 4.5 Reference Manual Yahoo! Inc., April 14, 2006. Available at http://Widgets.yahoo.com/gallery/dl_item.php?item=WidgetEngineReference_3.1.1.pdf

[*WCAG*]
Web Content Accessibility Guidelines 1.0. W. Chisholm, G. Vanderheiden, and I. Jacobs. W3C Recommendation, 5 May 1999. Available at http://www.w3.org/TR/WAI-WEBCONTENT/

[*ECMAScript*]
ECMAScript Language Specification, Third Edition. ECMA, December 1999. Available at http://www.ecma-international.org/publications/standards/Ecma-262.htm

[*HTML*]
HTML 4.01 Specification, D. Raggett, A. Le Hors, I. Jacobs, 24 December 1999. Available at http://www.w3.org/TR/html401/

[*HTTP*]
Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt

[*MIME Type*]
Multipurpose Internet Mail Extensions (MIME) Part Two: media types, N. Freed and N. Borenstein, November 1996. Available at http://www.ietf.org/rfc/rfc2046.txt.

[*Unicode*]
The Unicode Standard, The Unicode Consortium, Version 5.

[*XML*]
Extensible Markup Language (XML) 1.0 Specification (Second Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, 6 October 2000. Available at http://www.w3.org/TR/REC-xml/

[*XMLHttpRequest*]
The XMLHttpRequest object. A. van Kesteren. 2006. W3C Working Draft, Available at http://www.w3.org/TR/XMLHttpRequest/

[*X.509*]
CCITT, *Recommendation X.509: The Directory Authentication Framework*, 1988.

[*IRI*]
Internationalized resource Identifiers (IRIs), M. Duerst, M. Suignard. IETF, January 2005. RFC3987 is Available at http://www.ietf.org/rfc/rfc3987

[*Zip*]
.ZIP File Format Specification. PKWare Inc., September 2007. Available at http://www.pkware.com/documents/casestudies/APPNOTE.TXT

Related Sources

[ ]

[*Light Web Applications*]
Setting the scope for light-weight Web-based applications. B. Bos. Work in Progress. 26 Feb 2004. Available at http://www.w3.org/People/Bos/Webapps.html

[*XML Packaging*]
XML Packaging Working Group Charter, J. Nava. W3C. Available at http://www.w3.org/XML/2000/07/xml-packaging-charter.html

[*Semantic Webapps*]
Semantic Webapps? Lightweight RDF interfaces for SVG. Sept 7, SVGOpen 2004, Japan. Available from: http://www.w3.org/2001/sw/Europe/talks/200409-svgopen/slide1-0.html