Deliverable D3.2
FP7-ICT-2009-5 257103
June 2011

D3.2: webinos phase I device, network, and server-side API specifications

Project Number	:	FP7-ICT-2009-5 257103
Project Title	:	Secure WebOS Application Delivery Environment (webinos)
Deliverable Type	:	Public

Deliverable Number	:	D 3.2
Contractual Delivery Date	:	June, 30th, 2011
Actual Date of Delivery	:	June, 30th, 2011
Title of Deliverable	:	*webinos phase I device, network, and server-side API specifications*
Contributing Work Package	:	WP 3
Nature of Deliverable	:	Report
Editor	:	SonyEricsson
Authors	:	Fraunhofer FOKUS, Deutsche Telekom, ERCIM, Telecom Italia, TNO, BMW F+T, SonyEricsson., ISMB,

Document History

Version	**Date**	**Author (Partner)**	**Remarks**
0.9	01/07/2011	SonyEricsson	Initial version created from Wiki
1.0 α	04/07/2011	Fraunhofer FOKUS	Updated, Word-formatted version

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Abstract

This deliverable covers the APIs to be implemented by a webinos device. According to the use cases and requirements defined in WP2 and the common components introduced in task 3.1, task 3.2 has defined a set of application programming interface specifications (APIs) to make the desired functionalities available to webinos applications.

Due to earlier or ongoing standardization and implementation activities, e.g. within W3C and WAC, some needed APIs are already specified and available in modern browsers. Other APIs have been specified but implementations have not yet been established in existing browsers.Some APIs needed to fulfill webinos functionality do not yet exist so these APIs have to be specified within the webinos project.

A key feature of webinos is the ability to discover services on remote devices and access these services using APIs. For example, an application can use the core webinos Service Discovery API to search for a geolocation service on another device and then access this service through the standard W3C Geolocation API.

The webinos APIs can be divided into a number of categories:

• Webinos base and generic objects/interfaces: For example the webinos core interface
• APIs for service discovery and remote API access: APIs allowing applications to discover other devices and services/applications on other devices and on network servers and access these remote services.
• HW Resources APIs: APIs allowing applications to access information and functionality relating to device HW resources such as GPS, camera, microphone, sensors, etc.
• Application Data APIs: APIs allowing applications read and write access to application capabilites such as contact items, calender information, messages, media files, etc.
• Communication APIs: APIs allowing applications to communicate with other applications in the same or another device.
• Application execution APIs: APIs allowing webinos applications to launch other webinos and native applications.
• User profile and context APIs: APIs allowing applications access to user profile data and user context.
• Security and Privacy APIs: APIs related to the security model for webinos

Note:
This Word/PDF linear document represents only a snapshot of the specification for the purpose of review as a single document. The actual specification is located on the webinos redmine/Wiki. That version is the one relevant for the work within the project. Due to the close interworking between the specification and the implementation work packages in webinos, experience gained about gaps that need to be filled in the specification will be fed back directly into the online specification. The Word/PDF document has been exported from the online version and represents the status of the specification on June, 30th, 2011.

Keyword list:
webinos, specification, API, JavaScript, attestation, authentication, context, events, applauncher, messaging, nfc, payment, sensors, discovery, tv, suerprofile, vehicle, webinoscore, widget, calendar, contacts, device (status, interaction, orientation), file (reader, writer, directory), gallery, geolocation, media capture

Content

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

1. Introduction

According to the use cases and requirements defined in WP2 and the common components introduced in task 3.1, task 3.2 has defined a set of application programming interface specifications (APIs) to make the desired functionalities available to webinos applications.

Due to earlier or ongoing standardization and implementation activities, e.g. within W3C and WAC, some needed APIs are already specified and available in modern browsers. Other APIs have been specified but implementations have not yet been established in existing browsers and some APIs needed to fulfill webinos functionality do not yet exist so these APIs have to be specified within the webinos project.

A key feature of webinos is the ability to discover services on remote devices and access these services using APIs. For example, an application can use the core webinos Service Discovery API to search for a geolocation service on another device and then access this service through the standard W3C Geolocation API.

The webinos APIs can be divided into a number of categories:

- **Webinos base and generic objects/interfaces:** For example the webinos core interface
- **APIs for service discovery and remote API access:** APIs allowing applications to discover other devices and services/applications on other devices and on network servers and access these remote services.
- **HW Resources APIs:** APIs allowing applications to access information and functionality relating to device HW resources such as GPS, camera, microphone, sensors, etc.
- **Application Data APIs:** APIs allowing applications read and write access to application capabilites such as contact items, calender information, messages, media files, etc.
- **Communication APIs:** APIs allowing applications to communicate with other applications in the same or another device.
- **Application execution APIs:** APIs allowing webinos applications to launch other webinos and native applications.
- **User profile and context APIs:** APIs allowing applications access to user profile data and user context.
- **Security and Privacy APIs:** APIs related to the security model for webinos.

All webinos API specifications are available here: webinos Device APIs

Given the sensitive nature of the data to which these APIs grant access, the APIs specified are either secure and privacy-enabling by design or implemented so that access to APIs are controlled by the webinos security framework specified in WP 3.5.

The API development works in collaboration with WP8.1 to enable API standardizations on the one hand and to make use of existing specifications on the other hand.

2. Acknowledgement

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

For more information about this programme, see: http://cordis.europa.eu/fp7/ict/ssai/

3. Methodology

The methodology used in WP 3.2 was according to the following steps:

1. Identify needed APIs based on requirements:

- Use cases and requirements defined in WP 2 were analyzed in order to identify needed APIs.

2.Identify needed APIs based on webinos architecture:

- In parallel with the ongoing specification of the webinos architectural platform elements in WP 3.1 needed APIs were identified.

3. API investigations:

Based on step 1 and 2 existing APIs from W3C, WAC and elsewhere were investigated. The result of these investigations were:

- Description of referred APIs from W3C, WAC or elsewhere that can be used by webinos without modification.
- Description of referred APIs from W3C, WAC or elsewhere that can be used by webinos with modifications.
- Description of new APIs that need to be specified within the webinos project.

4. API specifications:

- Creation of new webinos API specifications.
- Creation of specifications of webinos modifications/extensions to existing APIs where needed.
- Creation of webinos "wrapper" specifications for referred API specifications.

5. Prototyping and demos

- Creation of API stubs and small demo applications to test API applicability for developers.

Further work will be performed within WP 8.1 to propose new webinos specifications and modified standard specifications to relevant standardization organizations.

4. API landscape

This section is an overview of existing API standardization and collaboration projects.

World Wide Web Consortium (W3C)

The World Wide Web Consortium is an international community where member organizations, a full-time staff, and the public work together to develop Web standards. W3C's mission is "to lead the Web to its full potential". W3C is the most important organization for standardizing Web technology and an extensive set of APIs for Web application developers have been specified by W3C.

W3C Web Applications (Web Apps) Working Group

The W3C Web Applications WG provides specifications that enable improved client-side application development on the Web, including specifications both for application programming interfaces (APIs) for client-side development and for markup vocabularies for describing and controlling client-side application behavior.

This WG hosts a number of API specifications that are core for the Web as an application execution environment. APIs that are implemented in all browers are for example XMLHTTPRequest and Document Object Model (DOM). Other important APIs created by the Web Apps WG that are deployed in modern browers are for example Web Workers, Web Messaging, File Reading, Server-Sent Events and Web Sockets. The Web Apps WG has also created a set of specifications for installable Web applications, Web Widgets, which are core specifications in Web runtime platforms such as WAC.

See Web Apps WG charter and Web Apps WG Web Site. A full list of the WG's publications and their status can be found at Web Apps WG publications.

W3C Device APIs and Policy (DAP) Working Group

The mission of the Device APIs and Policy Working Group is to create client-side APIs that enable the development of Web Applications that interact with device hardware, services and applications such as the camera, microphone, system sensors, native address books, calendars and native messaging applications. Devices in this context include desktop computers, laptop computers, mobile Internet devices (MIDs), cellular phones, TVs, cameras and other connected devices.

A full list of API specifications created by the WG is here offers the DAP Roadmap.

The DAP specifications that so far has got most implementation attention are the Contacts API and HTML Media Capture API. The latter is supported in Android 3.0, for instance and the Contacts API has experimental implementations.

Previously a framework for the expression of security policies that govern access to security-critical APIs was included in the deliverables of the WG but according to the new proposed DAP charter this is left out of the DAP WG deliverables. It is proposed to rename the WG to "Device APIs Working Group". However, this does not mean that the WG no longer addresses privacy and security as the group also aims at crafting APIs that are both secure and privacy-enabling by design, based on the current Web browser security model. This entails reusing existing browser-based security metaphors where they apply and looking into innovative security and privacy mechanisms where they don't.

Furthermore, the new charter expands the set of APIs that should be delivered by the WG. For example APIs for device and service discovery is now inluded in the charter.

DAP's public web site is here: W3C DAP.

W3C Geolocation Working Group

The mission of the Geolocation Working Group is to define a secure and privacy-sensitive Geolocation API for accessing location information from built-in GPS receiver or network positioning information as well as a Device Orientation Event specification for using device orientation information originating from built-in accelerometer, magnetometer and gyro.

The Geolocation API is currently implemented in major modern Web browsers and according to public information the Device Orientation Event specification is at least under implementation for iOs, Android and Chrome.

See the Geolocation WG Charter.

W3C Web Real-Time Communications (Web-RTC) Working Group

The mission of the recently formed Web Real-Time Communications Working Group is to define client-side APIs to enable real-time communications in Web browsers. APIs specified by the WG will enable streaming access to device capabilities, e.g camera and microphone, and API functions for establish peer-to-peer connections between Web browsers, independent of the network protocols used to establish the connections between peers.

See Web-RTC Working Group Charter and Web-RTC Web site

Other W3C actvities creating APIs for Web applications

- The HTML5 specification contains a number of APIs for Web applications, e.g. an API for playing of video and audio to be used with the video and audio elements, an API that enabling offline Web applications and a drag & drop API.
- The Web Notifications API is an API for displaying simple notifications to the user.

- The Web Events Working Group develops specifications for physical multitouch interface events.
- The HTML Speech Incubator Group investigates the feasibility of integrating speech technology in HTML5.

Web Hypertext Application Technology Working Group (WHATWG)

WHATWG is a community of people interested in evolving the Web. It focuses primarily on the development of HTML and APIs needed for Web applications. The WG was founded by individuals of Apple, Mozilla Foundation, and Opera Software in 2004, due to concerns on the W3C's direction with HTML and XHTML.

WHATWG has provided major input to the W3C HTML5 specification as well as to other specifications relating to Web applications, e.g. Web Workers , Web Storage, the Web Sockets API, and Server-Sent Events.

Wholesale Application Community (WAC)

The Wholesale Applications Community is an open, global alliance formed from the world's leading telecoms operators. WAC will unite a fragmented applications marketplace and create an open industry platform that benefits the entire ecosystem, including applications developers, handset manufacturers, OS owners, network operators and end users.

WAC is based on W3C Web technology such as HTML5, JavaScript, DOM and Web widgets. In addition WAC has specified a set of APIs providing access to hardware and software device capabilities as well as a security policy framework to control the access to the sensitive device APIs.

Full list of WAC specifications can be found here:

- WAC 1.0
- WAC 2.0

PhoneGap

PhoneGap allows developers to build applications with Web technology that are wrapped into native applications suited for the target platform giving access to APIs provided by the native platform. The following table lists the APIs available for the platforms supported by PhoneGap: PhoneGap supported feature. For API documentation see PhoneGap API reference.

Nokia/Symbian Web Runtime environment

The Nokia/Symbian Web Runtime provides an application environment for Web widgets that includes a set of device APIs (Symbian Platform Services 2.0).

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

5. API types

JavaScript APIs

A JavaScript API is the most common way to provide Web application access to device hardware and software resources. Web Interface Description Language is used to specify JavaScript APIs.

A typical example of a JavaScript API is the W3C Contacts API. The usage of the "contacts.find" method is examplified below:

Perform an address book search. Obtain the 'name' and 'emails' properties and initially filter the list to Contact records containing 'Bob':

```
navigator.contacts.find(  ['name', 'emails'], successCallback, errorCallback,
{filter: 'Bob'} );
```

The example above illustrates an asynchronous JavaScript method, which is very common for JS device APIs. Asynchronous methods return immediately and notify the caller at some point in the future of the results via callback methods. Methods that may take a long time to be executed or that may be subject to security prompt must be defined as asynchronous methods. The successCallback above is a function to be invoked in case of success and the errorCallback is a function to call when the asynchronous operation fails.

Using HTML-elements

In some cases access to device resources can be provided through a simple html-element. One example is the W3C HTML Media Capture specification. This specification states that if an input element in the File Upload state contains accept attribute with values image/*, audio/*, or video/*, the user agent can invoke a file picker that allows respectively the user to take a picture, record a sound file, or record a video in addition to selecting an existing file from the file system. Furthermore, a new “capture” attribute may be added to the input element. This attribute gives a hint to the user agent on the source of the input. The “capture” attribute can take the values camera, camcorder, microphone and filesystem.

For example, the following code indicates that the user is expected to upload an image from the device camera:

```
< input type="file" accept="image/* " capture="camera"  id="capture" >
```

When rendering this code the user agent will open the camera viewfinder and allow the user to take a picture.

HTML "file-picker" based access to device resources is very straightforward and intuitive for users and provide for "implicit user consent" as the user must provide a tangible action to allow access to the requested resource. For example navigating to a folder in the file system and selecting a file or using the camera viewfinder and pressing the shutter button to take a picture.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Using DOM events

For providing Web applications access to data that is frequently updated, for example data from sensors in the device, a DOM event based interface is often applicable. Examples are the W3C DeviceOrientation Event Specification and the W3C Battery Status Event Specification. Several new webinos APIs are also DOM event based.

An event based API is defined by adding attributes to the DOM event interface. For example, the DeviceOrientation Event Specification defines an "deviceorienationevent", which has three attributes for device orientation, alpha angle, beta angle and gamma angle.

A Web application can register to listen to DOM events using the addEventListener method. For example, registering to receive deviceorientation events could be done with the code below:

```
window.addEventListener("deviceorientation", function(event) {
          // code for processing the the device orientation event data, i.e.
event.alpha, event.beta and
          // event.gamma
      }, true);
```

The second parameter is a method that is called whenever an event occurs of type "deviceorientation".

Using REST

REST (Representational State Transfer) APIs are frequently used on the Web. Such an API is specified as a URI and the requested resource/service is accessed through the standard HTTP methods GET, POST, PUT, DELETE.

A simple example is a Twitter API for retrieving the 20 most recent statuses. In this example the requested data is returned in JSON format.
http://api.twitter.com/1/statuses/public_timeline.json

One major advantage with REST APIs is that a requested resource/service could be situtated "anywhere", in the cloud or in the device, but still be accessed with the same API. By implementing access to local resources/services through “Virtual Local Web Servers” REST APIs could also be used to access local in-device resources/services.

Furthermore REST is stateless, which facilitates scalable solutions so that many users can be supported.

If coding gets complicated when REST APIs are used then JavaScript "wrapper" methods can be created to facilitate for developers. These "wrapper" methods do not have to be standardized and could be provided by exstablished JavaScript library/framework providers.

6. API investigations

This section contains the results of the investigations on APIs of the different categories and acts as background information to the APIs supported by webinos.

Based on webinos requirements/use cases and architecture needed APIs are identified. Potential existing APIs from W3C, WAC and elsewhere are investigated and analyzed. If no existing APIs that could fulfill the webinos requirement is found high level requirements on a new API to be specifed within the webinos project are stated.

HW Resource APIs

Description

This section contains investigation results on APIs for access to HW related resources.

Resources

Primary contributor/editor for this API category: Telecom Italia
Supporting contributors/reviewers: SEMC / AmbiSense / Fraunhofer / BMW

APIs based on existing standards/implementations

Device Orientation API

Description: Information about the physical orientation of a device, typically implemented by using information from accelerometer, magnetometer and gyro.
Requirement/architectural reference: CAP-DEV-SEMC-009:webinos **SHALL** provide means for applications to access device physical orientation
Phase: webinos phase 1
Webinos responsible: Claes Nilsson/SEMC

Candidate API	**Short Description**	**Implementation Status**	**Gaps**	**Notes**	**Decision**
W3C DeviceOrientation Event	Two DOM event types that provide information about the physical orientation of a hosting device. - The first event is a simple, high-level source of information about the physical orientation of a device, expressed as device rotation in angles around 3 different axes. While the spec is agnostic to the source of information, this is	iOS 4.2 Android 3.0 Chrome 7 Opera Mobile for Android (experimental)	No gaps identified		webinos will use this API

	typically implemented by combining information from an accelerometer and a magnetometer. - The second event provides direct access to motion data from an accelerometer and gyroscope and is intended for more sophisticated applications. Acceleration is expressed in m/s2 and rotation rate is expressed as degrees/s.				
WAC 2.0 Device APIs: The orientation module	Device orientation information, expressed as device rotation in angles around 3 different axes.	Existing 3rd party implementations of WAC WRT clients from Obigo, Opera, Aplix, etc, for Android and other platforms	No gaps identified		
WAC 2.0 Device APIs: The accelerometer module	Provides access to the device accelerometer information expressed in m/s2 in 3 different axis.	Existing 3rd party implementations of WAC WRT clients from Obigo, Opera, Aplix, etc, for Android and other platforms	The API does not provide a means to separate acceleration due to movement from acceleration due to gravity, which could be provided by devices containing both an accelerometer and a gyroscope.		

Generic SensorActuator API

Description: It currently exist a set of APIs tailored for specific sensor data. Examples are the W3C Geolocation API (GPS), the W3C DeviceOrientation Event (accelerometer etc) and the W3C HTML Media Capture API (camera, microphone). However, there is also a need for a generic/extensible API to get access to sensors. This is needed as new types of sensors are frequently introduced. These sensors could be:

- Built in the user's current device, for example a built thermometer or barometer
- Connected with the user's current device through a local connectivity method such as USB, Bluetooth or ANT+, for example a Bluetooth enabled medical sensor.
- Located anywhere in "the cloud”.

The API should be agnostic to the location of the sensors and to underlying discovery and connection methods.

Requirement/architectural references:

- CAP-DEV-SEMC-015: webinos **MUST** support a generic/extensible API for allowing applications access to locally connected non-webinos enabled sensors/actuators.
- CAP-DEV-SEMC-016: webinos **MUST** support a generic/extensible API for allowing applications access to webinos enabled sensors/actuators connected to the webinos cloud.

Phase: webinos phase 1

Webinos responsible: Claes Nilsson / SEMC

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
W3C The System Information API	A high-level API to system information and sensors. - A set of simple sensor APIs is included in the specification - Agnostic to underlying sensor access method - All APIs are asynchronous. - Simple get value or watch for continuous "callbacks" when the values change or when the values reach below or above certain defined threshold values. - The sensor properties currently included in the specification are:	No known implentations	Gaps: - For each additional sensor a new sensor property has to be defined - Reading only, writing data or control sensor not supported - Only one	This API has been criticized within W3C and the future for this API is uncertain. See: Sys Info feedback. There is a proposal to create a set a smaller discrete APIs to specific system properties such as network and battery. For sensors a set of discrete event based APIs similar	

	AmbientLight AmbientNoise AmbientTemperature AmbientAtmosphericPressure Proximity		single value for each sensor property, compound data patterns not supported	to the DeviceOrientation Event has been proposed. In addition it is proposed to break out the current sensor part from The System Information API to create a separate "Generic sensor API".	
The Bondi sensor Module - Version 1.5	*Sensor API Sensors are classified by type. Sensor type names are defined Strings, and creation new type names must be centrally defined (by the owner of this API definition)	Unknown			
WAC 2.0 devicestatus module	Access to various information regarding the status of the device. - All APIs are asynchronous. - Simple get value or watch for continuous "callbacks" when the values change or when the values changes a certain percent. - Compound data patterns supported	Existing 3rd party implementations of WAC WRT clients from Obigo, Opera, Aplix, etc, for Android and other platforms	Gaps: - Extensions to the WAC vocabulary is needed to cover not only internal device status but also internal and external sensors. - Reading only, writing data or control		

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

			sensor not supported		
Symbian WRT Platform Service 2.0 Sensors API	Access to sensor data: - Asynchronous event based - Compound data patterns supported	Nokia/Symbian WRT	Gaps: - Extensions, i.e. additional sensor channels need to be specified for all sensors supported - Seems as not possible to trigger on threshold values - Reading only, writing data or control sensor not supported		

Decision:

A new generic sensor API will be specified. This API is inspired by W3C DeviceOrientation Event Specification, W3C Battery Status Event Specification and the Android sensor API. For phase 1 only reading, not writing, sensor data will be supported.

