Opera Widgets

Widget information file syntax

Editor

Arve Bersvendsen, Opera Software ASA, arveb@opera.com

Abstract

This document describes the format of the widget information file, which is a file that provides data neccessary to present information to the widget application runner, so it can start up the widget and display it to the user.

Status of this document

This document is a working draft, and this specification may change in the future.

Terminology

The keywords *"must"*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this document are to be interpreted as described in RFC 2119. This applies regardless of the case or weight of the word in the specification.

Table of contents

Introduction

Widgets are small, locally installed applications created using web technology, and are typically displayed on a user's desktop using web browser technology. These widgets are delivered in a compressed single-file format that contain the code needed to run the widget and information on the widget.

This document describes the format of the widget information file, which is a file that provides data neccessary to present information to the widget application runner, so it can start up the widget and display it to the user. This file is an XML file which contents should be displayed later on.

Prerequisites

Widget configuration file **must** be named `config.xml`, and **must** be a valid XML 1.0 file according to the Extensible Markup Language (XML) 1.0 specification.

Namespace

Elements in Opera widget information files are members of the `http://xmlns.opera.com/2006/widget` namespace. The namespace declaration **should** be present in the widget configuration file, and a widget application runner **should not** launch widgets that do not have a namespace.

> Open issue: Please note that there is a known issue with Opera 9.0 and prior versions with widget support, where widgets will not launch if a namespace declaration is present. If compatibility with Opera 9.0 is a target, it is therefore accepted to omit the namespace declaration.

config.xml

The `config.xml` file contains all the information needed for a User Agent to be able to start the widget. In addition to this, the file contains information that *may* be presented to an end-user prior to or after installation of a web widget. An example widget information file may look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<widget xmlns="http://xmlns.opera.com/2006/widget">
  <widgetname>Hello World! widget</widgetname>
  <description>
    This is an example widget for widget developers
  </description>
  <icon>myicon.png</icon>
  <id>
    <host>example.com</host>
    <name>foo</name>
    <revised>2005-10</revised>
  </id>
  <width>468</width>
  <height>60</height>
  <author>
    <name>John Doe</name>
    <email>nobody@example.com</email>
    <link>http://www.example.com</link>
    <organization>Example, inc.</organization>
  </author>
  <security>
    <access>
      <host>example.com</host>
    </access>
    <content>
      <java>yes</java>
      <plugins>no</plugins>
    </content>
  </security>
</widget>
```

The `widget` element

The `widget` element is the root element of a *`config.xml`* document, and acts as the container for the various elements of the widget file.

Required elements of `widget`

`widgetname`

The `widgetname` element *must* be present in the document. The value of the element is a string used to provide a human-readable title for the widget, used in the widget managment interface.

`id`

The `id` element *must* be present in the document. The purpose of this element is to establish a unique ID for a widget. The `id` has three child elements:

`host`
: This element is *required*, and *must be* a fully qualified domain name naming the host from which the widget was downloaded.

`name`
: This element is *required*, and *must be* a string that is unique to the domain specified in the `host` element.

`revised`
: This element is *optional*, and if present, *must be* a string in the W3CDTF format, with the exception that both Year and Month is made mandatory.

`width`

The `width` element is a required element of the `widget` element and *may* be present in the

document. If the element is present, it's value *must* be a valid integer. The value of the element contents describe the width of the widget in pixels.

`height`
The `height` element is a required element of the `widget` element and *may* be present in the document. If the element is present, it's value *must* be a valid integer. The value of the element contents describe the height of the widget in pixels.

The purpose `width` and `height` of the width and height elements for the widget is to establish an initial window size for the widget. For performance optimization, widget authors are advised to not set an excessively large value for these elements. If resizing of a widget is desired, authors should use the javascript method `window.resizeTo` instead.

Optional elements of `widget`

`author`
The `author` element is an optional element of the `widget` element. If present, the purpose of the element is to provide information about the widget's author. If present, this element has child elements, as defined later in this specification.

`name`
: If the `author` element is present in the document, the `name` element *should* be present as a child element of the `author` element. If present, this element should contain a string with the name of the widget author. The author element can only contain one name element.

`organization`
: If the `author` element is present in the document, the `organization` element *may* be present as a child element of the `author` element. If present, this element should contain a human-readable name for an organization.

`email`
: If the `author` element is present in the document, the `email` element *may* be present as a child element of `author`. If present, this element should contain a string with a valid e-mail address, according to RFC 2822.

`link`
: If the `author` element is present in the document, the `link` element *may* be present as a child element of `author`. If present, this element should contain a string, whose value must be a valid IRI as specified by RFC 3987.

`description`
: The `description` element is an optional element of the `widget` element. If present, the element should contain a string that serves as a human-readable short description of the widget

`icon`
The `icon` element is an optional element of the `widget` element. The purpose of this element is to provide a pointer to an icon file contained within the widget archive. This image *should* be a png or gif file. This icon is typically displayed in the widget manager interface, or in the Operating

System's task bar.

security

The `security` element is an optional element of the `widget` element and *may* be present in the document. The purpose of the `security` element is to act as a container for security-related settings in a widget. The elements of the `security elements are described below`.

If the `security` element is missing, the widget is assumed to have universal access. As such, `security` is used to establish a trust relationship with the user. If the security element is present, the widget is not permitted to contact sites outside the permissions given by the rules in the `security` element.

access

The `access` element is an optional element of the `security` element. The child elements of `access` declares which protocols, hosts, ports, and paths the widget may use. Missing subelements are interpreted to mean "any". Furthermore, there can be multiple subelements with the same name, and the product of all these is used.

`protocol`

The protocols the widget will be using to contact external servers. All protocols, except the *`file`* protocol is permitted

`host`

The `host` element establishes which hostnames may be contacted. The hostnames is an exact match, so a widget specifying `www.example.com` *MUST NOT* be able to contact `example.com`. IP addresses *may* also be used as values.

`port`

The `port` element establishes which port numbers the widget will be using. The value is either a number, a range of numbers separated by a dash, e.g. `1024-2048`, or a comma-separated list of ports, e.g. `80, 31337`.

`path`

The `path` element specifies the path part of the URI a widget may contact.

content

The `content` element is an optional element of the `security` element. The purpose of this element is to allow the widget to declare if it needs to use plug-ins or Java. A widget that makes use of such technologies *must* use this element, and the appropriate child elements. The possible child elements are:

`java`

The `java` element can contain either of the (case-insensitive) strings `yes` or `no`. When the value of the element is `yes`, the widget requests access to run Java applets, eg.

`plugin`

The `plugin` element can contain either of the (case-insensitive) strings `yes` or `no`. When

the value of the element is `yes`, the widget requests access to run plugins.

Please note that while authors can ask for plug-in and Java access, the user may have turned off these features by configuration. Authors should therefore provide an appropriate fallback mechanisms if these technologies are unavailable, in accordance with guidelines 1 and 6 of Web Content Accessibility Guidelines 1.0.

Widgets and intranets

Definition

An intranet is defined based on the resolved IPv4 address of a host name. The following IPv4 ranges are defined as intranets:

- 10.0.0.0 to 10.255.255.255
- 172.16.0.0 to 172.31.255.255
- 192.168.0.0 to 192.168.255.255
- 169.254.0.0 to 169.254.255.255

URL Access

While widgets are allowed to contact intranet sites, and specify them in the access element, there are some exceptions:

- A widget that is attempting to communicate with intranet sites cannot later communicate with internet sites
- A widget that is attempting to communicate with internet sites cannot later communicate with intranet sites