Editor: Claes Nilsson / SEMC

High level Requirement	**Notes**
Find sensors in device, locally connected to the device or in the cloud	
Configure a selected sensor	
Provide sensor data as a DOM event	

Microphone API

Description: Capture audio samples from microphone

Requirement/architectural reference: CAP-DEV-SEMC-004

Phase: webinos phase 1

Webinos responsible: Stefano Vercelli / Telecom Italia

Candidate API	**Short Description**	**Implementation Status**	**Gaps**	**Notes**	**Decision**
W3C Media Capture Api (see http://www.w3.org/TR/media-capture-api/)	Api for capturing audio/video/image data	W3C working draft. Implemented by Phonegap project. See PhoneGap Capture	Only useful for capturing media files, doesn't give access to live stream		webinos will use this API due to security and remote access reasons
W3C HTML Media Capture Api (see http://www.w3.org/TR/html-media-capture/)	Defines a new interface for media files, a new parameter for the accept attribute of the HTML input element in file upload state, and recommendations for providing optimized access to the microphone and camera of a hosting device	W3C working draft, under implementation for Android 3.0 and a Bug tracking implementation in WebKit	Not an API in itself, more an add-on to file upload. Only useful for capturing media files, doesn't give access to live stream		
WhatWG Device Element (see http://www.whatwg.org/specs/web-apps/current-work/complete/commands.html#devices)	Provides a Stream API to be used on top of user-selected sources	WhatWG draft, experimental impl. in WebKit			

and Stream API	of input. Note: Probably replaced by getUserMedia	(e.g. Ericsson's)			
WhatWG getUserMedia (see http://www.whatwg.org/specs/web-apps/current-work/complete/video-conferencing-and-peer-to-peer-communication.html)	Early draft in WHAT WG. Provides a Stream API to be used on top of user-selected sources of input.	Experimental implementation for Opera Mobile		Work in W3C started in Web RTC WG	Proposed to support in phase 2

Camera API

Description: Capture video stream from device camera
Requirement/architectural reference: CAP-DEV-SEMC-005
Phase: webinos phase 1
Webinos responsible: Stefano Vercelli / Telecom Italia

Candidate API	**Short Description**	**Implementation Status**	**Gaps**	**Notes**	**Decision**
WAC 2.0 camera module (see http://specs.wacapps.net/wac2_0/feb2011/deviceapis/camera.html)	Interface to device camera for capturing video or image	Implementations of WAC WRTs by Obigo, Opera, Aplix, Borqs			
W3C Media Capture Api (see http://www.w3.org/TR/media-capture-api/)	Api for capturing audio/video/image data	W3C working draft. Implemented by Phonegap project. See PhoneGap Capture	Only useful for capturing media files, doesn't give access to live stream		webinos will use this API due to security and remote access reasons
W3C HTML Media Capture Api (see http://www.w3.org/TR/html-media-capture/)	Defines a new interface for media files, a new	W3C working draft, under implementation	Not an API in itself,		

	parameter for the accept attribute of the HTML input element in file upload state, and recommendations for providing optimized access to the microphone and camera of a hosting device	for Android 3.0 and and a Bug tracking implementation in WebKit	more an add-on to file upload. Only useful for capturing media files, doesn't give access to live stream		
WhatWG Device Element (see http://www.whatwg.org/specs/web-apps/current-work/complete/commands.html#devices) and Stream API	Provides a Stream API to be used on top of user-selected sources of input. Note: Probably replaced by getUserMedia	WhatWG draft, experimental impl. in WebKit (e.g. Ericsson's)			
WhatWG getUserMedia (see http://www.whatwg.org/specs/web-apps/current-work/complete/video-conferencing-and-peer-to-peer-communication.html)	Early draft in WHAT WG. Provides a Stream API to be used on top of user-selected sources of input.	Experimental implementation for Opera Mobile		Work in W3C started in Web RTC WG	Proposed to support in phase 2

Here is an analisys of Media Capture and HTML Media Capture. Notice that both apis require the File api (http://www.w3.org/TR/FileAPI/).

W3C HTML Media Capture API

It looks like it has been designed for uploading pictures/audio/videos (it uses the HTML input tag). A "file picker" is launched and it can select an existing file or take a new picture/audio/wideo. No options are available before launching the app. It looks like there's no way to know at JS level when the picture/audio/video has been taken. It is not clear if the picture/audio/video is saved on the filesystem.

Supposing we want an app that takes a picture and displays it, here's a theorical code snippet with HTML Media Capture:

```
...
<script type="text/javascript">
function displayImage() {
var captureInput = document.getElementById('capture');
var file = captureInput.files[0];
document.getElementById("myImage").src = file.url;
}
</script>
...
<body>
...
<input type="file" accept="image/*;capture=camera" id="capture">
<img id="myImage" src="defaultImage.jpg"/>
...
</body>
```

Notice that the displayImage() function should probably be invoked explicitly by the user.

W3C Media Capture APIi

It uses an external app to take the picture/audio/video. A few options are available before launching the app ("limit", that is the number of pictures/videos/audios to take; the duration of the video), and a few have been proposed (height and width of image, format of the output, duration of audio). Error and success callbacks are available. It is not clear if the picture/audio/video is saved on the filesystem (it depends on the external app).

Supposing we want an app that takes a picture and displays it, here's a theorical code snippet with Media Capture:

```
...
<script type="text/javascript">
function takePicture() {
navigator.device.capture.captureImage(successCB, errorCB, { limit: 1 }); //it
takes 1 picture and exits
}
fucntion successCB(data) {
document.getElementById("myImage").src = data[0].url;
}
function errorCB(err) {
alert("an error occurred");
}
</script>
...
<body>
...
<button onClick="takePicture()">Take picture</button>
<img id="myImage" src="defaultImage.jpg"/>
...
</body>
```

Geolocation API

Description: Access to device location information

Requirement/architectural reference: CAP-DEV-SEMC-008

Phase: webinos phase 1

Webinos responsible: Stefano Vercelli / Telecom Italia

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
W3C Geolocation API	Access to device location regardless of the source of information (it may be GPS, GSM/CDMA cell id, wifi, ...)	Implemented in modern browers such as Chrome, Firefox, iOS, Android, Opera Mobile, etc	No gaps identified	A 2nd version of the API will also provide civic address information	webinos will use this API
GSMA OneAPI Location RESTful API	A RESTful API for querying the location of one or more mobile devices.	Open Source Reference Implementation in PHP/Java Commercial Pilot in Canada	Possibly no GPS support.	This method asks the network for location based on the MSISDN of the device	

Devicestatus API

Description: Access to device status informations

Requirement/architectural reference: CAP-DEV-SEMC-012, CAP-DEV-SEMC-013

Phase: webinos phase 1

Webinos responsible: Stefano Vercelli / Telecom Italia

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
WAC 2.0 devicestatus module	Access to various informations regarding the status of the device	Implementations of WAC WRTs by Obigo, Opera, Aplix, Borqs	An extension of the WAC vocabulary is needed to cover all info needed (CPU load, system temperature, audio/video codecs capabilities, input devices,	WAC Device Status Vocabulary	This API will be used, with an extended vocabulary

			...)		
W3C System Info API	"Access to various properties of the system which they are running on"	Not implemented		This API has been critiized within W3C and the future for this API is uncertain. See: Sys Info feedback. There is also a proposal to rework the sensor APi to a set of event based APIs according to the DeviceOrientation Event. For example see see http://lists.w3.org/Archives/Public/public-device-apis/2011Mar/0122.html and http://lists.w3.org/Archives/Public/public-device-apis/2011Mar/0123.html	
GSMA OneAPI 2.0 Device Capability	A RESTful API to query capabilities of a device.	unknown	Provides access to static information only like hardware and software platform properties. No access to e.g. battery status.		

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

TV and STB control API

Description: Control TV/STB via API so other devices can act as a remote control.
Requirement/architectural reference:

- WOS-US-3.3: Social Event Sharing
- WOS-US-10.1: User Centric Video Playback
- WOS-UC-TA1-001: Virtual Device
- WOS-UC-TA4-019: Ad hoc use of Foreign Devices for Playback of Film
- WOS-UC-TA7-005: Seamless Session Transfer between Devices

Phase: webinos phase 1
Webinos responsible: Fraunhofer

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
Open IPTV Forum Rel 2 Vol 5 Declarative Application Environment (See pdf)	Based on CE-HTML this includes a JavaScript API for apps on a TV/STB. Supports e.g. app installation and management, channel configuration, video playback, recordings, etc.	Unknown. TV sets supporting CE-HTML exist.	Addresses much more than what is needed to access/control features related to the broadcast.	The HbbTV standard is also based on this.	
BBC Universal Control API	A RESTful API which returns XML responses to GET requests to control TV and STB.	BBC prototypes (See blog announcement).	There seems to be no way to get access to the broadcast stream; e.g. to embedded it into an app.	Discussion on the mailing list wether all device features should be exposed via API on the server (TV/STB, TV watching as "app") and client (e.g. smartphone as remote control) or should they be just accessible to clients. For the former the proposed data model may be too strict and	

				limiting. See announcement and following discussion in replies	
The Dreambox Webinterface API	A RESTful API that is used on the DBox2 STB (for DVB-S, -C). The API allows to control volume, audio tracks, channel, EPG, messaging, etc. Responses are in XML.	DBox2 Linux Distro	No support for accessing the broadcast stream to embedded in own app, instead a control functionality only. Used only by dreambox hardware.	A community based project.	

Decision: A new TV control API will be specified. This API makes available access to TV channel streams that can then be plugged into a HTML5 HTMLVideoElement. Alternatively, it also provides means to control the channel playback of a native hardware component.

Deviceinteraction API

Description: Access to apis for interacting with the end user

Requirement/architectural reference:

Phase: webinos phase 1

Webinos responsible: Stefano Vercelli / Telecom Italia

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
WAC waikiki deviceinteraction module	Interaction with the user through features like device vibrator and screen backlight	Implementations of WAC WRTs by Obigo, Opera, Aplix, Borqs			This API will be used as W3C device interaction API is not yet in place.
Chrome extension for UI interaction	Chrome offers several apis for customizing the browser UI (add menus, tabs, desktop notifications)				
Bondi User Interaction module	Allows customization of menus related to specific phone keys as well as control of beeping, vibration, backlight, screen orientation				

Barcode API

Description: APIs for decoding barcodes using the camera of the device.

Requirement/architectural reference: CAP-DWP-ambiesense-51

Phase: webinos phase 1

Webinos responsible/editor: Stefano Vercelli / Telecom Italia.

Contributor: Hans Myrhaug, AmbieSense Ltd

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
ZXing	ZXing (pronounced "zebra crossing") is an open-source (Apache 2.0 licensed), multi-format 1D and 2D barcode image	In reality ZXing is becoming the open source industry standard for barcode recognition in mobile applications and	There are concerns that a JavaScript port might	This (possibly 3rd party provided) library would not be part of a device API but apps could choose to include	Bar code reading will be supported through a JavaScript

	processing library implemented in Java. The focus is on using the built-in camera on mobile phones to photograph and decode barcodes on the device, without communicating with a server. A JavaScript library based on ZXing is proposed.	there are fully working implementations on Android and other mobile platforms. See here for a way to use the library on a mobile device on a webpage.	be slow.	this library or not based on whether they need the functionality. See Measurements that states it takes 40-60 ms to process the image on a Gingerbread Android device with a 1GHz processor. That indicates that porting of Zxing to JS is probably feasible.	port of the ZXing Java library. This means that bar code reading is out of scope for further work within WP 3.2

APIs for which no existing standards/implementations exist

Vehicle API

Description: Provides access to vehicle proberties (e.g. current speed, mileage, fuel consumption)
Requirement/architectural reference: Extension Handling
Phase: phase 1
Webinos responsible/editor: Simon Isenberg, BMW

High level Requirement	**Notes**
Access to the Automotive API **MUST** be authorized based on applications	
In case of a denied access to the vehicle API the requesting application **SHALL** be informed.	
The following car properties **SHALL** be available read-only for applications - model - speed - current fuel consumption - average fuel consumption - trip kilometers/miles - total kilometers/miles - current units - gear - engine status - position of the steering wheel	
An application **MUST** be able to bind to car properties and be infomend about the new value	

An API for communicating with the in-car navigation system **SHOULD** be available	
It **SHALL** be possible to set time intervals for applications to access a car property	

A possible solution is to see this api as an extension of devicestatus for vehicles.

Candidate API	**Short Description**	**Implementation Status**	**Gaps**	**Notes**	**Decision**
WAC 2.0 devicestatus module	Access to various informations regarding the status of the device	Implementations of WAC WRTs by Obigo, Opera, Aplix, Borqs	An extension of the WAC vocabulary is needed (for example you can add a new Aspect ("vehicleInfo") with properties "model", "speed", ...)		
W3C System Info API	"Access to various properties of the system which they are running on"	W3C working draft		This API has been critiized within W3C and the future for this API is uncertain. See: Sys Info feedback. There is also a proposal to rework the sensor APi to a set of event based APIs according to the DeviceOrientation Event. Comment by Claes/SEMC: Seems as W3C is taking an event model route now for sensors. Consider to make a DOM level 3 event model based Vehice API similar to DeviceOrientation Event specification and Battery Status Event specification	

Decision: A separate Vehicle API will be specified. The API will provide read-access to car data in the first phase of the project. This API is inspired by W3C DeviceOrientation Event Specification, W3C Battery Status Event Specification.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Background information on the Vehicle API

In the browser/Web domain no developer API has been specified to access vehicle data so far. In other domains there are a few APIs publicly available, which provide access to vehicle data, but haven't been largely used yet.

In JSR 298 the OSGI vehicle expert group (VEG) defined a Telematics API for JAVA. This API provides access to the vehicle data for JAVA ME developers. The OSGI VEG was discontinued in 2006. The API has not been refined afterwards. The specification is available at: http://jcp.org/en/jsr/summary?id=Telematics.

In other EU founded projects the Serial Line Automotive Protocol (SLAP) introduced by Volkwagen has been used to retrieve vehicle data. The SLAP is based on XML-formatted messages to request and receive vehicle data.

Due to the lack of a feasible API for vehicle data inside the browser domain, we define a new API which provides read-only access to the following data:

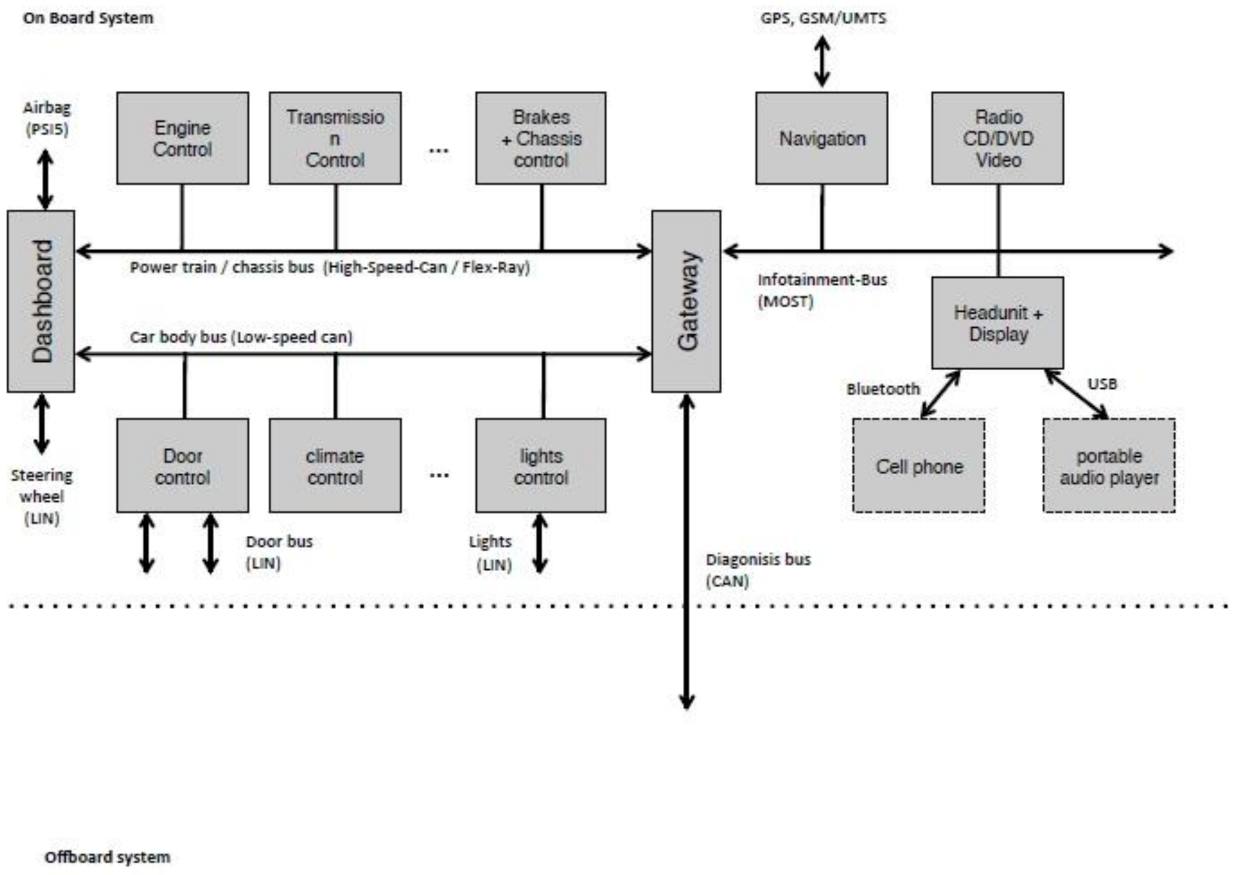
- static/general information (brand, model, year, transmission, fuel)
- trip computer (average consumption1, average consumption2, average speed 1, average speed 2, trip distance, mileage, range)
- climate control (zone, desired temperature, vent status (automatic or level))
- controls (lights (including signals, hibeam, fog), whiper)
- engine (gear, speed, acceleration)
- park sensors

Furthermore the API provides access to the following functions:

- setting the destination of the in-car navigation system
- canceling the guidance of the in-car navigation system
- querying the in-car navigation system for POIs

The API is aligned to the current W3C's approach of event based APIs. The vehicle API does not provide information about the geolocation, speed and acceleration. These attributes are already accessible using the W3C Geolocation API for speed and position and the W3C Device Orientation API for acceleration.

In the first iteration of the project the vehicle API focuses on the access of read-only data, which is available on the infotainment bus. The in-car headunit is usually connected this bus system as shown in the following depiction.



For the second iteration of webinos the extension of the vehicle API to data outsite of the infotainment bus (MOST) as well as more methods for interacting with the vehicle system seems beneficial.

NFC API

Near Field Communication (NFC) is an international standard (ISO/IEC 18092) that specifies an interface and protocol for simple wireless interconnection of closely coupled devices operating at 13.56 MHz. The overall application scenario is to hold a device close to a wireless tag to exchange some digital information or data. Alternatively, the scenario is to hold two devices close to each other in order to exchange some information or data between them. NFC is also sometimes referred to as contactless communication.

There are three use case categories for NFC driven by NFC Forum, www.nfc-forum.org:

1. NFC peer to peer communication, with use cases for sharing data between devices, and for pairing with other devices.
2. Tag R/W mode, with use cases for any application provider proposition integrate real world objects with Internet and applications.
3. Card Emulation mode, is to move the existing smart cards that you have in your wallet today into the phone and make use of contactless NFC connections.

All three propositions addressed by use cases of the NFC forum are indeed relevant to the webinos user stories and use cases. Ideally, webinos application developers will therefore need all of these implemented. Thus, in terms of webinos implementaiton priority, we recommend the following order: first 2, then 1, and then 3, because our target group is application developers, and most of them will most likely be doing 2 in the beginning. 1 will become increasingly common when there are a lot of smartphones with NFC capabilities around in the market. Currently, in June 2011, the following NFC enabled devices are being sold in the mass market: Samsung Nexus S and Nokia Oro. The Samsung Galaxy S2 and the Nokia C7 will also be shipped with NFC capability.

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
Android NFC		Read write mode is complete. Peer to peer mode is being implemented			This is our choice because Android is now shipped more than iPhone, the activity is high Android NFC, and the license is Apache 2.0
Open NFC		There are several versions for various platforms becoming available. The information about what is being implemented is unclear.			This is not a choice because the Android implementation is based on the Android NFC API, and it is unclear how much activity this open source project.
Libnfc		The implementation is in C and can be cross compiled for different operating systems.			This is not our choice, because of the LGPL license.
J2ME (JSR-257) NFC		The implementation is complete several years ago. Due to the current shift in the market, it is less likely that there will be any peer to peer mode supported.			This is not our choice, because of the shift from J2ME enabled devices towards Android in the current market.
QT Mobility NFC		The implementation is complete.			Due to the Nokia announcement on the Microsoft alliance, this seems more risky in the long term.
Symbian NFC		The implementation is complete.			Due to the Nokia announcement on the Microsoft alliance, this seems

					more risky in the long term.

Decision: An NFC API will be specified within webinos.

Requirement/architectural reference: TBD
Phase: Phase 1
Webinos responsible/editor: Hans Myrhaug / AmbiSense, Stefano Vercelli / TIM

High level Requirement	**Notes**
Tag R/W mode	Based on Android NFC API Gingerbread
NFC peer to peer communication mode	Based on Android upcoming NFC API

Below is the proposed roadmap for the webinos API implementation. It seems clear that NFC peer to peer it will be supported on Android and
that it already is supported on both QT and Symbian. Thus, we believe that NFC peer to peer should also be supported by webinos after the NFC read and write capabilities (v=implemented, x=not yet implemented).

Proposed roadmap for implementation of the webinos NFC API	**Phase I**	**Phase II**
Register to launch application	v	v
Launch application on a specific NFC tag type	v	v
Launch application on connection to a specified service name (LLCP)	x	v
Listening to NFC discovery events	v	v
An NDEF tag has been discovered	v	v
A specified NDEF record type has been discovered	v	v
An NFC target has been detected	v	v
Reading and writing to NFC tags	v	v
Read NDEF messages and records from an NFC tag	v	v
Write NDEF messages and records to an NFC tag	v	v
Peer to peer communcation between NFC devices	x	v
Open connection to an NFC device (LLCP)	x	v

Send data to an NFC device (LLCP)	x	v
Receive data from an NFC device (LLCP)	x	v

Application Data APIs

Description

This section contains investigation results on APIs for access to application data.

Resources

Primary contributor/editor for this API category: Fraunhofer

Supporting contributors/reviewers: SEMC / W3C

APIs based on existing standards/implementations

See W3C current state of mobile Web app technologies: data storage as a good starting point.

Contacts API

Description: Access/use the native contact application and its data. e.g. the contact application of a mobile device with Android or a fixed PC where Outlook is installed.

Requirement/architectural reference:

- WOS-US-1.1: Smart Device Integration
- WOS-US-2.3: Converging Applications within and across Devices
- WOS-UC-TA8-004: Install-time presentation and negotiation of application policies

Phase: 1

Webinos responsible: Fraunhofer

Candidate API	**Short Description**	**Implementation Status**	**Gaps**	**Notes**	**Decision**
WAC 2.0 Device APIs: The contact module	A JavaScript API to access multiple address books to add, update, delete or search for contacts.	Fraunhofer MWR prototype (partial)			
W3C DAP Contacts API	A JavaScript API for finding contacts. Adding or updating contacts should be done via existing Web	W3C ED Mozilla Labs experimental Firefox add-on (See latest release		The experimental Firefox add-on lags behind the DAP API draft (documented here). It also adds a	webinos will adapt this API from W3C - additions necessary due

	platform APIs (e.g. attach a vcard string or file to an html anchor element).	announcement) Bug tracking implementation in Webkit		service API for limited interaction with social Web services.	to webinos specific issues are listed below this table
PhoneGap Contacts API	A JavaScript API for creating, updating and search for contacts.	Android BlackBerry WebWorks (OS 5.0 and higher) iOS			
Nokia Platform Services 2.0 Contacts API	A JavaScript API for accessing and managing contact information in the default contact database.	Symbian WRT 1.1			

webinos specific additions

The W3C Contacts API specification defines the concept of a user's unified address book - where address book data may be sourced from a plurality of sources - both online and locally. However, the selection of sources for this unified address book is out of scope for the W3C Contacts specification. For the multi-device useability of webinos, a function needs to be added that allows the retrieval of a list of contacts across devices using search/discovery criteria, most likely be based on the webinos ServiceDiscovery module.

Calendar API

Description: Access/use to native calendar application and its data, e.g. the calendar application of a mobile device with Android or a fixed PC where Outlook or Thunderbird are installed.

Requirement/architectural reference:

- WOS-US-1.1: Smart Device Integration
- WOS-US-2.3: Converging Applications within and across Devices
- WOS-US-3.3: Social Event Sharing
- WOS-US-8.2: Seamless Navigation
- WOS-UC-TA1-013: Generating Reports
- WOS-UC-TA1-014: Solving Problem with Clashing Appointments
- WOS-UC-TA7-006: The publicity and privacy of Context Information
- WOS-UC-TA8-004: Install-time Presentation and Negotiation of Application Policies
- WOS-UC-TA8-009: User switching between personal policies

Phase: 1
Webinos responsible: Fraunhofer

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
WAC 2.0 Device APIs: The calendar module	A JavaScript API to access multiple calendars defined as set of events that can be created, updated, deleted or searched for.	Fraunhofer MWR prototype (partial)			
W3C DAP Calendar API	A JavaScript API for finding events. Adding or updating events should be done via existing Web platform APIs (e.g. attach file with *.ics or *.ical extension to an html anchor element).	W3C ED			webinos will adapt this API from W3C - additions necessary due to webinos specific issues are listed below this table
Nokia Platform Services 2.0 Calendar API	A JavaScript API for accessing, creating and managing calendar entries in the default calendar. Ability to subscribe to calendar entries being added, modified, deleted.	Symbian WRT 1.1			

webinos specific additions

The W3C Calendar API specification is designed to be agnostic of any underlying calendaring service sources. However, the selection of sources for this calendar information is out of scope for the W3C Calendar specification. For the multi-device useability of webinos, a function needs to be added that allows the retrieval of a list of calandar data across devices using search/discovery criteria, most likely be based on the webinos ServiceDiscovery module.

Messaging API

Description: Send and receive messages of type email, SMS, MMS.
Requirement/architectural reference:

- WOS-US-1.1: Smart Device Integration
- WOS-US-3.3: Social Event Sharing
- WOS-US-5.1: Context Sensitive Triggering
- WOS-UC-TA1-011: Continuous monitoring of diabetic’s blood glucose levels
- WOS-UC-TA7-006: The publicity and privacy of Context Information

- WOS-UC-TA8-001: Receiving local messages and alerts

Phase: 1 (2 for instant messaging functionality)
Webinos responsible: Fraunhofer

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
W3C DAP Messaging API	A JavaScript API for sending SMS, MMS and email based on URL schemes. Supports attachments.	W3C ED	As of May '11 no reading or subscribing of messages is supported. Possibly planned for later API revisions.	Consider extending this API for reading and subscribing as well.	
WAC 2.0 Device APIs: The messaging module	A JavaScript API to send, search for and subscribe to SMS, MMS and email messages.	Fraunhofer MWR prototype (partial, only sms sending)			webinos will base the Messaging API on the WAC/BONDI API due to the availability of receiving messages - an extension of the W3C API would most likely resemble the WAC API, so it is more convenient to start with that. webinos specific remarks follow this table.
GSMA OneAPI SMS, MMS RESTful Version 1.0 (pdf)	A RESTful API that allows sending and receiving of SMS and MMS.	Open Source Reference Implementation in PHP/Java Commercial Pilot in Canada		Uses application/x-www-form-urlencoded and application/json for requests and application/json for responses.	
Nokia Platform Services 2.0	A JavaScript API that allows the sending,	Symbian WRT 1.1			

Messaging API	retrieving and managing of SMS, MMS and email messages.				

webinos specific changes

In addition to the message types e-mail, SMS and MMs, the webinos Messaging API also supports an 'Instant Messaging' (or 'Twitter'-style) type of sending and receiving messages. These are handled similar to SMS messaging, but require a different addressing scheme. Depending on the underlying messaging service, the retrieval of previous messages.

might or might not be possible. To a certain extent, IM-type messages can also be used for sending text based notification messages to devices and applications as well as to users.

To support this, the basic WAC API has been extended by adding a fourth messaging type and an onInstantMessage(in OnIncomingMessage messageHandler) function. This has been done for phase 1 of webinos to allow experimentation, with a more specific and well defined set of messaging modi (e.g. Notifications) to be determined for phase 2.

Filesystem API

Description: Access to device filesystem

Requirement/architectural reference: CAP-DEV-SEMC-002, CAP-DEV-SEMC-003, CAP-DEV-SEMC-014

Phase: 1

Webinos responsible: Stefano Vercelli / Telecom Italia

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
WAC 2.0 Device APIs: The filesystem module	A JavaScript API to access device filesystem	Implementations of WAC WRTs by Obigo, Opera, Aplix, Borqs			
W3C File API & FileReader API	Interface to read user-selected files	Firefox 3.6+ Google Chrome 7+	No gaps identified		webinos will implement this api
W3C File API: Writer	A JavaScript API to write files	TBD, spec as early W3C WD	No gaps identified		webinos will implement this api
W3C File API: Directories and System	A JavaScript API to navigate file system hierarchies	W3C working draft			webinos will implement this api
PhoneGap File	A JavaScript API to	Android			

API	access a mobile device filesystem for reading and writing files.	BlackBerry WebWorks (OS 5.0 and higher) iOS			

Multimedia/Gallery API

Motivation for a Gallery API:

There has been discussion as to whether a gallery is required and whether file access alone is enough. The rationale being that all media files are encapsulated as files.

There are several reasons why a gallery it would be a good idea:

- Remote galleries: file access api will not give to access to remote galleries - a common use case
- Performance: many galleries are very large. Iterating over 10,000 files may not be best way to deal with it
- Metadata and thumbnails: for media the most important thing is the meta -data - a file access method would require implementing all file types and tag formats on JavaScript. This it to complex and inefficient
- Binary data: Javascript and file access methods do not yet support binary data very well

Description: Access to media type files (audio, video, image) and playback.
Requirement/architectural reference:

- WOS-US-3.2: Sharing Music within a community social context
- WOS-US-10.1: User Centric Video Playback
- WOS-UC-TA1-012: Bridging to the Home Network
- WOS-UC-TA4-019: Ad hoc use of Foreign Devices for Playback of Film
- WOS-UC-TA7-004: Finding Devices in Close Physical and Social Proximity

Phase: 1
Webinos responsible: Fraunhofer

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
HTML5 <video> element HTML5 <audio> element	An HTML API to playback video and audio files and streams.	W3C WD implemented with varying supported codecs in: Firefox 3.6+ Firefox Mobile Google Chrome 3+ Android Webbrowser (not			

		inline) Apple Safari 5+ Internet Explorer 9			
Nokia Platform Services 2.0 Media Management API	A JavaScript API to obtain a list of media files and their properties.	Symbian WRT 1.1			
W3C Gallery API	A JavaScript API for searching in multiple gallery objects for media files.	unknown			webinos will adapt this API from W3C - changes necessary due to webinos specific issues are listed below this table
PhoneGap Media API	A JavaScript API to playback (and record) audio files on a mobile device.	Android BlackBerry WebWorks (OS 5.0 and higher) iOS		Playback part obsolete because of HTML5 <audio>, <video>.	
WAC 1.0 AudioPlayer	A JavaScript API that can playback audio files and streams.	Opera Widget Runtime for Android		Obsolete because of HTML5 <audio>	

webinos specific changes

Unlike Contacts and Calendar, the W3C Gallery API already provides a getGalleries method that allows access not only to local, but also to external galleries. While it might be useful to add a "webinos.findServices(user, "Galleries"...)" method as well (for consistency with Calendar and Contacts), this is just a possible addition, but not a necessity.

Payment API

Description: And API to charge users for apps/in-app-purchase/app-usage.

Requirement/architectural reference: [State reference to webinos requirement or architectural component, interface, specification etc]

Phase: 1

Webinos responsible: Fraunhofer

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision

GSMA OneAPI Payment RESTful Version 1.0 (pdf)	A RESTful API to enable to charge mobile subscribers for Web application usage or content.	Open Source Reference Implementation in PHP/Java Commercial Pilot in Canada		Uses application/x-www-form-urlencoded and application/json for requests and application/json for responses.	
PayPal Direct Payment API (Description)	A SOAP based API to allow payments from Web applications using PayPal as a service provider.	Java, ASP.NET and PHP wrappers/SDKs available from PayPal SDK site (https://www.paypal.com/sdk)	Specific to one payment service provider.		
JSR-000229 Payment API (PDF download)	Java specific architecture for payment handling, unchanged since 2005.		Architecture quite Java specific, difficult to map to other languages architectures. No detailed payment functionality specified (payment itself is only via a product name, which needs to be known to the payment provider, hence essentially only direct payments to shops (payment service provider is equal		

			to product provider) is possible)		
Android In-app Billing API (http://developer.android.com/Gide/market/billing/index.html)	Billing API that links in-app purchasing to the Android Market account.	Android SDK API accessing the Android Market service.	Useful API, but specific to Java and Android Market.		

webinos specific changes

Since none of the existing solution provides a sufficiently generic payment solution, webinos will define a generic and simple shopping basket based solution that can be mapped to different underlying payment systems to provide a systems that can address payments on platform bound payment solutions as well as open payment services.

Communication APIs

Description

This section contains investigation results on APIs for communication with other devices, other applications and servers.

Resources

Primary contributor/editor for this API category: Samsung

Supporting contributors/reviewers: SEMC / ISMB / VisionMobile

APIs based on existing standards/implementations

Socket Communication

API's mentioned in this section can be used by application developer to connect to application resources once the device discovered are presented and connected.

Phase 1

Description: To establish communication between two webinos devices.

Requirement/architectural reference: CAP-DEV-SEMC-006 webinos **SHALL** provide means for applications to execute streamed real-time interactive bi-directional communication with two or more other webinos applications running in the same device or running in different devices.

Phase: webinos Phase 1
Webinos responsible: SEMC

Phase 2

"Web RTC": http://rtc-web.alvestrand.com/	Web RTC is still in a start-up phase and the goal is to define/select api's, protocols and codecs that are required to enable real-time bi-directional communication in a Web browser. Web RTC will enable the possibility to implement video/audio conference applications without installing plug-in components. The work with Web RTC will be divided between IETF and W3C/WHATWG. IETF will define/select the protocols and codecs whilst W3C/WHATWG will define the client API's. The new client API's will define the possibility to: 1. get user media from a camera	Experimental implementations exists: https://labs.ericsson.com/developer-community/blog/beyond-html5-peer-peer-conversational-video http://my.opera.com/core/blog/2011/03/23/webcam-orientation-preview	Web RTC does not replaces WebSockets. However, Web RTC is much better suited for exchanging data between peer with real-time characteristics.	Work in W3C started in Web RTC WG. If the work is successful, implementations will most likely exist in all modern browsers before webinos is ready.	It might be provided in browser and no work might be required in webinos project

	or microphone (GetUserMedia API), 2. connect directly to a different peer (PeerConnection API) and 3. stream media and data between the peers (Stream API). As opposed to WebSockets, PeerConnection do not require that connections are relayed via a server. For example, two devices on the same IP sub network or with public ip addresses can connect directly to each other. To establish the connection the current working assumption is that ICE [RFC5245] will be used to negotiate and discover which addresses that can utilized to communicate directly between the peers. In some cases it is not possible to find a direct				

	path between the peers depending on the network topology. In those cases a TURN server is used to relay the traffic between the peers. The working group has tried to agree upon a single session establishment protocol. SIP and XMPP have been proposed but it seems that the group will not agree upon one protocol. The current prediction is that the session protocol will be left out of the standard and initiatives will be started to create open source JavaScript implementations and let the market decide which implementation that will be used.				

A common approach is to use either WebSockets or Server-Sent events and if fails use XMLHttpRequest.

Individual Components Communication

Description: To establish communication with server, which does not require continuous communication.

Requirement/architectural reference: ID-USR-Oxford-37 webinos **SHALL** provides methods to make applications addressable so that other applications can communicate with them.

Phase: webinos Phase 1

Webinos responsible:

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
W3C XMLHttpRequest	It allows performing HTTP client functionality directly from the script. It sends request directly to the WebServer without loading whole Web page. Response are loaded without need to load the page, reply from server can be in XML, text, or in JSON format.	Supported in all browsers	Cross origin website access were used to be blocked but addressed via CORS	It is ideal where communication is required between client and server but does require communication continuously such as submitting form.	It is already supported in modern browser and there is no need of webinos specific implementation

Messaging

Description: To send messages between application running on same device but with different instances.

Requirement/architectural reference: NM-DEV-FOKUS-001 It **SHALL** be possible to exchange information between multiple entities in terms of events.

DA-DEV-ISMB-003 Applications installed on a device **SHALL** be addressable, with multiple instances of the same application being separately addressable.

Phase: webinos Phase 1

Webinos responsible:

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
W3C HTML5	Two independent	It is implemented in at least	No		It is already

Web Messaging (Referred as Channel Messaging in HTML5 Document)	code that want to communicate directly. It supports port to port communication. Ideal for intra communication between two instances that run in different contexts.	Chrome and Firefox as well in Android browser. For example test with http://www.html5test.com/	caching support		supported in modern browser and there is no need of webinos specific implementation

APIs for which no existing standards/implementations exist

Low level event handling API

Description: To send/receive/forward arbitrary data among any entity, in particular being suited for developing higher level APIs relying on data exchange featuring webinos' overlay networking and discovery

Requirement/architectural reference: All "Remote Notifications and Messaging Requirements" (NM-...)

Phase: 1

Webinos responsible/editor: Stefano D'Angelo/ISMB

High level Requirement	**Notes**
Generating events	
Sending/forwarding events	
Registering/unregistering event listeners for incoming events	

Application execution APIs

This section contains investigation results on application execution APIs.

Resources

Primary contributor/editor for this API category: VisionMobile

Supporting contributors/reviewers: Fraunhofer

Description

The Application Execution API allows activation of native and webinos applications installed on the device.

In addition, the API will support a facility for performing late run-time binding between different webinos applications. This facility is modeled after Intent mechanism of Android OS. An intent is an abstract description of an operation to be performed, which holds a passive data structure containing an abstract description of an action to be performed. For example webinos application may request the system to show a map using generic intent mechanism. The run-time will then choose which mapping application should be activated to perform the action. The requesting application is not required to have any knowledge of which specific mapping application is installed in the device.

Policies

Operation of Application Execution API is guided by Application Execution Policies, which can be modified by user. The policies control the following aspects of API operation:

- Enable/disable of activation of native applications
- Enable/disable of activation of webinos installable applications
- Enable/disable of notifications to users when a webinos application attempts to activate another application
- Enable/disable application's ability to discover installed applications
- Enable/disable of logging of operations performed using the API

Application Execution API provides mechanisms for webinos applications to discover current application execution policies, as well as test if specific webinos application is installed in the device, or is running in the device.

Analysis of requirements from WP2.2

The table below lists relevant requirements identified in WP2.2 and the compliance status based on current proposal.

Requirement	**Description**	**Compliance Status**	**Notes**
DA-DEV-SEMC-004	webinos **SHALL** provide means for an Application to detect the availability of a service.	Phase 2	Postponed due to T3.5 decision to drop any policy querying API
DA-ASP-FHG-006	webinos **SHALL** provide means to discover devices that have a specific application installed.	Phase 2	Postponed due to T3.5 decision to drop any policy querying API
DA-DEV-ISMB-002	Applications installed on a device **SHALL** be discoverable, according to security policies.	Phase 2	Postponed due to T3.5 decision to drop any policy

			querying API
NM-DEV-FOKUS-002	It **SHALL** be possible to subscribe to certain event types in order to get notified if the related event occurs.	Phase 2	
NM-USR-IBBT-002	It **SHALL** be possible to notify the user of application launch requests	**Phase 1**	
PS-USR-Oxford-112	The webinos Runtime Environment **SHALL** be capable of specifying fine-grained security policies on all features of devices and user data.	**Phase 1**	
PS-USR-Oxford-37	webinos **SHALL** allow access control decisions to be logged	**Phase 1**	
PS-USR-Oxford-38	webinos **SHALL** allow policies which specify confirmation at runtime by a user when an access request decision is required	**Phase 1**	
PS-USR-Oxford-40	Users **SHALL** be able to modify policies about events before they occur (e.g. up-front policy specification)	**Phase 1**	
PS-USR-Oxford-49	User **SHALL** be able to view & manage application policies	**Phase 1**	
PS-USR-Oxford-52	Users **SHALL** be able to modify policies to allow or deny access to further functionality or data	**Phase 1**	
PS-USR-Oxford-75	The webinos runtime **SHALL** be able to alert the user at runtime using a visual notification	**Phase 1**	
PS-USR_DEV-Oxford-46	Applications **SHALL** request for access rights to any device feature or policy-controlled item prior to accessing it. Applications **MUST** be able to continue to work in a limited manner if an access request to a feature is not granted.	**Phase 1**	
PS-USR-Oxford-62	Applications **SHALL** be isolated from each other. An application **MUST NOT** be able to view or modify another application's data or execution state	**Phase 1**	
LC-DWP-ISMB-116	Lifecycle operations regarding the webinos runtime itself **SHALL** nicely integrate with the package	Phase 2	

	management system of the underlying platform and **SHALL** follow platform-specific common practices.		
CAP-DEV-SEMC-202	It **MUST** be possible to register a background application for automatic execution at device start-up.	Phase 2	
CAP-DEV-SEMC-203	webinos runtime **MUST** be able to start applications based on events, e.g. an incoming message, detected wifi coverage, sensor connected etc.	Phase 2	
CAP-DEV-SEMC-204	The webinos runtime **SHALL** be able to invoke applications by a timer based event.	Phase 2	

Functional API groups

The Application Execution API contains the following groups of functions:

1. Activation of native app. Due to differences in security models between native and webinos apps, activation of native apps requires user consent. Optional completion code can be passed to the initiating webinos application.
2. Activation of installable webinos apps. Activated application should be able to pass results to the originating application. The originating application shall be able to receive asynchronous notifications about completion of the activated application.
3. Sending intents activating generic set of functions. There should be default handler and user selection of alternative handler.
4. Inquire activation policies to discover current system configuration
5. Test if specific webinos application is installed in the device
6. Test if specific webinos application is running in the device
7. Delivery of system-wide events (e.g. boot or shut-down, power-management) that start webinos app automatically whenever event occurs. This is similar to Android broadcast intents. (registration for the events is performed in webinos app manifest file.) This API is different from general event/messaging API, which is intended to be used to deliver information between two *running* apps. Broadcast event reception API is intended to deliver system-wide broadcast events, including starting an app that registered event, in case the app is not running.

The following table shows planned implementation status for Phase 1 and Phase 2.

API name	**Description**	**Phase**
Launch native	Launch native apps installed on the device	Phase 2

Launch webinos	Launch webinos apps installed on the device	**Phase 1**
Launch action	Sending intents activating generic set of functions.	Phase 2
Check policies	Inquire activation policies to discover current system configuration. (Postponed due to T3.5 decision to drop any policy querying API (Berlin meeting))	Phase 2
Check local	Test if specific webinos application is installed in the device	**Phase 1**
Check running	Test if specific webinos application is running in the device	Phase 2
Broadcast event reception	Delivery of system-wide events (e.g. boot or shut-down, power-management) that start webinos app automatically whenever event occurs.	Phase 2

Phase 1 APIs

webinos App Launcher API

Description: API for launching webinos applications installed in the device (local webinos apps)

Requirement/architectural reference:

- NM-USR-IBBT-002 - It **SHALL** be possible to notify the user of application launch requests
- PS-USR-Oxford-36 - webinos APIs shall provide error results when an access control request is denied
- PS-USR-Oxford-62 - Applications **SHALL** be isolated from each other. An application **MUST NOT** be able to view or modify another application's data or execution state

Phase: 1

Webinos responsible: Michael Vakulenko, VisionMobile

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
BONDI 1.1 applauncher Module	A JavaScript API that lists and launches applications installed on a mobile device. The apps are identified	BONDI RI (reference implementation)	Use of MIME types for identification of apps is different from webinos approach where apps are identified using application	Proposal: webinos API will be modelled after BONDI launcher API with necessary modifications to reflect webinos	

	by URI with well-known MIME types.		ID.	approach.	

Check installed app API

Description: API for checking if specific webinos application is installed in the device

Requirement/architectural reference:

- DA-DEV-SEMC-004 - webinos **SHALL** provide means for an Application to detect the availability of a service.
- DA-ASP-FHG-006 - webinos **SHALL** provide means to discover devices that have a specific application installed.
- DA-DEV-ISMB-002 - Applications installed on a device **SHALL** be discoverable, according to security policies.

Phase: 1

Webinos responsible: Michael Vakulenko, VisionMobile

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
BONDI 1.1 applauncher Module	A JavaScript API that lists and launches applications installed on a mobile device. The apps are identified by URI with well-known MIME types.	BONDI RI (reference implementation)	BONDI API allows to map apps to URI names discovering all registered apps. webinos will allow to check for presence of a specific webinos app.	Proposal: webinos API will be modelled after BONDI launcher API with necessary modifications to reflect webinos approach.	

Phase 2 APIs

Activation Policies API

Description: API for discovery of current policy setting

Requirement/architectural reference:

- PS-USR-Oxford-112 - The webinos runtime environment **SHALL** be capable of specifying fine-grained security policies on all features of devices and user data.
- PS-USR-Oxford-37 - webinos **SHALL** allow access control decisions to be logged
- PS-USR-Oxford-38 - webinos **SHALL** allow policies which specify confirmation at runtime by a user when an access request decision is required

- PS-USR-Oxford-40 - Users **SHALL** be able to modify policies about events before they occur (e.g. up-front policy specification)
- PS-USR-Oxford-52 - Users **SHALL** be able to modify policies to allow or deny access to further functionality or data
- PS-USR-Oxford-75 - The webinos runtime **SHALL** be able to alert the user at runtime using a visual notification
- PS-USR_DEV-Oxford-46 - Applications **SHALL** request for access rights to any device feature or policy-controlled item prior to accessing it. Applications **MUST** be able to continue to work in a limited manner if an access request to a feature is not granted.

Phase: 2
Webinos responsible: Michael Vakulenko, VisionMobile

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
No suitable candidate identified				This API may be folded into general policy control framework. The issue is discussed with T3.5. If decided otherwise, a new API will be specified for this functionality	

Native App Launcher API

Description: API for launching native applications installed in the device
Requirement/architectural reference:

- LC-DWP-ISMB-116 - Lifecycle operations regarding the webinos runtime itself **SHALL** nicely integrate with the package management system of the underlying platform and **SHALL** follow platform-specific common practices.
- PS-USR-Oxford-62 - Applications **SHALL** be isolated from each other. An application **MUST NOT** be able to view or modify another application's data or execution state

Phase: 2
Webinos responsible: Michael Vakulenko, VisionMobile

webinos Intent API

Description: API for sending intents activating generic set of functions

Requirement/architectural reference: Need to clarify with T3.1

Phase: 2
Webinos responsible: Michael Vakulenko, VisionMobile

Candidate	Short Description	Implementation Status	Gaps	Notes	Decision

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

API					
Web Introducer	Web Introducer concept was initiated by Google. SEMC and Mozilla is cooperating with Google on the concept. The goal is to make the concept a W3C recommendation specification. Web Introducer enables Web applications to discover a user's personal resources, no matter where they are hosted or produced, and gain permission to interact with them via a one-click user interaction.	Currently there is an experimental pure HTML+JS implementation of the Web Introducer API that works in currently deployed modern browsers. Experimental applications are: - Share Link - Get image	TBD	More detailed information about this API is available at Web Introducer	

Activation Test API

Description: API for checking if specific webinos application is running in the device

Requirement/architectural reference:

DA-DEV-SEMC-004 - webinos **SHALL** provide means for an Application to detect the availability of a service.

Phase: 2
Webinos responsible: Michael Vakulenko, VisionMobile

Broadcast event reception API

Description: API for reception of system-wide events. This API is different from general event/messaging API, which is intended to be used to deliver information between two *running* apps. Broadcast event reception API is intended to deliver system-wide broadcast events, including starting an app that registered event, in case the app is not running.

Requirement/architectural reference:

- NM-DEV-FOKUS-002 - It **SHALL** be possible to subscribe to certain event types in order to get notified if the related event occurs.
- CAP-DEV-SEMC-202 - It **MUST** be possible to register a background application for automatic execution at device start-up.
- CAP-DEV-SEMC-203 - webinos runtime **MUST** be able to start applications based on events, e.g. an incoming message, detected wifi coverage, sensor connected etc.
- CAP-DEV-SEMC-204 - The webinos runtime **SHALL** be able to invoke applications by a timer based event.

Phase: 2

Webinos responsible: Michael Vakulenko, VisionMobile

Supporting information

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
WAC 2.0 Device APIs: Web Standards/ 2.8. URI Schemes	URI schemes are the defined method for WAC applications to launch other applications.	Specification is Proposed Released Version	Just local apps.	The system defines which application is started depending on scheme or file type. E.g. `<a href="tel:+123">dial</a>` would start the dialer or `<a href="data:application/pdf:...>open</a>` would open the pdf data.	
W3C DAP The Application Launcher API	A JavaScript API for launching native applicatinos on a device.	W3C ED	Just local apps.	Current Editor's Draft allows to query installed applications and set default applications. So you can either let the system decide which app to start or explicitly start an application. But multiple members of DAP want just URI schemes, few voices suggest a module is needed for mobile (See minutes and current charter). Note by Claes 20110331: According to DAP phone meeting 2011-03-30 there is a decision to not include the App Launcher in the charter.	
Mozilla Open Web Apps JavaScript API	Open Web Apps from Mozilla is a spec that can package a website and make it installable in the browser. The JavaScript API handles installation and	Experimental Firefox 4 add-on Experimental Google Chrome extension		Besides listing and launching applications, installing and uninstalling other applications is also supported.	

	management functions.				
BONDI 1.1 applauncher Module	A JavaScript API that lists and launches native applications on a mobile device.	BONDI RI (reference implementation)	Just local apps.	Doesn't support setting default applications. Application can be explicitly started. E.g. `bondi.applauncher.launchApplication(succCallB, errCallB, "file:/bin/fpsgame");`	

Background tasks

Description: To run task in background. This is an independent entity and result of worker thread is updated to event.

Requirement/architectural reference: CAP-DEV-FHG-200 The webinos runtime **SHALL** be able to run applications in the background.

Phase: 2

Webinos responsible: N.N.

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
W3C Web Workers	Web Workers instantiate scripts which run in parallel and does not require any input from UI or script handling page. To handle I/O operation it can make use of XMLHttpRequest to get output. Results of Worker thread are updated to the subscribed event.	Implementation of Web Workers is present in all browsers, some have basic functionality and some support shared worker functionality	None identified, except it has quite high performance startup time and high memory consumption.	Web workers are intended to facilitate multi-threading in Web apps. This is good for background tasks that do not require to update the DOM tree/UI directly. However, Web workers are not feasible for background jobs that needs to be started at system start up etc.	

Discovery APIs

Description

This section contains investigation results on APIs for device and service discovery.

Resources

Primary contributor/editor for this API category: Samsung

Supporting contributors/reviewers: Fraunhofer / SEMC / T-Systems / W3C / DoCoMo

Overview of Discovery Technologies

Each interconnect technology can have its own discovery mechanisms, and some have several, each with their own terminology. Webinos will need to provide an overlay that abstracts away from the variations and which offers simple naming for end users and Web developers. The overlay will need to store the mapping from user-friendly names to the underlying data registered for the device and needed to communicate with it.

Interconnect technologies include:

- 3G, WiFi, WiMAX, Bluetooth, ZigBee, NFC, RJ45, USB, IEEE 1394, ...

Only some of these support IP directly. Webinos should provide a means for an IP accessible device to proxy for devices that are not IP addressable.

IP based networks

For IP based networks, three such mechanisms are:

Multicast DNS

Invented by Apple for simplifying the connection of devices in home networks. The Apple implementation is called "Bonjour". An open source equivalent is "Avahi". Devices start by randomly picking a link local IP address in the range (169.254.*.*) and making an ARP request to see if this address is already in use.

Devices assign themselves name in the ".local" domain, and will adjust this if they detect other devices with the same name. Users can assign human meaningful names with spaces in them. User agents query for devices with multicast UDP requests. The devices check for a match and respond with a multicast UDP packet with the IP address and port number, the device's domain name, and a list of protocol and/or service names. The device domain names use human meaningful conventions e.g. "Dave's laptop._workstation.local". A nice feature is the ability for a device to report on other devices including external services such as the BBC news on the Web, or a hotel's local Web server giving details of the hotel's services. Multicast DNS isn't designed to support tens of thousands of devices, but this can be worked around with discovery hubs.

More information:

- Introductory talk
- Stuart Cheshire's website
- Avahi introduction

On Linux, try

```
mdns-scan
```

or

```
avahi-browse -a -v
```

This only found my Linux workstation on my WiFi network and not the ADSL Modem, nor the Samsung laser printer.

Simple Service Discovery Protocol (SSDP)

Invented by Microsoft and considerably more complicated than Multicast DNS. SSDP is a UPnP-based protocol and it uses HTTP for notification announcements that give a service type URI and a unique service name.

On linux try

```
gssdp-device-sniffer
```

On my WiFi network this found the ADSL Modem and the Laser printer, but not the Linux workstations.

Service Location Protocol (SLP)

Defined in RFC 2608 and supported by Hewlett-Packard's network printers, Novell, and Sun Microsystems, but ignored by some other large vendors. SLP works over UDP or TCP. On UDP, devices listen on port 427 for multicast requests. Devices advertise themselves with a URI like

```
service:printer:lpr://myprinter/myqueue
```

This may be supplemented by a list of attributes, e.g.

```
 (printer-name=Hugo),
 (printer-natural-language-configured=en-us),
 (printer-location=In my home office),
 (printer-document-format-supported=application/postscript),
 (printer-color-supported=false),
 (printer-compression-supported=deflate, gzip)
```

If the data doesn't fit in a single packet, a flag is given that the user agent can act on to request the SLP info via TCP. SLP also supports discovery agents and presumably this helps with scaling up to larger networks with bridged local networks.

UPnP and DLNA

On linux try

```
upnp-inspector
```

On my network this found the ADSL modem but not my laser printer.

Other interconnect technologies

WiFi

For WiFi it is possible to detect access points and what kind of encryption they are using, if any, as well as devices operating in ad-hoc mode. It should be possible to detect device MAC addresses.

USB

For USB see the source code for the Linux lsusb command. This lists the bus and device number, the device ID and a human readable description e.g. Logic3 / SpectraVideo plc A4Tech SWOP-3 Mouse, and Microdia Sonix Integrated Webcam.

Bluetooth

For Bluetooth, a scan shows the device ID and type, e.g. phone. This id can be used as a proxy identifier for people, i.e. who is present in the room or nearby. There is a small set of known device types, e.g. phone, input device, headset, modem, computer, network, camera, printer or video device. Users can provide a meaningful name for their device e.g. "Me" as seen for a phone in a car that drove past my window!

APIs based on existing standards/implementations

Low level Service Advertising APIs

Description: Service to advertise its availability and capabilities APIs

Requirement/architectural reference:

- DA-DEV-SEMC-001. The webinos network **SHALL** provide means for a service to expose its availability and capabilities on a webinos network.

Phase: webinos phase 1

Webinos responsible: Ziran/Samsung

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
Avahi Register Services API based on DNS-SD	It uses DNS service locator (SRV), Text Record(TXT), and Pointer recorder (PTR) records to advertise Service Instance Names. The hosts offering services publish details of available services: instance, service type, domain name and optional configuration parameters. In case of mDNS, each computer on the LAN stores its own list of DNS resource records (e.g., A, MX, SRV) and joins the mDNS multicast group. If a unicast DNS is available, two ways to advertise services:	Avahi had already become the de-facto standard implementation of mDNS/DNS-SD on free operating systems such as Linux.	Native Codes. It suits Local or wide-area network with the same domain.	This is low level API - one option is to wrap native code and expose as JavaScript Object method - here is an example	To be considered for phase 2

	via dynamic DNS server or manually add DNS records to describing the services to add.				
Strophe.js API for XEP-0124 BOSH	Strophe.js provides Javascript library for BOSH implementation, which enable XMPP over HTTP. For service advertisement, XEP-0060 Publish-subscribe APIs shall be implemenated on the top the existing Strophe.js library. XEP-0060 Publish-subscribe specifies that an entity publishes information to a node at a publish-subscribe service. The pubsub service pushes an event notification to all entities that are authorized to learn about the published information.	Strophe.js is well used. It has been tested on Firefox 1.5, 2.x, and 3.x, IE 6, 7, and 8, Safari, Safari Mobile, Google Chrome, and it should also work on the mobile Opera browser as well as the desktop Opera browser.	Strophe.js does not needs particular support for specific XEP. Expanding XEP-0060 implementation based on Strophe.js should be straightforward		No API will be developed by WP3.2 as a downloadable JS library is available

Low level Find Service API

Description: Allows applications to find services based on description

Requirement/architectural reference:

- DA-DEV-SEMC-002. webinos **SHALL** provide the means to discover new service advertised on a webinos network.
- DA-DEV-FHG-001. webinos shall provide means for applications to be capable of discovering other devices based on a User.
- DA-USR-ISMB/FHG-005. webinos shall provide means for an application to discover and address applications and services offered by other users.

Phase: webinos phase 1

Webinos responsible: Ziran/Samsung

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
Avahi Browse Services API based on DNS-SD	To browse for available services. In case of mDNS, Updates about the new service availability is done by sending the multicast advertisement		same as advertisement	same as advertisement	To be considered for phase 2
Strophe.js API for XEP-0124 BOSH"	To find service, XEP-0030: Service discovery API shall be implemented on the top of Strophe.js. XEP-0030 specifies discovering service via JID. (1) the identity and capabilities of an entity, including the protocols and features it supports; and (2) the items associated with an entity, such as the list of rooms hosted at a multi-user chat service.	Strophe.js has been tested on most well-used browsers (see above).	Strophe.js does not needs particular support for specific XEP. To implement discovery over BOSH, simply send IQ-get stanzas to the server with a certain namespace.		No API will be developed by WP3.2 as a downloadable JS library is available

APIs for which no existing standards/implementations exist

High level Discovery API

Description

Currently there exist several methods to do service discovery. This has been explored in the state of the art investigation for service discovery. Some of these methods are fairly well deployed and used such as Bluetooth service discovery, Universal Plug & Play, mDNS or DNS Service Discovery. However, neither of these discovery methods has been exposed to Web application developers. In addition methods like Universal Plug & Play, mDNS and DNS SD do not have any robust security model.

The goal with the webinos high-level service discovery API is to be able to provide a simple API for application developers. The API shall provide the means to discovery services within personal zones and from low level service discovery methods supported by the device. The fact webinos uses a overlaying network, service discovery will not be limited to local services but will also enable to discover remote services. The API hides the complexity for communicating with services residing in a different peer in a trusted manner.

Requirement/architectural reference

The following 2.2 requirements are applicable for the high level Discovery API.

Phase: webinos phase 1

- DA-DEV-SEMC-001: The webinos network **SHALL** provide means for a Service to Expose its availability and capabilities on a webinos Network.
- DA-DEV-SEMC-002: webinos **SHALL** provide the means to discover new services advertised on a webinos Network.
- DA-DEV-SEMC-004: webinos **SHALL** provide means for an Application to detect the availability of a service (such as being able to detect when a service is started, stopped or not available due to out of proximity).
- DA-DEV-SEMC-005: webinos shall provide means for an Application to find devices and services that are available on a webinos network, based on the Device and Service Description.
- DA-DEV-SEMC-006: webinos **SHALL** provide means for an Application to find devices and services in close proximity of the device
- DA-DEV-SEMC-007: webinos **SHALL** provide means for an Application to find devices and services based on the physical location of the current user device.
- DA-ASP-FHG-001: webinos **SHALL** provide means for applications to be capable of discovering other devices based on a User.
- DA-ASP-FHG-006: webinos **SHALL** provide means to discover devices that have a specific application installed.
- DA-DEV-ISMB-001: webinos **SHALL** provide means for applications to discover and address features and services available on devices owned by the user even if such devices are not directly connected to the device on which the application is running.
- DA-DEV-ISMB-004: It **SHALL** be possible to address sensors and actuators, which do not provide webinos support.
- DA-DEV-ISMB/FHG-005: webinos **SHALL** provide means for applications to discover and address applications and services offered by other users.
- DA-DEV-ISMB-006: It **SHALL** be possible to address webinos enabled devices based on their user information.
- DA-DEV-NTUA-002: webinos **SHALL** provide the means to Applications to identify an event occurring in a device.

Phase: webinos phase 2

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

- DA-DEV-SEMC-008: webinos **SHALL** provide means for an Application to discover which user that currently is using a discovered device outside the personal webinos Network.
- DA-DEV-ISMB-003: Applications installed on a device **SHALL** be addressable, with multiple instances of the same application being separately addressable.
- DA-DEV-NTUA-003: webinos **SHALL** provide the means to Applications to be capable of discovering other devices based on a piece of contextual information.
- DA-DEV-NTUA-004: webinos **SHALL** be able to calculate the social proximity of webinos Devices.

Webinos responsible/editor: Anders Isberg / SEMC

Security and Privacy APIs

Description

This section contains an overview of the required APIs for the webinos security architecture and background information about related work.

Resources

Primary contributor/editor for this API category: Oxford

Supporting contributors/reviewers: Polito, DOCOMO

Aims for Security and Privacy APIs

The following requirements must be satisfied:

- PS-DEV-VisionMobile-11: webinos applications **SHALL** have access to the standardized webinos user privacy preferences
- PS-DEV-Oxford-56: Applications shall be aware of changes to policies and may alter their behaviour as a result
- ID-DEV-POLITO-005: A webinos device may be able to provide Attestation of the webinos platform.

However, the first two of these requirements have been moved to phase 2 of the implementation as the security architecture is further clarified.

In addition, various requirements (PS-USR-Oxford-103, PS-USR-Oxford-26) require users to authenticate through device-specific capabilities. Therefore, an authentication API has been specified.

Aims for Security and Privacy APIs in phase 2

The following proposals will be investigated in phase 2 of the webinos platform:

- Expose user privacy preferences to applications, including data retention policies and access control.

- Provide applications with the ability to query the *obligations* they will be under should they use a particular data item.
- Provide applications with the ability to query their own permissions - what have they been granted access to.

- Expose device security capabilities
    - Statements about whether the device can provide certain security features and to what level of assurance is provided. The main challenge with implementing this would be validating the results.
    - These features might include (a) authentication methods, (b) attestation, (c) secure storage, (d) isolated execution, (e) auditing and logging, (f) event reporting / monitoring of platform state, (g) credentials for keys held in the platform & revocation lists.
- Depending on future use cases and requirements, webinos may choose to expose certain cryptography APIs to applications.
- Remote management APIs, perhaps like the Android Device Admin API

Existing standards

The following APIs for security (e.g. cryptography and authentication) and for attestation exist.

- APIs for cryptography and transport sessions
    - javax.crypto for cryptography
    - javax.crypto.interface for Diffie-Hellman
    - javax.crypto.spec for specification of crypto-parameters
    - java.net.ssl for SSL/TLS
- APIs for authentication
    - java.security.auth for authentication credential management
- APIs for policy management
    - android.app.admin to manage device policies
- APIs for DRM
    - android.drm for Digital Rights management
- Device status and attestation
    - WAC device status API as currently suggested in HW_Resource_APIs.
    - Trusted Computing Group Trusted Software Stack .
- Secure coding APIs
    - OWASP ESAPI - useful for input validation, credit card validation, etc.

APIs for which no existing standards/implementations exist

Attestation API

Description: The purpose of this API is to provide a secure means to query the device to find out the identity and integrity of running software. The example use case is Trusted Computing Mobile Reference Architecture. However, this is aimed at a lower layer than webinos. The aim of the attestation API is simply to allow access to existing functionality.

Requirement/architectural reference: ID-DEV-POLITO-005, ID-DEV-POLITO-006, ID-DEV-POLITO-007, ID-DEV-POLITO-008

Phase: webinos phase 1

Webinos responsible/editor: John Lyle

High level Requirement	Notes
This API shall be capable of exposing basic TCG attestation capabilities	
This API shall not rely upon a specific hardware implementation	
This API shall provide applications with the ability to fetch authenticated data about the runtime state of the platform	

Authentication API

Description: Provides information to applications about the current authentication status of users, as well as allowing applications to request re-authentication..

Requirement/architectural reference: PS-USR-Oxford-103, PS-USR-Oxford-26

Phase: 1

Webinos responsible/editor: John Lyle

High level Requirement	Notes
This API shall allow applications to request that the user authenticate to the device	
This API shall allow applications to find out when and how the user last authenticated	
This API shall not expose identity information about the user	

User profile and context APIs

Description

This section contains investigation results on user profile APIs and context APIs.

The user profile API defines attributes and methods to access to user related information (e.g. name, nickname, gender birthday, etc.) while the application data API provide information about application related information (e.g. installed application).

Resources

Primary contributor/editor for this API category: T-Systems
Supporting contributors/reviewers: DoCoMo, NTUA

Analysis of requirements from WP2.2

The table below lists relevant requirements identified in WP2.2 and the compliance status based on current proposal.

Requirement	**Description**	**Compliance Status**	**Notes**
CAP-DEV-SEMC-010	webinos **SHALL** provide means for applications to access user's profile data.	**Phase 1**	
CAP-DEV-SEMC-011	webinos **SHALL** provide means for applications to access user's preference data	Phase 2	An secure and optimal method to store user preferences must be found to provide privacy aspects and avoid a blow up of user preferences (e.g. different applications would like to store the same information in the user preferences -> ColorBlind:RedGreen is the same as ColorBlind:GreenRed).
ID-USR-POLITO-100	webinos components that have to be shared or referenced (device, application, data, user) **SHALL** be identifiable.	**Phase 1**	The user profile has a unique id.
DA-DEV-ambiesense-040	It **MUST** be possible for applications to share context information across devices, so that for instance social context can be updated when friends enter/ leave the same situation.	Phase 2	In phase 1 the user profile API implements social contact information. Further contextual information must be evaluated for phase 2.

PS-DEV-Oxford-86	The webinos runtime **SHALL** support the confidential storage of user credentials including usernames and passwords.	Phase 2	A secure method to store login credentials must be evaluated for phase 2.
PS-USR-IBBT-005	The webinos system **SHOULD** store associations between device, user and context information securely and provide this information based on user preferences.	Phase 2	Same as CAP-DEV-SEMC-011.
PS-USR-VisionMobile-10	webinos **SHALL** allow users to express their privacy preferences in a consistent way.	Phase 2	Same as CAP-DEV-SEMC-011.
PS-USR-VisionMobile-11	webinos applications **SHALL** be able to query the webinos user privacy preferences.	Phase 2	Same as CAP-DEV-SEMC-011.
NC-DEV-IBBT-0015	Applications **MUST** be able to access the user's general webinos preferences (with the permission of the user).	Phase 2	Same as CAP-DEV-SEMC-011.

Phase 1 APIs

APIs for which no existing standards/implementations exist

User Profile API

Description: User Information

*Requirement/architectural reference:

- CAP-DEV-SEMC-010: webinos **SHALL** provide means for applications to access user’s profile data.
- ID-USR-POLITO-100: webinos components that have to be shared or referenced (device, application, data, user) **SHALL** be identifiable.

Phase: webinos phase 1

Webinos responsible: Ronny Gräfe / George Gionis

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
W3C Contacts API	Contacts provides a basic list of information about a person, but is insufficient for webinos needs.	Contacts provide only 'real life' information about a user, but no technical information (like preferences). It also doesn't allow granularity of access. And has no specific API for the user (as opposed to other contacts).			Decision was made to build own API for webinos, based on elements from Contact and Portable Contacts. See note below table.
Portable Contacts	The reference presented by George Gionis added account information for user accounts for external social network profiles, which is useful for context awareness and calculation of social proximity. We can base user profile information on the account information suggested by portable contacts..				Decision was made to build own API for webinos, based on elements from Contact and Portable Contacts. See note below table.

The data accessible through the API for user profiles will be based on the W3C contacts information with additional information for social network profiles based on Portable Contacts.

Phase 2 APIs

User Profile API

Description: User Information

*Requirement/architectural reference:

- CAP-DEV-SEMC-011: webinos **SHALL** provide means for applications to access user's preference data.
- DA-DEV-ambiesense-040: It **MUST** be possible for applications to share context information across devices, so that for instance social context can be updated when friends enter/ leave the same situation.

- PS-DEV-Oxford-86: The webinos runtime **SHALL** support the confidential storage of user credentials including usernames and passwords.
- PS-USR-IBBT-005: The webinos system **SHOULD** store associations between device, user and context information securely and provide this information based on user preferences.
- PS-USR-VisionMobile-10: webinos **SHALL** allow users to express their privacy preferences in a consistent way.
- PS-USR-VisionMobile-11: webinos applications **SHALL** be able to query the webinos user privacy preferences.
- NC-DEV-IBBT-0015: Applications **MUST** be able to access the user's general webinos preferences (with the permission of the user).

Phase: webinos phase 2
Webinos responsible: N.N.

Candidate API	Short Description	Implementation Status	Gaps	Notes	Decision
No suitable API identified					

7. Tools for API specifications

The tools used to create the webinos specifications are:

- widlproc
- Redmine with Git

All mentioned tools are open source software and available for Windows, Mac OS and Linux.

Web IDL/widlproc

The webinos API specifications are written in Web IDL (http://www.w3.org/TR/WebIDL/), an interface definition language. It is the same language that is used by the W3C. These Web IDL files are further annotated with explanatory comments that cover the meaning of each interface module, attribute, method, method argument and method return type.

Having the specifications in Web IDL enables using tools to generated code stubs from the interfaces wich can be used as a basis for implementing the webinos API specifications.

The widlproc command line tool (http://widl.webvm.net/) is used to generate HTML documentation from the Web IDL file. Viewing and referencing these HTML pages is much more comfortable and they can be used as a guide and documentation for both developers that want to implement those APIs or developers that want to use these APIs.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

This approach is quite flexible and scriptable: e.g. it is very easy to create scripts that process many Web IDL files and based on that auto-generate the documentation. It also allows for a webinos own look & feel by customizing the CSS and HTML that is generated.

Git and Redmine Infrastructure

The webinos project members use Redmine as collaboration platform. Redmine is a web-based project management tool providing a wiki, bug-tracking, etc. It is the installation that is running behind http://dev.webinos.org.

Git is used to track and manage the Web IDL files for the API specifications. Git is a distributed revision control system (http://git-scm.com). Project members submit their changes to the repository on dev.webinos.org and pull changes other members have contributed, so that everybody has a consistent state of the ongoing specification process.

Every time a change is submitted to the server the widlproc tool (see above) is used to generate new HTML documentation for the API specifications that can then be viewed by everybody at http://dev.webinos.org/specifications/draft/.

Since Redmine provides all these features readily integrated, it makes it easy and comfortable for users and administrators of the system at the same time. Users quickly find all relevant informations at their fingertips since wiki pages, issues management and task management, calendars and the source repository are all linked together and can reference each other. The webinos server administrators on the other hand only need to take care of a single piece of software, which saves quite some time. It is also easy to extend the basic functionality with provided plugins. The Redmine software is actively maintained by its authors and enjoys great distribution.

8. JS API design patterns and guidelines

During the webinos API specification work common patterns and design criteria have been followed. webinos APIs are specified through WebIDL together with text descriptions that define the behaviour of the API interfaces, methods and attributes. Apart from the webinos defined APIs, webinos refers to other APIs defined by other parties (e.g. W3C and WAC). Webinos is not going to modify those APIs, so some of them may not be aligned with the patterns defined for webinos. In most of the cases, the webinos design patterns and guidelines do not prescribe a unique solution for the different API characteristics, but describes what are the feasible alternatives, what are the preferred one(s) and the situations in which different alternatives should be used.

The webinos API patterns and guidelines are described in this document: Device APIs - Design Patterns and Guidelines

9. List of webinos phase 1 API specifications

This sections lists all webinos API specifications included in the webinos WP 3.2 delivery.

Webinos API specifications are of two major types:

- A new API specification created within the webinos project.
- A wrapper specification that references an existing API specification defined elsewhere, e.g. by W3C or WAC. Optionally a wrapper specification can define webinos modifications to the referred specification.

All webinos API specifications are available here: webinos Device APIs

webinos base and generic objects/interfaces

API name	Specification	Editor	Comments
webinos core API module	webinos Core module.	Claes Nilsson/SEMC	

Discovery and access to remote services

API name	Specification	Editor	Comments
Service discovery API	webinos Discovery module.	Anders Isberg/SEMC	

HW Resource APIs

API name	Specification	Editor	Comments
Geolocation API	W3C Geolocation API	W3C Geolocation WG	
Device Orientation API	W3C DeviceOrientation Event	W3C Geolocation WG	
Generic SensorActuator API	webinos Sensor module	Claes Nilsson/SEMC	
Media Capture API	W3C Media Capture	W3C DAP WG	
Devicestatus API	WAC 2.0 devicestatus module	WAC	
Devicestatus vocabulary	webinos Device Status Vocabulary	WAC, Stefano Vercelli/TIM	WAC 2.0 devicestatus module with added webinos aspects

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Device Interaction API	WAC waikiki deviceinteraction module	WAC	
TV and STB control API	webinos TV Control module	Alexander Futasz/FHG	
Vehicle API	webinos Vehicle module	Simon Isenberg/BMW	
NFC API	webinos NFC module	Stefano Vercelli/TIM, Hans Myrhaug/AmbiSense	

Application Data APIs

API name	**Specification**	**Editor**	**Comments**
Contacts API	W3C Contacts API	W3C DAP WG	
Calendar API	W3C Calendar API	W3C DAP WG	
Messaging API	webinos Messaging module	WAC, Christian Fuhrhop/FHG	WAC 2.0 Messaging module with webinos modifications
File Reader API	W3C File API	W3C Web Apps WG	
File Writer API	W3C File API: Writer	W3C Web Apps WG	
File API: Directories and System	W3C File API: Directories and System	W3C Web Apps WG	
Gallery API	W3C Gallery API	W3C DAP WG	
Payment API	webinos Payment module	Christian Fuhrhop/FHG	

Communication APIs

API name	**Specification**	**Editor**	**Comments**
Event handling API	webinos Event Handling module	Stefano D'Angelo/ISMB	

Application Execution APIs

API name	**Specification**	**Editor**	**Comments**

Widget execution API	webinos Widget module	Andre Paul/FHG	Based on W3C Widget Interface
Application Launcher API	webinos AppLauncher module	Michael Vakulenko/VisionMobile	Based on BONDI 1.1

Security and Privacy APIs

API name	Specification	Editor	Comments
Platform attestation API	webinos Attestation module	John Lyle/Oxford	
User Authentication API	webinos Authentication module	DOCOMO, Oxford (John)	

User Profile and Context APIs

API name	Specification	Editor	Comments
User Profile API	webinos Userprofile module	Ronny Gräfe/T-Systems	Based on W3C DAP Contacts and Portablecontacts
Context API	webinos Context module	Heiko Desruelle/IBBT	Uses W3C SPARQL specification as context query language

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

10. APIs Specified by webinos

API Summary

Specification	Summary
The attestation module	The attestation API provides a secure method of querying the underlying device hardware to find out the identity and integrity of running software. This API should allow applications to communicate with hardware security systems, such as a Trusted Platform Module (defined in the Trusted Computing Group specifications).
The authentication module	Authentication API for providing applications with information about whether the current user has authenticated, and requesting re-authentication at runtime.
The context module	The Context API defines the high-level interfaces required to obtain access to a user's context data. The API supports two basic ways of accessing context data: The Context API defines the high-level interfaces required to obtain access to a user's context data. The API supports two basic ways of accessing context data:
The events module	The webinos Event Handling API provides means to exchange data in terms of events among addressable entities (e.g., applications, services), either locally or remotely.
The AppLauncher module	The application execution API allows activation of webinos applications installed locally on the device. The API is modelled after BONDI v1.1 AppLauncher API.
The messaging module	The messaging API provides access to the following capabilities: Sending messages through different technologies: SMS, MMS, Email and Instant Messages. Search for messages in the different folders. Subscribe for being notified upon incoming message events.
The nfc module	Near Field Communication is a kind of radio-frequency identification (RFID) technology that uses short-hold wireless communication to transfer messages between wireless NFC devices and NFC tags. The wireless tags are physically attached onto/ mounted nearby a physical object.
The payment module	This API provides generic shopping basket functionality to provide in-app payment.
The sensors module	The webinos Generic Sensor API provides Web applications with an API to access data from sensors in the device, connected to the device or in another

Specification	Summary
	device.
The discovery module	The webinos Discovery API provide Web applications with an API to discover services without any previous knowledge of the service. The Discovery API is not limited to discovery of local services but also enables discovery of remote services.
The tv module	The interface provides means to acquire a list of tv sources, channels and their streams.
The userprofile module	This API offers access to information of the user. UserProfile API is an extension of webinos Contact API to gather basic information about the user (e.g. name, nickname, gender, birthday, etc.) and extends it with social network attributes from Portablecontacts from August 5, 2008 (http://portablecontacts.net/draft-spec.html). These social network attributes are a simple pointer where the webinos user has non-webinos profiles. These information could be used by an application to query an external API for an additional information (e.g. query the Facebook Graph API for the buddylist).
The vehicle module	The webinos vehicle API provides access to specific vehicle data. It is derived from W3C's DOM Level 3 Events model and defines event types for retrieving information about the vehicle including trip computer data, gears or park sensors. Furthermore it offers methods for interacting with the on-board navigation system. The geolocation, speed and acceleration can be retrieved using the geolocation and device orientation API.
The webinoscore module	This specification defines the common interface from which all webinos APIs are can be accessed as well as several interfaces that are commonly reused.
The widget module	This specification defines the common widget interface. The webinos application packaging is based on W3C Widget Specifications, thus, the interface definition is also based on W3C. Namely W3C Widget Interface (http://www.w3.org/TR/2011/WD-widgets-apis-20110607/). This specification recaptures the W3C specification while adding webinos specific extensions.

APIs: The attestation module

webinos API Specifications

30 Jun 2011

Authors

- Andrew Martin; <andrew.martin@cs.ox.ac.uk;>
  John Lyle; <john.lyle@cs.ox.ac.uk>

Abstract

Attestation: find the identity and integrity of running software

Summary of Methods

Interface	Method
X509	
TBSCertificate	
Validity	
namePair	
AlgorithmIdentifier	
SubjectPublicKeyInfo	
attestationData	
WebinosAttestationInterface	attestationData attestPlatform(byte [] nonce, SubjectPublicKeyInfo key) SubjectPublicKeyInfo getAttestationKey() X509 getKeyCredential(SubjectPublicKeyInfo key)
WebinosAttestation	

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

1. Introduction

The attestation API provides a secure method of querying the underlying device hardware to find out the identity and integrity of running software.
This API should allow applications to communicate with hardware security systems, such as a Trusted Platform Module (defined in the Trusted Computing Group specifications).

However, the usual trusted computing specifications are generally aimed at a lower layer of the operating system than webinos.
The aim of the attestation API is to expose existing low-level functionality to Web applications.

Requirement/architectural reference: ID-DEV-POLITO-005, ID-DEV-POLITO-006, ID-DEV-POLITO-007, ID-DEV-POLITO-008

Example use of attestation, taken from the Security Architecture Documentation (D3.5):

1. User starts an application called "MyBankApp"
2. MyBankApp communicates with a remote webserver at http://bank.example.com
3. http://bank.example.com asks MyBankApp to attest to its current status
4. MyBankApp uses the Attestation API to request a public key and key credential for the local device, Peter's Smartphone.
- App calls getAttestationKey() and getKeyCredential() to retrieve this information
5. The key credential is forwarded to http://bank.example.com
6. http://bank.example.com assesses the credential and checks to see whether the endpoint is a trusted device.
- If not, attestation fails.
7. http://bank.example.com gives MyBankApp a fresh nonce, a 20 byte random value.
8. MyBankApp uses this nonce and the public key with the attestation API on Peter's Smartphone:
- call attestPlatform(nonce, key)
9. Peter's Smartphone returns attestation data, which includes a log of the integrity of the platform ("trustChain"), as well as validation data from the hardware trusted platform module ("validation data") with schema "TPM_Quote".
10. These values are passed on to http://bank.example.com
11. http://bank.example.com assesses the validation data and the integrity log using standard TCG techniques see http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14 and related documents
- If the platform integrity is not trusted, attestation fails
- If the validation data is not trusted, attestation fails
12. http://bank.example.com passes MyBankApp a temporary token which gives it access to the http://bank.example.com banking capabilities
13. User authentication is requested via the authentication API
14. The application is now able to perform transactions using remote http://bank.example.com APIs.

2. Interfaces

2.1. X509

Definition of X509 certificate data structure taken from the ASN1 X509 Specifications and RFC 2459 http://www.ietf.org/rfc/rfc2459.txt
We expect for TCG attestation that the SKAE extension to also be introduced, definition:
http://www.trustedcomputinggroup.org/files/resource_files/876A7F79-1D09-3519-AD321B21144AE93C/IWG_SKAE_Extension_1-00.pdf
however it is not defined here.

```
        interface X509 {
                readonly attribute TBSCertificate certificate;
                readonly attribute AlgorithmIdentifier signatureAlgorithm;
                readonly attribute byte[] signature;
        };
```

2.2. TBSCertificate

as defined in http://www.ietf.org/rfc/rfc2459.txt

```
        interface TBSCertificate {
                readonly attribute DOMString version;
                readonly attribute Integer serialNumber;
                readonly attribute AlgorithmIdentifier signature;
                readonly attribute namePairArray issuer;
                readonly attribute Validity validity;
                readonly attribute namePairArray subject;
                readonly attribute SubjectPublicKeyInfo subjectPublicKeyInfo;
                readonly attribute Any? extensions;
        };
```

2.3. Validity

as defined in http://www.ietf.org/rfc/rfc2459.txt

```
        interface Validity {
                readonly attribute Date notBefore;
                readonly attribute Date notAfter;
        };
```

2.4. namePair

```
A single pair of key and value.
        interface namePair {
                readonly attribute DOMString key;
                readonly attribute DOMString value;
        };
```

2.5. AlgorithmIdentifier

as defined in http://www.ietf.org/rfc/rfc2459.txt

```
        interface AlgorithmIdentifier {
            readonly attribute DOMString identifier;
                readonly attribute DOMString? parameters;
        };
```

2.6. SubjectPublicKeyInfo

as defined in http://www.ietf.org/rfc/rfc2459.txt

```
        interface SubjectPublicKeyInfo {
                readonly attribute AlgorithmIdentifier algorithm;
                readonly attribute byte[] publicKey;
        };
```

2.7. attestationData

The data returned by an attestation request

```
  interface attestationData  {
     readonly attribute byte[][] trustChain;
     readonly attribute byte[] validationData;
     readonly attribute DOMString schema;
   };
```

Attributes

readonly byte [] [] trustChain

List of binary data representing the identities of running software. In the TCG scheme, this corresponds to the integrity measurement log.

For example, in a TCG scheme you would expect the following contents:

0x00 -> [0x45ac76fec..., 0x956836fbc42...,]
0x01 -> [0x23c3414f1..., 0xbb3f4d282cf...,]
0x02 -> [0x72bb76045..., 0x04ccc997056...,]
...
0x0c -> [0x8a797441a..., 0xd642ac16d13...,]

This attribute is readonly.

readonly byte [] validationData

single binary blob, containing validation data for the trustChain

In the TCG scheme this would contain the following signed data, including:
- The TPM Quote version (1.1.0.0)
- A fixed byte[4] "QUOT"
- The SHA1 digest of the composite hash of the trustChain
- The 20 byte nonce

This attribute is readonly.

readonly DOMString schema

text string identifying the attestation scheme being used for example, "TPM_Quote"

This attribute is readonly.

2.8. WebinosAttestationInterface

interfaces for attesting the platform

```
  [NoInterfaceObject]
  interface WebinosAttestationInterface {
     attestationData attestPlatform (in byte[] nonce, in SubjectPublicKeyInfo key )
                        raises(AttestationException);
     SubjectPublicKeyInfo getAttestationKey () raises(AttestationException);
     X509 getKeyCredential (in SubjectPublicKeyInfo key) raises(AttestationException);
  };
```

Methods

`attestPlatform`

requests platform to provide attestation data

Signature

```
attestationData attestPlatform(in

                   byte

                [] nonce, in SubjectPublicKeyInfo key);
```

inputs determine freshness and signing key to be used
return values are loose, to support arbitrary schemes of attestation

Note for implementation: there will need to be configuration of attestation modules to support any authentication at the hardware layer.

Error conditions due to policy enforcement, as well as potentially from hardware failure.

Parameters

- **nonce**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** array
  - **Description:** is a 20 byte value used to guarantee freshness of the result
- **key**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** SubjectPublicKeyInfo
  - **Description:** is the (identifier of the) key which should be used for attestation. This will have been retrieved from getAttestationKey().

Return value

data structure returned is described above.

Exceptions

- AttestationException:

`getAttestationKey`

returns public key corresponding to device identity used to authenticate sessions

Signature
```
SubjectPublicKeyInfo getAttestationKey();
```

in TCG terminology this would be the Attestation Identity Key (AIK)

Errors due to permission being denied or if the platform has not implemented attestation due to hardware or software issues.

Return value
returns public key

Exceptions

- AttestationException:

`getKeyCredential`

return certificate for nominated key

Signature
```
X509 getKeyCredential(in SubjectPublicKeyInfo key);
```

If the key identifier is unknown, or the user policy does not allow access to it, an error "not found" is returned.

Parameters

- **key**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** SubjectPublicKeyInfo
  - **Description:** is the RSA public key (or identifier) for which a certificate is to be returned

Return value
returns certificate data structure corresponding to key

Exceptions

- AttestationException:

2.9. WebinosAttestation

The WebinosAttestation interface describes the part of the Attestation API accessible through the webinos object.

```
    [NoInterfaceObject] interface WebinosAttestation {
```

```
                readonly attribute WebinosAttestationInterface attestation;
        };
        webinoscore::Webinos implements WebinosAttestation;
```

Attributes

`readonly WebinosAttestationInterface attestation`

webinos.attestation object.

This attribute is readonly.

3. Type Definitions

3.1. namePairArray

as defined in http://www.ietf.org/rfc/rfc2459.txt

```
        typedef namePair[] namePairArray;
```

4. Exceptions

4.1. AttestationException

Exception handling for the attestation API

```
        exception AttestationException  {
        const unsigned short UNKNOWN_ERROR = 0;
        const unsigned short INVALID_ARGUMENT_ERROR = 1;
        const unsigned short IO_ERROR = 4;
        const unsigned short NOT_SUPPORTED_ERROR = 5;
        const unsigned short PERMISSION_DENIED_ERROR = 20;
        const unsigned short KEY_NOT_FOUND_ERROR = 21;
        unsigned short code;
        DOMString message;
        };
```

Field

`unsigned short code`

An error code assigned by an implementation when an error has occurred in attestation API processing.

`DOMString message`

5. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/attestation

6. Full WebIDL

```
module attestation {
```

```
interface X509 {
        readonly attribute TBSCertificate certificate;
        readonly attribute AlgorithmIdentifier signatureAlgorithm;
        readonly attribute byte[] signature;
};

interface TBSCertificate {
        readonly attribute DOMString version;
        readonly attribute Integer serialNumber;
        readonly attribute AlgorithmIdentifier signature;
        readonly attribute namePairArray issuer;
        readonly attribute Validity validity;
        readonly attribute namePairArray subject;
        readonly attribute SubjectPublicKeyInfo subjectPublicKeyInfo;
        readonly attribute Any? extensions;
};

interface Validity {
        readonly attribute Date notBefore;
        readonly attribute Date notAfter;
};

typedef namePair[] namePairArray;

interface namePair {
        readonly attribute DOMString key;
        readonly attribute DOMString value;
};

interface AlgorithmIdentifier {
    readonly attribute DOMString identifier;
        readonly attribute DOMString? parameters;
};

interface SubjectPublicKeyInfo {
        readonly attribute AlgorithmIdentifier algorithm;
        readonly attribute byte[] publickKey;
};


exception AttestationException  {
const unsigned short UNKNOWN_ERROR = 0;
const unsigned short INVALID_ARGUMENT_ERROR = 1;
const unsigned short IO_ERROR = 4;
const unsigned short NOT_SUPPORTED_ERROR = 5;
const unsigned short PERMISSION_DENIED_ERROR = 20;
const unsigned short KEY_NOT_FOUND_ERROR = 21;
unsigned short code;
DOMString message;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
  interface attestationData  {
     readonly attribute byte[][] trustChain;
     readonly attribute byte[] validationData;
     readonly attribute DOMString schema;
   };


  [NoInterfaceObject]
  interface WebinosAttestationInterface {
     attestationData attestPlatform (in byte[] nonce, in SubjectPublicKeyInfo key )
                        raises(AttestationException);
     SubjectPublicKeyInfo getAttestationKey () raises(AttestationException);
     X509 getKeyCredential (in SubjectPublicKeyInfo key) raises(AttestationException);
  };
        [NoInterfaceObject] interface WebinosAttestation {
                readonly attribute WebinosAttestationInterface attestation;
        };
        webinoscore::Webinos implements WebinosAttestation;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The authentication module

Webinos API Specifications

30 Jun 2011

Authors

- John Lyle; <john.lyle@cs.ox.ac.uk>

Abstract

Provides information to applications about the current authentication status of users, as well as allowing applications to request re-authentication.

Summary of Methods

Interface	Method
AuthStatus	
AuthError	
AuthSuccessCB	void onSuccess(AuthStatus status)
AuthErrorCB	void onError(AuthError error)
WebinosAuthenticationInterface	void authenticate(AuthSuccessCB successCB, AuthErrorCB errorCB) boolean isAuthenticated() AuthStatus getAuthenticationStatus()
WebinosAuthentication	

1. Introduction

Authentication API for providing applications with information about whether the current user has authenticated, and requesting re-authentication at runtime.

Requirement/architectural reference: PS-USR-Oxford-121

This API intentionally does not reveal identity information, but can give details about authentication method, which may reveal device information.

2. Interfaces

2.1. AuthStatus

The object returned when user authentication status is queried. This contains information about when and how authentication occurred.

```
[NoInterfaceObject]
interface AuthStatus  {
       attribute DOMString? lastAuthTime;
       attribute DOMString? authMethod;
       attribute DOMString? authMethodDetails;
};
```

Attributes

`DOMString? lastAuthTime`

The time of last authentication, as a valid date or time string.

No exceptions.

Code example

```
 {lastAuthTime: '2011-03-24T09:00-08:00'} // last authentication was on
March 24, 2011 @ 5pm (UTC)
```

`DOMString? authMethod`

An identifier for the type of authentication performed by the user.
Intended to be flexible for different devices. Examples include "PIN", "Password", "Fingerprint". This is a high-level method name, no details.

`DOMString? authMethodDetails`

Further details as to the authentication method. This might include the authentication device identifier, or the numberof digits in PINS, or any device-specific value. Optional.

2.2. AuthError

Definition of error codes for authentication events

```
[NoInterfaceObject]
interface AuthError  {
      const unsigned short UNKNOWN_ERROR = 0;
      const unsigned short INVALID_ARGUMENT_ERROR = 1;
      const unsigned short PERMISSION_DENIED_ERROR = 20;
      const unsigned short TIMEOUT_ERROR = 2;
      readonly attribute unsigned short code;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Constants

`unsigned short UNKNOWN_ERROR`

An unknown error occurred.

`unsigned short INVALID_ARGUMENT_ERROR`

An invalid parameter was provided when the requested method was invoked.

`unsigned short PERMISSION_DENIED_ERROR`

Access to the requested method was denied at the implementation or by the user.

`unsigned short TIMEOUT_ERROR`

Authentication timed out

Attributes

`readonly unsigned short code`

An error code assigned by an implementation when an error has occurred in authentication API processing.

This attribute is readonly.

2.3. AuthSuccessCB

Success callback for authentication events.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface AuthSuccessCB {
      void onSuccess(AuthStatus status);
};
```

2.4. AuthErrorCB

Error callback for authentication events.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface AuthErrorCB {
      void onError(AuthError error);
};
```

2.5. WebinosAuthenticationInterface

The authentication interface provides three methods which allow applications to check the current user authentication status and prompt for re-authentication.

```
  [NoInterfaceObject]
  interface WebinosAuthenticationInterface {
     void authenticate ( in AuthSuccessCB successCB, in optional AuthErrorCB errorCB
);
     boolean isAuthenticated ( ) raises(AuthenticationException);
     AuthStatus getAuthenticationStatus () raises(AuthenticationException);
   };
```

Methods

`authenticate`

this method instructs the runtime to request that the user authenticate themselves.
The method for authentication is not specified, it may be through any means provided by the platform.

Signature

```
void authenticate(in AuthSuccessCB successCB, in optional AuthErrorCB
errorCB);
```

Errors can occur due to: a policy restricting access to this API, or an unknown error in the device-specific authentication method.

This is an asynchronous method, although it may well be used (in practice) synchronously.

Parameters

- **successCB**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** AuthSuccessCB
  - **Description:** contains the status of the user with regards to authentication, including when
    and how he or she was last authenticated. It does not include user identity.
- **errorCB**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** AuthErrorCB
  - **Description:** is a callback for when errors occur

Return value

`isAuthenticated`

Query the runtime to ask whether the user has recently been authenticated. How the platform
determines this is not specified. It may return true if the user entered their PIN in the last 10
minutes, for example. It is expected that a platform preference based on current authentication
status would be defined. These preferences are security-sensitive.

Signature

```
boolean isAuthenticated();
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Errors can occur due to: a policy restricting access to this API, or if the platform does not have
a definitive answer due to a misconfigured preference or lack of information.

This is a synchronous method. Expected use would be to check at an important place whether the user is authenticated and, if not, call "authenticate" to do so.

Return value

True IFF the user has been authenticated to the satisfaction of the platform.

Exceptions

- AuthenticationException:

`getAuthenticationStatus`

Query the runtime for precise details about the current state of the user with regard to authentication.

Signature

```
AuthStatus getAuthenticationStatus();
```

Errors can occur due to: a policy restricting access to this API.

This is a synchronous method. Expected use is for when an application is determining whether the user ought to re-authenticate, or whether the user is suitably authenticated for a particular action. Future versions of this API may be able to insist that the user authenticates in a certain way.

Return value

AuthStatus - returns the status of the user with regards to authentication, including when
and how he or she was last authenticated. It does not include user identity.

Exceptions

- AuthenticationException:

2.6. WebinosAuthentication

The WebinosAuthentication interface describes the part of the Authentication API accessible through the webinos object.

```
[NoInterfaceObject] interface WebinosAuthentication {
        readonly attribute WebinosAuthenticationInterface authentication;
};
webinoscore::Webinos implements WebinosAuthentication;
```

Attributes

`readonly WebinosAuthenticationInterface authentication`

webinos.authentication object.

This attribute is readonly.

3. Exceptions

3.1. AuthenticationException

Exception codes for authentication events

```
exception AuthenticationException{
      unsigned short code;
      DOMString message;
      const unsigned short UNKNOWN_ERROR = 0;
      const unsigned short INVALID_ARGUMENT_ERROR = 1;
      const unsigned short PERMISSION_DENIED_ERROR = 20;
      const unsigned short TIMEOUT_ERROR = 2;
};
```

4. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/authentication

5. Full WebIDL

```
module authentication {
  [NoInterfaceObject]
  interface AuthStatus  {
         attribute DOMString? lastAuthTime;
         attribute DOMString? authMethod;
         attribute DOMString? authMethodDetails;
  };

  [NoInterfaceObject]
  interface AuthError  {
        const unsigned short UNKNOWN_ERROR = 0;
        const unsigned short INVALID_ARGUMENT_ERROR = 1;
        const unsigned short PERMISSION_DENIED_ERROR = 20;
        const unsigned short TIMEOUT_ERROR = 2;
        readonly attribute unsigned short code;
  };

  [Callback=FunctionOnly, NoInterfaceObject]
  interface AuthSuccessCB {
        void onSuccess(AuthStatus status);
  };

  [Callback=FunctionOnly, NoInterfaceObject]
  interface AuthErrorCB {
        void onError(AuthError error);
  };

  exception AuthenticationException{
        unsigned short code;
        DOMString message;
        const unsigned short UNKNOWN_ERROR = 0;
        const unsigned short INVALID_ARGUMENT_ERROR = 1;
        const unsigned short PERMISSION_DENIED_ERROR = 20;
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
        const unsigned short TIMEOUT_ERROR = 2;
  };
 [NoInterfaceObject]
  interface WebinosAuthenticationInterface {
     void authenticate ( in AuthSuccessCB successCB, in optional AuthErrorCB errorCB
);
     boolean isAuthenticated ( ) raises(AuthenticationException);
     AuthStatus getAuthenticationStatus () raises(AuthenticationException);
   };
        [NoInterfaceObject] interface WebinosAuthentication {
                readonly attribute WebinosAuthenticationInterface authentication;
        };

        webinoscore::Webinos implements WebinosAuthentication;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The context module

Webinos API Specifications

1 Jul 2011

Authors

- Heiko Desruelle <heiko.desruelle@intec.ugent.be>
- Dieter Blomme <dieter.blomme@intec.ugent.be>
- George Gionis <gionis@epu.ntua.gr>

Abstract

The Context API

Summary of Methods

Interface	Method
ContextManagerHook	
ContextManager	void executeQuery(QuerySuccessCallback successCallback, QueryErrorCallback? errorCallback, Query query) void subscribeContextEvent(SubscribeSuccessCallback subscribeSuccessCallback, SubscribeErrorCallback? subscribeErrorCallback, OccuringEvent eventHandler, DOMString eventIdentifier) void unsubscribe(unsigned long subscriptionIdentifier)
Query	
ContextError	
QuerySuccessCallback	void onsuccess(SPARQLquery queryResult)
QueryErrorCallback	void onerror(ContextError error)

Interface	Method
SubscribeSuccessCallback	void onsuccess(unsigned long subscriptionIdentifier)
SubscribeErrorCallback	void onerror(ContextError error)
OccuringEvent	void onContextEvent(DOMString eventContextData)

1. Introduction

The Context API defines the high-level interfaces required to obtain access to a user's context data. The API supports two basic ways of accessing context data:

1. executing a query against the context data storage and retrieving context data through the query results.

2. subscribing to receive real time context data updates as soon as a context related event happens.

2. Interfaces

2.1. ContextManagerHook

Defines what is instantiated in initialization.

```
[NoInterfaceObject] interface ContextManagerHook {
  readonly attribute ContextManager context;
};
```

There will be a webinos.context object that allows accessing the * functionality of this module.

2.2. ContextManager

This is the entry point for the context API. The interface provides the two basic methods to access the User's context data, i.e. query or subscribe to updates.

```
[NoInterfaceObject] interface ContextManager {

  void executeQuery(in QuerySuccessCallback successCallback,
                    in QueryErrorCallback? errorCallback,
                    in Query query)
       raises(ContextError);

  void subscribeContextEvent(in SubscribeSuccessCallback subscribeSuccessCallback,
                    in SubscribeErrorCallback? subscribeErrorCallback,
                    in OccuringEvent eventHandler,
                                        in DOMString eventIdentifier)
       raises(ContextError);

  void unsubscribe(in unsigned long subscriptionIdentifier)
       raises(ContextError);
};
webinos implements ContextManager;
```

Methods

`executeQuery`

Performs a context query against the context storage.

Signature

```
void    executeQuery(in    QuerySuccessCallback    successCallback,    in
QueryErrorCallback? errorCallback, in Query query);
```

When this method is invoked it executes the provided query against the context storage. The context storage is a collection context objects, each one with specific attributes, which hold context data that have been acquired over time by identifying a number of context related events. The Query parameter that this method uses specifies what context data (i.e. from which context objects) should be retrieved.

Mediation by policy and security: this method, as it provides application with data (context) about the user, is expected to have privacy considerations. Therefore the system is able to ignore the request of an app to receive context data if the User Privacy Policy dictates so (i.e. the user has not authorized the app to access the context data it ask for in the Query parameter).

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** QuerySuccessCallback
  - **Description:** Function to be invoked if the asynchronous query operation completes successfully.
- **errorCallback**
  - **Optional:** No.
  - **Nullable**: Yes
  - **Type:** QueryErrorCallback
  - **Description:** Function to be invoked if the asynchronous query operation results in errors.
- **query**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Query

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

- **Description:** The Query object describing the query to be executed against the context storage.

Return value

void call

Exceptions

- ContextError:

  with error code TYPE_MISMATCH_ERR if the input parameter is not compatible with the expected type for that parameter.

`subscribeContextEvent`

Registers the function to be notifies when a context related event occurs.

Signature

```
void subscribeContextEvent(in SubscribeSuccessCallback
subscribeSuccessCallback, in SubscribeErrorCallback?
subscribeErrorCallback, in OccuringEvent eventHandler, in DOMString
eventIdentifier);
```

When this method is invoked, the implementation must register the (app) function that is passed in the eventHandler argument as the hander function that will be notified when the context related event, which in turn is identified in the eventIdentifier, happens. This function will be invoked every time the indicated event occurs until the unsubscribe method is invoked to cancel the subscription.

If the subscription is successfully created, an identifier for the handler is created and returned in subscriptionSuccessCallback so that it is possible to cancel the subscription. If the subscription cannot be created, the subscriptionErrorCallback contains an error code that describes the reason for the error.

Mediation by policy and security: this method, as it can provide an application with real-time data (context) about the user, is expected to have privacy considerations. Therefore the system is able to ignore the request of an app to receive context data if the User Privacy Policy dictates so (i.e. the user has not authorized the app to access the context data of the dictated event).

Parameters

- **subscribeSuccessCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** SubscribeSuccessCallback

  - **Description:** Function to be invoked if the asynchronous subscribe request completes successfully.

- **subscribeErrorCallback**
  - **Optional:** No.
  - **Nullable**: Yes
  - **Type:** SubscribeErrorCallback
  - **Description:** Function to be invoked if the asynchronous subscribe request results in errors.
- **eventHandler**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** OccuringEvent
  - **Description:** The function to be invoked when the dictated event occurs.
- **eventIdentifier**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** The event identifier, for example "UserProfileUpdate", "ShoppingBasketCheckout".

Return value

void call

Exceptions

- ContextError:

  with error code TYPE_MISMATCH_ERR if the input parameter is not compatible with the expected type for that parameter.

`unsubscribe`

Cancels a subscription to a context related event.

Signature

```
void unsubscribe(in unsigned long subscriptionIdentifier);
```

If the subscriptionIdentifier argument is valid and corresponds to a subscription already in place the subscription process MUST be effectivelly stoped. If the subscriptionHandler argument does not correspond to a valid subscription, the method should return without any further action.

Parameters

- **subscriptionIdentifier**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** unsigned long
  - **Description:** The identifier of the subscription, returned by subscribeContextEvent().

Return value

void call

Exceptions

- ContextError:

  with error code TYPE_MISMATCH_ERR if the input parameter is not compatible with the expected type for that parameter.

2.3. Query

Query interface,

```
   [Callback, NoInterfaceObject] interface Query {
    attribute SPARQLquery xmlQuery;
  };
```

Attributes

`SPARQLquery xmlQuery`

query content in xml format

SPARQL query language for RDF

2.4. ContextError

Defines the error codes for this module

```
  [NoInterfaceObject] interface ContextError {
     const unsigned short SECURITY_ERR = 1;
     const unsigned short INVALID_QUERY_ERR = 2;
     const unsigned short TYPE_MISMATCH_ERR = 3;
   };
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Constants

`unsigned short SECURITY_ERR`

Security Error

`unsigned short INVALID_QUERY_ERR`

invalid query

`unsigned short TYPE_MISMATCH_ERR`

invalid query

2.5. QuerySuccessCallback

Interface for callbacks indicating success of executeQuery() operation.

```
[Callback=FunctionOnly, NoInterfaceObject] interface QuerySuccessCallback {
  void onsuccess (SPARQLquery queryResult);
};
```

Methods

`onsuccess`

Callback on success of a executeQuery() operation

Signature

```
void onsuccess(SPARQLquery queryResult);
```

Parameters

- **queryResult**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** SPARQLquery
  - **Description:** Result of the query operation serialized as a json string, see SPARQL query language for RDF.

Return value

void

2.6. QueryErrorCallback

Interface for callbacks indicating error of executeQuery() operation.

```
[Callback=FunctionOnly, NoInterfaceObject] interface QueryErrorCallback  {
  void onerror (ContextError error);
};
```

Methods

`onerror`

Callback on failure of a executeQuery() operation

Signature

```
void onerror(ContextError error);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ContextError
  - **Description:** The ContextError object capturing the type of the error.

Return value

void

2.7. SubscribeSuccessCallback

Interface for callbacks indicating success of subscribeContextEvent() operation.

```
[Callback=FunctionOnly, NoInterfaceObject] interface SubscribeSuccessCallback {
  void onsuccess (unsigned long subscriptionIdentifier);
};
```

Methods

onsuccess

Callback on success of a subscribeContextEvent() operation

Signature

```
void onsuccess(unsigned long subscriptionIdentifier);
```

Parameters

- **subscriptionIdentifier**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** unsigned long
  - **Description:** A subscription handler that can be later used to cancel the subscription.

Return value

void

2.8. SubscribeErrorCallback

Interface for callbacks indicating error of subscribeContextEvent() operation.

```
[Callback=FunctionOnly, NoInterfaceObject] interface SubscribeErrorCallback  {
  void onerror (ContextError error);
};
```

Methods

onerror

Callback on failure of a subscribeContextEvent() operation

Signature

```
void onerror(ContextError error);
```

Parameters

- **error**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** ContextError
    - **Description:** The ContextError object capturing the type of the error.

Return value

void

2.9. OccuringEvent

Interface for specifying the method called when a new context related event occurs.

```
[Callback=FunctionOnly, NoInterfaceObject] interface OccuringEvent {
  void onContextEvent (DOMString eventContextData);
};
```

This interface specifies a function that provides a serialized json string of the context data retrieved from the happening of a context related event.

Methods

onContextEvent

Method invoked when the context related event occurs.

Signature

```
void onContextEvent(DOMString eventContextData);
```

Parameters

- **eventContextData**
    - **Optional:** No.

- **Nullable**: No
- **Type:** DOMString
- **Description:** The context data acquired by the event.

Return value

void

3. Features

When the feature

- `http://webinos.org/api/context`

or any of the features hierarchically under that feature is successfully requested, the interface `ContextManager` is instantiated, and the resulting object appears in the global namespace as `.context`.

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/context

Acccess to all the module. This feature provides access to the whole API. Security and Privacy enforcement may depend on the query or subscription requested by the developer.

4. Full WebIDL

```
module context {
  [NoInterfaceObject] interface ContextManagerHook {
    readonly attribute ContextManager context;
  };
  webinos implements ContextManager;

  [NoInterfaceObject] interface ContextManager {
    void executeQuery(in QuerySuccessCallback successCallback,
                      in QueryErrorCallback? errorCallback,
                      in Query query)
         raises(ContextError);

    void subscribeContextEvent(in SubscribeSuccessCallback subscribeSuccessCallback,
                      in SubscribeErrorCallback? subscribeErrorCallback,
                      in OccuringEvent eventHandler,
                                          in DOMString eventIdentifier)
         raises(ContextError);

    void unsubscribe(in unsigned long subscriptionIdentifier)
         raises(ContextError);
  };

   [Callback, NoInterfaceObject] interface Query {
    attribute SPARQLquery xmlQuery;
  };

  [NoInterfaceObject] interface ContextError {
     const unsigned short SECURITY_ERR = 1;
     const unsigned short INVALID_QUERY_ERR = 2;
     const unsigned short TYPE_MISMATCH_ERR = 3;
   };

  [Callback=FunctionOnly, NoInterfaceObject] interface QuerySuccessCallback {
    void onsuccess (SPARQLquery queryResult);
  };

  [Callback=FunctionOnly, NoInterfaceObject] interface QueryErrorCallback  {
    void onerror (ContextError error);
  };

  [Callback=FunctionOnly, NoInterfaceObject] interface SubscribeSuccessCallback {
    void onsuccess (unsigned long subscriptionIdentifier);
  };

  [Callback=FunctionOnly, NoInterfaceObject] interface SubscribeErrorCallback  {
    void onerror (ContextError error);
  };

  [Callback=FunctionOnly, NoInterfaceObject] interface OccuringEvent {
    void onContextEvent (DOMString eventContextData);
  };
};
```

APIs: The events module

Webinos API Specifications

30 Jun 2011

Authors

- Stefano D'Angelo <dangelo@ismb.it>

Abstract

The Event Handling API

Summary of Methods

Interface	Method
WebinosEventEntity	
WebinosEventAddressing	
WebinosEvent	void dispatchWebinosEvent(WebinosEventCallbacks? callbacks, DOMTimeStamp? referenceTimeout, boolean sync) void forwardWebinosEvent(WebinosEventAddressing forwarding, boolean withTimeStamp, WebinosEventCallbacks? callbacks, DOMTimeStamp? referenceTimeout, boolean sync)
WebinosEventDeliveryError	
WebinosEventCallbacks	void onSending(WebinosEvent event, WebinosEventEntity recipient) void onCaching(WebinosEvent event) void onDelivery(WebinosEvent event, WebinosEventEntity recipient) void onTimeout(WebinosEvent event, WebinosEventEntity recipient) void onError(WebinosEvent event, WebinosEventEntity recipient, WebinosEventDeliveryError error)

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Interface	Method
WebinosEventListener	void handleEvent(WebinosEvent event)
WebinosEventsInterface	WebinosEvent createWebinosEvent(DOMString type, WebinosEventAddressing addressing, DOMString? payload, WebinosEvent? inResponseTo, boolean withTimeStamp, DOMTimeStamp? expiryTimeStamp, boolean addressingSensitive) DOMString addWebinosEventListener(WebinosEventListener listener, DOMString? type, WebinosEventEntity? source, WebinosEventEntity? destination) void removeWebinosEventListener(DOMString listenerId)
WebinosEvents	

1. Introduction

The Webinos Event Handling API provides means to exchange data in terms of events among addressable entities (e.g., applications, services), either locally or remotely.

This is an advanced API that is mostly meant to be used by third-party developers to implement custom event-based protocols by taking advantage of the features offered by the Webinos event handling system, that in turn leverages off of the features offered by Webinos overlay networking model.

It is, therefore, strongly recommended to carefully read the Messaging section in the Webinos system specifications before committing to the usage of this API.

Despite its apparent complexity, this API revolves around three simple basic concepts: generating events, sending/forwarding events and registering/unregistering event listeners for incoming events.

The following example demonstrates what the core functionality of a simplicistic textual chat application could look like if implemented using this API.

Code example

```
// Array of objects implementing the WebinosEventEntity interface that
// represents the list of participants to the chat session, excluding the
// current application.
var participants = [...];

// DOM element that keeps a log of the whole session.
var logElem;

// DOM Text object that allows the user to insert text messages.
var inputElem;

// Function that somehow returns a human-readable name associated to the input
// entity.
function getName(entity) {
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
  ...
}

// Function that returns a copy of the input string with HTML control characters
// ('<', '>', '&') escaped ('&lt;', '&gt;', '&amp;').
function escapeHTML(str) {
  str = str.replace(/&/g, '&amp;');
  return str.replace(/</g, '&lt;').replace(/>/g, '&gt;');
}

// Listener callback for incoming events.
function onMsg(evt) {
  // Appends event data to logElem.
  // E.g.: [00:00:00] Stefano said: Hi all!
  logElem.innerHTML += "[" + new Date(evt.timeStamp).toLocaleTimeString() + "] "
                     + getName(evt.addressing.source) + " said: "
                     + escapeHTML(evt.payload) + "\n";
}

// Delivery error notification callback.
function onMsgError(evt, recipient, error)
{
  // Pops up an alert dialog with error details.
  // E.g.: Stefano did not receive your message saying: "How are you?"
  //       Event refused (4)
  alert(getName(recipient) + ' did not receive your message saying: "'
        + evt.payload + '"\n' + error.message + '(' + error.code + ')');
}

// Listener to DOM "click" event for some "Send Message" button.
function onSendButtonClicked() {
  // Creates a new event of type "chatMessage" directed to all the chat
  // participants with payload containing the text in inputElem and with
  // timestamp.
  var evt = webinos.events.createWebinosEvent("chatMessage", {to: participants},
                                              inputElem.value, null, true, null,
                                              true);

  // Sends the event and specifies the onMsgError callback for handling error
  // delivery notifications.
  evt.dispatchWebinosEvent({onError: onMsgError});

  // Appends the input message to logElem.
  // E.g.: [00:01:00] you said: "Let's try again... how are you?"
  logElem.innerHTML += "[" + new Date(evt.timeStamp).toLocaleTimeString()
                     + "] you said: " + escapeHTML(evt.payload) + "\n";
}

// Initialization stuff.
document.onLoad = function() {
  // Gets "log" element.
  logElem = document.getElementById("log");

  // Gets "input" element.
  inputElem = document.getElementById("input");

  // Adds the onMsg() callback as an event listener for incoming events with
  // type "chatMessage", from any source and to any destination (within the
  // application).
  webinos.events.addWebinosEventListener(onMsg, "chatMessage");
}
```

2. Interfaces

2.1. WebinosEventEntity

The WebinosEventEntity interface describes an addressable entity of any kind.

```
        [NoInterfaceObject] interface WebinosEventEntity {
                attribute DOMString id;
        };
```

Attributes

`DOMString id`

Globally unique identifier.

2.2. WebinosEventAddressing

The WebinosEventAddressing interface contains references to the sender and recipients of an event.

```
        [NoInterfaceObject] interface WebinosEventAddressing {
                attribute WebinosEventEntity source;
                attribute WebinosEventEntity[] to;
                attribute WebinosEventEntity[] cc;
                attribute WebinosEventEntity[] bcc;
        };
```

This interfaces comes in two flavors: a strict normalized form for events generated and/or processed by the Webinos runtime, and a more lax non-normalized form to ease API usage.

Please, keep in mind that the Webinos runtime always operates on normalized equivalents of user-supplied objects implementing this interface in non-normalized form, hence it creates those equivalents by applying the normalization process described in the documentation of the createWebinosEvent() function of the WebinosEventsInterface interface.

Details on both forms are given in each attribute's description.

Attributes

`WebinosEventEntity source`

Event source.

In the normalized form it SHALL always be set.

The non-normalized form allows to use null or undefined to indicate the current application.

`WebinosEventEntity [] to`

Array of primary recipients.

It MUST always contain at least one element.

In the normalized form it SHALL be sorted in ascending order by Unicode code points and SHALL NOT contain duplicate entries.

`WebinosEventEntity [] cc`

Array of secondary recipients.

It MAY be empty.

In the normalized form it SHALL be sorted in ascending order by Unicode code points, SHALL NOT contain duplicate entries and SHALL NOT contain entries that are also found in the "to" array.

The non-normalized form allows to use null or undefined to indicate no secondary recipients.

`WebinosEventEntity [] bcc`

Array of blind-carbon-copy recipients.

It MAY be empty.

In the normalized form it SHALL be sorted in ascending order by Unicode code points, SHALL NOT contain duplicate entries and SHALL NOT contain entries that are also found in the "to" or "cc" arrays.

The non-normalized form allows to use null or undefined to indicate no blind-carbon-copy recipients.

2.3. WebinosEvent

The WebinosEvent interface describes an incoming or outgoing event.

```
[NoInterfaceObject] interface WebinosEvent {
        readonly attribute DOMString type;
        readonly attribute WebinosEventAddressing addressing;
        readonly attribute DOMString id;
        readonly attribute WebinosEvent inResponseTo;
        readonly attribute DOMTimeStamp? timeStamp;
        readonly attribute DOMTimeStamp? expiryTimeStamp;
        readonly attribute boolean addressingSensitive;
        readonly attribute WebinosEventAddressing forwarding;
        readonly attribute DOMTimeStamp? forwardingTimeStamp;
        readonly attribute DOMString? payload;

        void dispatchWebinosEvent(
                        in optional WebinosEventCallbacks? callbacks,
                        in optional DOMTimeStamp? referenceTimeout,
                        in optional boolean sync)
             raises(WebinosEventException);
        void forwardWebinosEvent(
                        in WebinosEventAddressing forwarding,
                        in optional boolean withTimeStamp,
                        in optional WebinosEventCallbacks? callbacks,
                                    in optional DOMTimeStamp? referenceTimeout,
                                    in optional boolean sync)
                        raises(WebinosEventException);
         };
```

Attributes

`readonly DOMString type`

Event type identifier.

It MUST match the following regular expression: [_a-zA-Z][_a-zA-Z0-9]*

Identifiers "deliveryNotification", "JSONRPC20Request" and "JSONRPC20Response" are reserved, hence not allowed.

This attribute is readonly.

`readonly WebinosEventAddressing addressing`

References to the original sender and recipients in normalized form.

This attribute is readonly.

`readonly DOMString id`

Event identifier.

It is calculated by hashing a partial serialization of the WebinosEvent object that involves:
- the event type;
- the original event source and primary recipients, in case the "addressingSensitive" attribute is true;
- the identifier of the event that this event is a response to, if any;
- the event timestamp, if present;
- the event expiry timestamp, if present;
- the payload, if present.

Even though not strictly required, conforming implementations are recommended to implement some strategy to try to limit the likelihood that an application instance is delivered more than one event with a given id. Users of this API MUST assume that no more than one event with a given id is ever delivered to a given recipient.

For more details, please refer to the Webinos system specifications.

This attribute is readonly.

`readonly WebinosEvent inResponseTo`

Event that this event is a response to.

If null, this event was not sent in response to another event.

This attribute is readonly.

`readonly DOMTimeStamp? timeStamp`

Moment in time in which the event is generated by the original event source.

It MAY be null.

This attribute is readonly.

`readonly DOMTimeStamp? expiryTimeStamp`

Moment in time past which the event is no more valid or meaningful.

It MAY be null.

This attribute is readonly.

`readonly boolean addressingSensitive`

Indicates whether the original addressing information is part of the informative content of the event.

In practice, when this is set to true, the identifiers of the orignal event source and primary recipients are used to compute the event id.

This attribute is readonly.

`readonly WebinosEventAddressing forwarding`

References to the entity that forwarded the event and the recipients of such forwarding.

It is null if the event was not subject to any forwarding, hence it comes straight from the original sending entity.

This attribute is readonly.

`readonly DOMTimeStamp? forwardingTimeStamp`

Moment in time in which the event was forwarded by the forwarding source.

It SHALL be null if "forwarding" is null and MAY be null also if "forwarding" is not null.

This attribute is readonly.

`readonly DOMString? payload`

Event type-specific data.

It MAY be null.

This attribute is readonly.

Methods

`dispatchWebinosEvent`

Sends an event.

Signature

```
void dispatchWebinosEvent(in optional WebinosEventCallbacks? callbacks,
in optional DOMTimeStamp? referenceTimeout, in optional boolean sync);
```

W.r.t. the "Delivery notification wanted" attribute described in the Webinos system specification, its value in the implementation is to be deferred from the callbacks parameter and allowed to change between dispatchWebinosEvent() and forwardWebinosEvent() calls. It SHALL be true when delivery and/or error callbacks are passed via the callbacks argument, false otherwise.

NOTE: Please, keep in mind that all recipients SHALL get references to all entities specified in the "to" and "cc" attributes.

Parameters

- **callbacks**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** WebinosEventCallbacks
  - **Description:** Set of callbacks to monitor sending status (null and undefined are considered as equivalent to a WebinosEventCallbacks object with all attributes set to null).
- **referenceTimeout**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** DOMTimeStamp
  - **Description:** Moment in time until which the Webinos runtime SHALL ensure that the WebinosEvent object being sent is not garbage collected for the purpose of receiving events in response to the event being sent (null, undefined and values up to the current date/time mean that no special action is taken by the runtime in this regard).
- **sync**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** If false or undefined, the function is non-blocking, otherwise if true it will block until one of the following conditions becomes true:
    - if referenceTimeout represents a moment in time in the future at call time, that moment is reached;
    - otherwise, if the "expiryTimeStamp" attribute is specified as a moment in time

in the future at call time, that moment is reached;
- in any case, the end result of the operation is completely determined for all recipients and all callbacks that were to be called have run.

Exceptions

- WebinosEventException:

  INVALID_ARGUMENT_ERROR if any of the supplied arguments is not valid.
  PERMISSION_DENIED_ERROR if some local policy rule does not allow for the event to be sent.

`forwardWebinosEvent`

Forwards an event.

Signature

```
void forwardWebinosEvent(in WebinosEventAddressing forwarding, in optional boolean withTimeStamp, in optional WebinosEventCallbacks? callbacks, in optional DOMTimeStamp? referenceTimeout, in optional boolean sync);
```

W.r.t. the "Delivery notification wanted" attribute described in the Webinos system specification, its value in the implementation is to be deferred from the callbacks parameter and allowed to change between dispatchWebinosEvent() and forwardWebinosEvent() calls. It SHALL be true when delivery and/or error callbacks are passed via the callbacks argument, false otherwise.

Conforming implementations SHALL NOT modify the "Forwarding" attribute of the local WebinosEvent object when this function is called.

NOTE: Please, keep in mind that all recipients referenced by the forwarding argument SHALL get references to all entities specified in the "to", "cc" and "bcc" arrays of the "addressing" attribute, as well as references to all entities specified in the "to" and "cc" arrays of the forwarding argument.

Parameters

- **forwarding**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEventAddressing
  - **Description:** References to the entity on the behalf of which the application wants to forward the event and to the recipients of such forwarding. This argument SHALL be processed in the same way as the webinos.events.createWebinosEvent() function processes its "addressing" argument (i.e., make a normalized copy).
- **withTimeStamp**

- **Optional:** Yes.
- **Nullable**: No
- **Type:** boolean
- **Description:** Whether to set the forwarding timestamp (undefined is considered as equivalent to false).

- **callbacks**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** WebinosEventCallbacks
  - **Description:** Set of callbacks to monitor forwarding status (null and undefined are considered as equivalent to an WebinosEventCallbacks object with all attributes set to null).
- **referenceTimeout**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** DOMTimeStamp
  - **Description:** Moment in time until which the Webinos runtime SHALL ensure that the WebinosEvent object being forwarded is not garbage collected for the purpose of receiving events in response to the event being forwarded (null, undefined and values up to the current date/time mean that no special action is taken by the runtime in this regard).
- **sync**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** If false or undefined, the function is non-blocking, otherwise if true it will block until one of the following conditions becomes true:
    - if referenceTimeout represents a moment in time in the future at call time, that moment is reached;
    - otherwise, if the "expiryTimeStamp" attribute is specified as a moment in time in the future at call time, that moment is reached;
    - in any case, the end result of the operation is completely determined for all recipients and all callbacks that were to be called have run.

Exceptions

- WebinosEventException:

INVALID_ARGUMENT_ERROR if any of the supplied arguments is not valid.
PERMISSION_DENIED_ERROR if some local policy rule does not allow for the event to be forwarded.

2.4. WebinosEventDeliveryError

The WebinosEventDeliveryError interface describes event delivery errors reported using the delivery notification protocol.

```
[NoInterfaceObject] interface WebinosEventDeliveryError {
        readonly attribute unsigned short code;
        readonly attribute DOMString message;

        const unsigned short UNKNOWN_ERR                = 0;
        const unsigned short INVALID                    = 1;
        const unsigned short BAD_DESTINATION            = 2;
        const unsigned short EXPIRED                    = 3;
        const unsigned short REFUSED                    = 4;
        const unsigned short NO_REFERENCE               = 5;
};
```

Constants

`unsigned short UNKNOWN_ERR`

Unknown error.

`unsigned short INVALID`

The recipient got an invalid event (e.g., transmission error).

`unsigned short BAD_DESTINATION`

The intended recipient is unknown or unreachable.

`unsigned short EXPIRED`

The event expired before the actual delivery, according to its "expiryTimestamp" attribute.

`unsigned short REFUSED`

The event could not be received because of lack of authorization and/or policy settings.

`unsigned short NO_REFERENCE`

The recipient does not hold a local reference to the event specified by the "inResponseTo" attribute.

Attributes

`readonly unsigned short code`

Error code.

This attribute is readonly.

`readonly DOMString message`

Error description.

This attribute is readonly.

2.5. WebinosEventCallbacks

The WebinosEventCallbacks interface allows to pass a set of status monitoring callbacks to event sending/forwarding methods.

```
[Callback, NoInterfaceObject] interface WebinosEventCallbacks {
        void onSending(in WebinosEvent event,
                       in WebinosEventEntity recipient);
        void onCaching(in WebinosEvent event);
        void onDelivery(in WebinosEvent event,
                        in WebinosEventEntity recipient);
        void onTimeout(in WebinosEvent event,
                       in WebinosEventEntity recipient);
        void onError(in WebinosEvent event,
                     in WebinosEventEntity recipient,
                     in WebinosEventDeliveryError error);
};
```

TODO: does this definition allow to use null/undefined? Otherwise should define callback types and use nullable attributes.

Methods

`onSending`

Called right after the event has been successfully transmitted to the "next hop".

Signature

```
void onSending(in WebinosEvent event, in WebinosEventEntity recipient);
```

This callback SHALL NOT be called more than once per recipient for each send/forward.

Parameters

- **event**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEvent
  - **Description:** The event being transmitted.
- **recipient**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEventEntity
  - **Description:** The recipient to which the event is being transmitted.

onCaching

Called right after the event has been put into the Local Event Cache for later transmission (e.g., when trying to send it to a remote entity but no connectivity is currently available).

Signature

```
void onCaching(in WebinosEvent event);
```

Parameters

- **event**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEvent
  - **Description:** The event being cached.

onDelivery

Called as soon as successful event delivery has been reported by a given recipient or if the recipient notifies that it did already receive an event with the same ID.

Signature

```
void onDelivery(in WebinosEvent event, in WebinosEventEntity
recipient);
```

This callback SHALL NOT be called more than once per recipient for each send/forward.

Conforming implementations SHALL set the "Delivery notification wanted" attribute described in the Webinos system specifications as true whenever this callback is specified.

Parameters

- **event**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEvent
  - **Description:** The event that was successfully delivered.
- **recipient**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEventEntity
  - **Description:** The recipient that notified delivery success.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`onTimeout`

Called right after the moment in time specified by the "referenceTimeout" attribute is reached and the given primary recipient did not successfully sent back at least one event in response to the given event.

Signature

```
void onTimeout(in WebinosEvent event, in WebinosEventEntity recipient);
```

This callback SHALL only be called if the "referenceTimeout" attribute indicates a moment in time in the future at sending/forwarding time and SHALL NOT be called more than once per recipient for each send/forward.

Parameters

- **event**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEvent
  - **Description:** The event.
- **recipient**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEventEntity
  - **Description:** The recipient that did not successfully sent back at least one event in response to the given event.

`onError`

Called as soon as unsuccessful event delivery has been reported w.r.t. a given recipient.

Signature

```
void onError(in WebinosEvent event, in WebinosEventEntity recipient, in
WebinosEventDeliveryError error);
```

This callback SHALL be called when the recipient reports unsuccessful delivery or when it was not possible to send the message and the event expired.

This callback SHALL NOT be called more than once per recipient for each send/forward.

Conforming implementations SHALL set the "Delivery notification wanted" attribute described in the Webinos system specifications as true whenever this callback is specified.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Parameters

- **event**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEvent
  - **Description:** The event that was not successfully delivered.
- **recipient**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEventEntity
  - **Description:** The recipient that notified delivery error.
- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEventDeliveryError
  - **Description:** The reported error.

2.6. WebinosEventListener

The WebinosEventListener interface describes an event listener callback.

```
[Callback=FunctionOnly] interface WebinosEventListener {
        void handleEvent(in WebinosEvent event);
};
```

Methods

`handleEvent`

Called when a new event is received.

Signature

```
void handleEvent(in WebinosEvent event);
```

Parameters

- **event**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEvent

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

- **Description:** The event.

2.7. WebinosEventsInterface

The WebinosEventsInterface interface describes the part of the Event Handling API accessible through the webinos.events object.

```
[NoInterfaceObject] interface WebinosEventsInterface {
        WebinosEvent createWebinosEvent(
                        in DOMString type,
                        in WebinosEventAddressing addressing,
                        [TreatUndefinedAs=Null]
                          in optional DOMString? payload,
                        in optional WebinosEvent? inResponseTo,
                        in optional boolean withTimeStamp,
                        in optional DOMTimeStamp? expiryTimeStamp,
                        in optional boolean addressingSensitive)
                     raises(WebinosEventException);
        DOMString addWebinosEventListener(
                        in WebinosEventListener listener,
                        [TreatUndefinedAs=Null]
                          in optional DOMString? type,
                        in optional WebinosEventEntity? source,
                        in optional WebinosEventEntity? destination)
                  raises(WebinosEventException);
        void removeWebinosEventListener(in DOMString listenerId)
             raises(WebinosEventException);
};
```

Methods

`createWebinosEvent`

Creates a new event.

Signature

```
WebinosEvent createWebinosEvent(in DOMString type, in WebinosEventAddressing addressing, in optional DOMString? payload, in optional WebinosEvent? inResponseTo, in optional boolean withTimeStamp, in optional DOMTimeStamp? expiryTimeStamp, in optional boolean addressingSensitive);
```

The function SHALL accept the "addressing" argument both in normalized and non-normalized form and, in either case, the resulting "addressing" attribute in the newly created WebinosEvent object SHALL reference a newly created and normalized equivalent of such argument where:
- entries that are found both in "to" and "cc" in the original argument are removed from the "cc" array of the resulting WebinosEventAddressing object;
- entries that are found both in "to" and "bcc" in the original argument are removed from the "bcc" array of the resulting WebinosEventAddressing object;
- entries that are found both in "cc" and "bcc" in the original argument are removed from the "bcc" array of the resulting WebinosEventAddressing object.

Furthermore, the function SHALL also make sure that it is valid for the application to create the event on the behalf of the specified source, otherwise an

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

WebinosEventException exception with error code INVALID_ARGUMENT_ERROR SHALL be thrown.

Parameters

- **type**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** Event type identifier.
- **addressing**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEventAddressing
  - **Description:** References to the sending entity on the behalf of which the application wants to create the event and to the event recipients.
- **payload**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** DOMString
  - **Description:** Event type-specific data or null (undefined is considered as equivalent to null).
- **inResponseTo**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** WebinosEvent
  - **Description:** Event that this event is a response to (undefined is considered as equivalent to null).
- **withTimeStamp**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** boolean

  - **Description:** Whether to set the generation timestamp (undefined is considered as equivalent to false).

- **expiryTimeStamp**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** DOMTimeStamp
  - **Description:** Moment in time past which the event is no more valid or meaningful (undefined is considered as equivalent to null).

- **addressingSensitive**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** Whether the addressing information is part of the informative content of the event (undefined is considered as equivalent to false).

Return value
Newly created WebinosEvent object or null if an error occurred.

Exceptions

- WebinosEventException:

  INVALID_ARGUMENT_ERROR if any of the supplied arguments is not valid.

`addWebinosEventListener`

Registers an event listener.

Signature
```
DOMString addWebinosEventListener(in WebinosEventListener listener, in
optional DOMString? type, in optional WebinosEventEntity? source, in
optional WebinosEventEntity? destination);
```

The arguments to this function act as filters, in the sense that when a new event is received, the listener is called if the event's attributes match with all arguments passed to this function.

When a new event is received, all listeners that were registered via this function with matching arguments SHALL be called, yet the order of such calls is unspecified.

Registering a listener SHALL NOT have consequences on other listeners, hence it SHALL be possible to register multiple listeners to the same event type/source/destination combinations.

Parameters

- **listener**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** WebinosEventListener
  - **Description:** The event listener.
- **type**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** DOMString
  - **Description:** Specific event type or null for any type (undefined is considered as null).
- **source**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** WebinosEventEntity
  - **Description:** Specific event source or null for any source (undefined is considered as null).
- **destination**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** WebinosEventEntity
  - **Description:** Specific event recipient (whether primary or not) or null for any destination (undefined is considered as null).

Return value

Listener identifier.

Exceptions

- WebinosEventException:

  INVALID_ARGUMENT_ERROR if any of the supplied arguments is not valid.

`removeWebinosEventListener`

Unregisters an event listener.

Signature

```
void removeWebinosEventListener(in DOMString listenerId);
```

Unregistering a listener SHALL NOT have consequences on other listeners.

Parameters

- **listenerId**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** Listener identifier as returned by addWebinosEventListener().

Exceptions

- WebinosEventException:

  INVALID_ARGUMENT_ERROR if any of the supplied arguments is not valid.

2.8. WebinosEvents

The WebinosEvents interface describes the part of the Event Handling API accessible through the webinos object.

```
[NoInterfaceObject] interface WebinosEvents {
        readonly attribute WebinosEventsInterface events;
};
webinoscore::Webinos implements WebinosEvents;
```

Attributes

`readonly WebinosEventsInterface events`

webinos.events object.

This attribute is readonly.

3. Exceptions

3.1. WebinosEventException

Error codes for the events module.

```
exception WebinosEventException {
        unsigned short code;
        DOMString message;

        const unsigned short INVALID_ARGUMENT_ERROR     = 1;
        const unsigned short PERMISSION_DENIED_ERROR    = 2;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Field

`unsigned short code`

Error code.

`DOMString message`

Error description.

4. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/events

5. Full WebIDL

```
module events {
        [NoInterfaceObject] interface WebinosEventEntity {
                attribute DOMString id;
        };

        [NoInterfaceObject] interface WebinosEventAddressing {
                attribute WebinosEventEntity source;
                attribute WebinosEventEntity[] to;
                attribute WebinosEventEntity[] cc;
                attribute WebinosEventEntity[] bcc;
        };

        [NoInterfaceObject] interface WebinosEvent {
                readonly attribute DOMString type;
                readonly attribute WebinosEventAddressing addressing;
                readonly attribute DOMString id;
                readonly attribute WebinosEvent inResponseTo;
                readonly attribute DOMTimeStamp? timeStamp;
                readonly attribute DOMTimeStamp? expiryTimeStamp;
                readonly attribute boolean addressingSensitive;
                readonly attribute WebinosEventAddressing forwarding;
                readonly attribute DOMTimeStamp? forwardingTimeStamp;
                readonly attribute DOMString? payload;

                void dispatchWebinosEvent(
                                in optional WebinosEventCallbacks? callbacks,
                                in optional DOMTimeStamp? referenceTimeout,
                                in optional boolean sync)
                     raises(WebinosEventException);
                void forwardWebinosEvent(
                                in WebinosEventAddressing forwarding,
                                in optional boolean withTimeStamp,
                                in optional WebinosEventCallbacks? callbacks,
                                in optional DOMTimeStamp? referenceTimeout,
                                in optional boolean sync)
                     raises(WebinosEventException);
        };
```

```
exception WebinosEventException {
        unsigned short code;
        DOMString message;

        const unsigned short INVALID_ARGUMENT_ERROR     = 1;
        const unsigned short PERMISSION_DENIED_ERROR    = 2;
};

[NoInterfaceObject] interface WebinosEventDeliveryError {
        readonly attribute unsigned short code;
        readonly attribute DOMString message;

        const unsigned short UNKNOWN_ERR                = 0;
        const unsigned short INVALID                    = 1;
        const unsigned short BAD_DESTINATION            = 2;
        const unsigned short EXPIRED                    = 3;
        const unsigned short REFUSED                    = 4;
        const unsigned short NO_REFERENCE               = 5;
};

[Callback, NoInterfaceObject] interface WebinosEventCallbacks {
        void onSending(in WebinosEvent event,
                       in WebinosEventEntity recipient);
        void onCaching(in WebinosEvent event);
        void onDelivery(in WebinosEvent event,
                        in WebinosEventEntity recipient);
        void onTimeout(in WebinosEvent event,
                       in WebinosEventEntity recipient);
        void onError(in WebinosEvent event,
                     in WebinosEventEntity recipient,
                     in WebinosEventDeliveryError error);
};

[Callback=FunctionOnly] interface WebinosEventListener {
        void handleEvent(in WebinosEvent event);
};

[NoInterfaceObject] interface WebinosEventsInterface {
        WebinosEvent createWebinosEvent(
                        in DOMString type,
                        in WebinosEventAddressing addressing,
                        [TreatUndefinedAs=Null]
                          in optional DOMString? payload,
                        in optional WebinosEvent? inResponseTo,
                        in optional boolean withTimeStamp,
                        in optional DOMTimeStamp? expiryTimeStamp,
                        in optional boolean addressingSensitive)
                     raises(WebinosEventException);
        DOMString addWebinosEventListener(
                        in WebinosEventListener listener,
                        [TreatUndefinedAs=Null]
                          in optional DOMString? type,
                        in optional WebinosEventEntity? source,
                        in optional WebinosEventEntity? destination)
                  raises(WebinosEventException);
        void removeWebinosEventListener(in DOMString listenerId)
             raises(WebinosEventException);
        };

        [NoInterfaceObject] interface WebinosEvents {
                readonly attribute WebinosEventsInterface events;
        };

        webinoscore::Webinos implements WebinosEvents;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The AppLauncher module

Webinos API Specifications

29 Jun 2011

Authors

- Michael Vakulenko <michael@visionmobile.com>

Abstract

The application execution API (AppLauncher) allows activation of webinos applications installed locally on the device. Functionality defined in this version of the specification refers to webinos Phase 1 scope.

Summary of Methods

Interface	Method
AppLauncherManager	PendingOperation launchApplication(SuccessCallback successCallback, ErrorCallback errorCallback, applicationID appID, ObjectArray params) boolean AppInstalled(applicationID appID)
SuccessCallback	void onSuccess()
ErrorCallback	void onError(LaucherAPIError error)
LauncherAPIError	
PendingOperation	void cancel()
WebinosLauncher	

1. Introduction

The application execution API allows activation of webinos applications installed locally on the device. The API is modelled after BONDI v1.1 AppLauncher API.

Operation of the API is guided by application execution policies, which can be modified by user. The policies control the following aspects of API operation:
- Enable/disable of activation of native applications
- Enable/disable of activation of webinos installable applications
- Enable/disable of notifications to users when a webinos application attempts to activate another application
- Enable/disable of application ability to discover installed applications
- Enable/disable of logging of operations performed using the API

The application execution API provides mechanisms for webinos applications to check if specific webinos application is installed in the device.

2. Interfaces

2.1. AppLauncherManager

NOTE:
- applicationID type will be defined in webinoscore module. Each application will have a unique ID coming from its manifest file. applicationID is a string composed of ID from the app certificate and identifiers assigned by the maker of the app. For the purposes of phase 1 AppLauncher API, we can assume the ID is a string that will be known to the application that starts another application and is known to the runtime based on manifests of installed apps.

```
  [NoInterfaceObject] interface AppLauncherManager {

    PendingOperation launchApplication(in SuccessCallback successCallback,
                                       in ErrorCallback errorCallback,
                                       in applicationID appID,
                                       in optional ObjectArray params)
                        raises( AppLauncherException);

    boolean AppInstalled(in applicationID appID)
            raises( AppLauncherException);
    };
```

This is the main interface for the AppLauncher API.

Code example

```
        // Define the launchApplication success callback.
        function launchedCallback(response)    {
               alert("Webinos application launched successfully");
        }
        // Define the error callback
        function errorCallback(response) {
                         alert(  "The  following  error:  "  +    response.code  +  ",
occurred");
        }
        //  Activate  webinos  application,  if  the  application  is  installed  in  the
device.
        var appinstalled = webinos.AppLauncher.AppInstalled( appID);

        if ( appinstalled)  {
            webinos.AppLauncher.launchApplication(launchedCallback,      errorCallback,
appID, null);
        }
```

Methods

`launchApplication`

Starts a webinos applicaiton identified by appID. The method is asynchronous. If the app was started successfuly, successCallback is invoked. In case of error, errorCallback is called.

Signature

```
PendingOperation launchApplication(in SuccessCallback successCallback,
in ErrorCallback errorCallback, in applicationID appID, in optional
ObjectArray params);
```

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** SuccessCallback
  - **Description:** Callback invoked when a requested webinos app was activated successfully.
- **errorCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ErrorCallback
  - **Description:** Callback invoked if activation of webinos app was not successful.
- **appID**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** applicationID
  - **Description:** Identifies webinos application that needs to be activated.
- **params**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** ObjectArray

    - **Description:** Optional set of parameters for starting the application.

Return value

A pending operation object

Exceptions

- AppLauncherException:

  Thrown when activation of the application was not successful.

`AppInstalled`

Reports if a specific webinos application is installed in the device. The method is synchronous.

Signature

```
boolean AppInstalled(in applicationID appID);
```

Parameters

- **appID**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** applicationID
    - **Description:** Identifies webinos application presence of which needs to be tested.

Return value

True if the application is installed, false if the application is not installed.

Exceptions

- AppLauncherException:

  Thrown when activation of the application was not successful.

2.2. SuccessCallback

This interface defines the callback for a asynchronous launchApplication method.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface SuccessCallback{
  void onSuccess();
};
```

Methods

`onSuccess`

This method is called if function app was launched successfully.

Signature

```
void onSuccess();
```

2.3. ErrorCallback

This interface defines the callback for a failed activation of asynchronous launchApplication method.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface ErrorCallback{
  void onError(in LaucherAPIError error);
};
```

Methods

onError

This method is called if asychronous launchApplication method fails.

Signature

```
void onError(in LaucherAPIError error);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** LaucherAPIError
  - **Description:** contains information about the error

2.4. LauncherAPIError

API-specific error handling interface

```
[NoInterfaceObject] interface LauncherAPIError {
    readonly attribute unsigned short code;
    readonly attribute DOMString message;
    const unsigned short      UNKNOWN_ERR                    = 0;
    const unsigned short      NOT_SUPPORTED_ERR              = 9;
    const unsigned short      TYPE_MISMATCH_ERR              = 17;
    const unsigned short      SECURITY_ERR                   = 18;
    const unsigned short      NETWORK_ERR                    = 19;
    const unsigned short      INVALID_APP_ID                 = 100;
    const unsigned short      APP_NOT_FOUND                  = 101;
    const unsigned short      NO_RESOURCES                   = 102;
    const unsigned short      ALREADY_STARTED                = 103;
    const unsigned short      POLICY_NOT_ALLOWED             = 104;
};
```

The LaucnherAPIError interface describes error interface for the Launcher API.

2.5. PendingOperation

The PendingOperation interface

```
[NoInterfaceObject] interface PendingOperation {
   void cancel ();
};
```

The PendingOperation interface describes operation of cancellable aynchronous methods. Cancellable asynchronous methods return PendingOperation objects exporting methods for cancelling the operation.

Methods

`cancel`

Cancel method for cancelling asynchronous operation

Signature

`void cancel();`

Cancel ongoing asynchronous method call. Upon calling this method the runtime must immediately stop the pending operation and return.

2.6. WebinosLauncher

The WebinosLauncher interface describes the part of the App Execution API accessible through the webinos object.

```
[NoInterfaceObject] interface WebinosLauncher {
        readonly attribute AppLauncherManager launcher;
};
webinoscore::Webinos implements WebinosLauncher;
```

Attributes

`readonly AppLauncherManager launcher`

webinos.launcher object.

This attribute is readonly.

3. Type Definitions

3.1. ObjectArray

Array of DOMStrings.

```
typedef sequence<object> ObjectArray;
```

3.2. applicationID

Application ID for identifying installed webinos applications. NOTE: This definition could be moved to webinos core module in the future.

```
typedef DOMString applicationID;
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

4. Exceptions

4.1. AppLauncherException

Exception definition for AppLauncher module. Error codes are defined in LauncherAPIError interface.

```
exception AppLauncherException {
  unsigned short code;
  DOMString message;
};
```

5. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/applauncher.launch

> Start webinos application - allows to invoke webinos application identified by a unique identifier.

http://webinos.org/api/applauncher.check

> Check if application is installed - allow to test of an application identified by a specific application ID is installed on the device.

6. Full WebIDL

```
module AppLauncher {
  typedef sequence<object> ObjectArray;
  typedef DOMString applicationID;

  exception AppLauncherException {
    unsigned short code;
    DOMString message;
  };
  [NoInterfaceObject] interface AppLauncherManager {
    PendingOperation launchApplication(in SuccessCallback successCallback,
                                       in ErrorCallback errorCallback,
                                       in applicationID appID,
                                       in optional ObjectArray params)
                        raises( AppLauncherException);

    boolean AppInstalled(in applicationID appID)
            raises( AppLauncherException);
    };

    [Callback=FunctionOnly, NoInterfaceObject]
    interface SuccessCallback{
      void onSuccess();
    };

    [Callback=FunctionOnly, NoInterfaceObject]
    interface ErrorCallback{
      void onError(in LaucherAPIError error);
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
    };

    [NoInterfaceObject] interface LauncherAPIError {
        readonly attribute unsigned short code;
        readonly attribute DOMString message;
        const unsigned short      UNKNOWN_ERR                     = 0;
        const unsigned short      NOT_SUPPORTED_ERR               = 9;
        const unsigned short      TYPE_MISMATCH_ERR               = 17;
        const unsigned short      SECURITY_ERR                    = 18;
        const unsigned short      NETWORK_ERR                     = 19;
        const unsigned short      INVALID_APP_ID                  = 100;
        const unsigned short      APP_NOT_FOUND                   = 101;
        const unsigned short      NO_RESOURCES                    = 102;
        const unsigned short      ALREADY_STARTED                 = 103;
        const unsigned short      POLICY_NOT_ALLOWED              = 104;
    };

     [NoInterfaceObject] interface PendingOperation {
        void cancel ();
     };

        [NoInterfaceObject] interface WebinosLauncher {
                readonly attribute AppLauncherManager launcher;
        };

        webinoscore::Webinos implements WebinosLauncher;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The messaging module

Webinos API Specifications

28 Jun 2011

Authors

- WAC 2.0 Proposed Release Version (PRV) 28 January 2011
- Extended with InstantMessaging functionality for webinos by Christian Fuhrhop <christian.fuhrhop@fokus.fraunhofer.de>

Abstract

WAC based Messaging interface.

Summary of Methods

Interface	Method
DeviceapisMessaging	
Messaging	Message createMessage(short type) PendingOperation sendMessage(SuccessCallback successCallback, ErrorCallback errorCallback, Message message) PendingOperation sendMessage(MessageSendCallback successCallback, ErrorCallback errorCallback, Message message) PendingOperation findMessages(FindMessagesSuccessCallback successCallback, ErrorCallback errorCallback, MessageFilter filter) unsigned long onSMS(OnIncomingMessage messageHandler) unsigned long onMMS(OnIncomingMessage messageHandler) unsigned long onEmail(OnIncomingMessage messageHandler) unsigned long onIM(OnIncomingMessage messageHandler) void unsubscribe(unsigned long subscriptionHandler)
Message	PendingOperation update(UpdateMessageSuccessCallback successCallback, ErrorCallback errorCallback)

Interface	Method
MessageFilter	
MessageAttachment	
FindMessagesSuccessCallback	void onsuccess(MessageArray messages)
UpdateMessageSuccessCallback	void onsuccess(Message message)
OnIncomingMessage	void onevent(Message message)
MessageSendCallback	void onsuccess() void onmessagesendsuccess(DOMString recipient) void onmessagesenderror(DeviceAPIError error, DOMString recipient)
PendingOperation	void cancel()

1. Introduction

The messaging API provides access to the following capabilities: Sending messages through different technologies: SMS, MMS, Email and Instant Messages. Search for messages in the different folders. Subscribe for being notified upon incoming message events.

This API is a read only API that does not allow message or folder management.

2. Interfaces

2.1. DeviceapisMessaging

Defines what is instantiated in the deviceapis object

```
[NoInterfaceObject] interface DeviceapisMessaging {
  readonly attribute Messaging messaging;
};
Deviceapis implements DeviceapisMessaging;
```

When the messaging feature is instantiated, the messaging object is available in the deviceapis object.

2.2. Messaging

Messaging creation, sending and reading capabilities

```
[NoInterfaceObject] interface Messaging {
  const short TYPE_SMS = 1;
  const short TYPE_MMS = 2;
  const short TYPE_EMAIL = 3;
  const short TYPE_IM = 3;
  const unsigned short FOLDER_INBOX = 1;
  const unsigned short FOLDER_OUTBOX = 2;
  const unsigned short FOLDER_DRAFTS = 3;
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
    const unsigned short FOLDER_SENTBOX = 4;

    Message createMessage(in short type)
                          raises(DeviceAPIError);

    PendingOperation sendMessage(in SuccessCallbackSuccessCallbackSuccessCallback
successCallback,
                                 in ErrorCallbackErrorCallbackErrorCallback
errorCallback,
                                 in Message message)
                                 raises(DeviceAPIError);

    PendingOperation sendMessage(in MessageSendCallback successCallback,
                                 in           ErrorCallbackErrorCallbackErrorCallback
errorCallback,
                                 in Message message)
                                 raises(DeviceAPIError);

    PendingOperation findMessages(in FindMessagesSuccessCallback successCallback,
                                  in  optional  ErrorCallbackErrorCallbackErrorCallback
errorCallback,
                                  in optional MessageFilter filter)
                                  raises(DeviceAPIError);

    unsigned long onSMS(in OnIncomingMessage messageHandler)
                        raises(DeviceAPIError);

    unsigned long onMMS(in OnIncomingMessage messageHandler)
                        raises(DeviceAPIError);

    unsigned long onEmail(in OnIncomingMessage messageHandler)
                          raises(DeviceAPIError);

    unsigned long onIM(in OnIncomingMessage messageHandler)
                          raises(DeviceAPIError);

    void unsubscribe(in unsigned long subscriptionHandler)
                     raises(DeviceAPIError);
  };
```

This interface allows a Web application to create a message through the createMessage() method that returns an instance of the Message interface. That message could be manipulated through the functionality offered by the Message interface and sent afterwards through the sendMessage() method.

Messages created through this API are not persistent in device memory until the implementation tries to send them through the send operation. When that operation has been performed, the message will be available on the relevant folder depending on the result of the operation (e.g. sent, drafts...). The only way to access the messages that have been sent is through the use of the findMessages method. The findMessages method allows developers to retrieve the content of the messages available in the device folders.

This interface also offers mechanism to subscribe for being notified upon incoming message events.

Code example

```
  // Define the success callback
 function messageSent() {
   alert("The SMS has been sent");
 }

 // Define the error callback
 function messageFailed(error) {
   alert("The SMS could not be sent " + error.message);
 }

 // SMS sending example
 var msg = deviceapis.messaging.createMessage(deviceapis.messaging.TYPE_SMS);
 msg.body = "I will arrive in 10 minutes";
 msg.to = ["+34666666666"];

 // Send request
 deviceapis.messaging.sendMessage(messageSent, messageFailed, msg);
```

Constants

`short TYPE_SMS`

Identifier for messages of type SMS.

`short TYPE_MMS`

Identifier for messages of type MMS.

`short TYPE_EMAIL`

Identifier for messages of type Email.

`short TYPE_IM`

Identifier for messages of type IM.

`unsigned short FOLDER_INBOX`

Identifier for the message inbox.

`unsigned short FOLDER_OUTBOX`

Identifier for the message outbox.

`unsigned short FOLDER_DRAFTS`

Identifier for the message draft folder.

`unsigned short FOLDER_SENTBOX`

Identifier for message sent-items folder.

Methods

`createMessage`

Create a message of a given type.

Signature
```
Message createMessage(in short type);
```

Parameters

- **type**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** short
  - **Description:** The type of message that is created. The possible values are: TYPE_SMS, TYPE_MMS, TYPE_EMAIL and TYPE_IM.

Return value

A Message object of the given type or null if there is any problem during the message creation.

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if the input parameter is not compatible with the expected type for that parameter.

  INVALID_VALUES_ERR if the input parameter contains an invalid value.

Code example
```
var msg = deviceapis.messaging.createMessage(deviceapis.messaging.TYPE_SMS);
msg.body = "webinos first SMS message.";
```

`sendMessage`

Attempt to send the specified message.

Signature
```
PendingOperation sendMessage(in SuccessCallback successCallback, in
ErrorCallback errorCallback, in Message message);
```

If the message type is set to email and the user has multiple email accounts set up, the runtime SHOULD use the default e-mail account. If no account has been set up, the runtime MAY either provide respective mechanisms to create a new one or throw the given ErrorCallback back to the requesting widget.

Only the parameters supported by a specific technology and that can be set up by the developer (see Message interface attribute definition) are sent as specified in the following table (the rest are ignored):

Attribute	SMS	MMS	Email	IM
to	Yes	Yes	Yes	Yes
body	Yes	Yes	Yes	Yes
subject	No	Yes	Yes	No
attachments	No	Yes	Yes	No
cc	No	No	Yes	Yes
bcc	No	No	Yes	Yes
priority	No	No	Yes	No

When a message has been successfully or unsuccessfully sent, it will be stored in the relevant folder (e.g. sent folder if successfully sent). Please not that some platforms may store multiple copies of the message if multiple recipients were included.

When the operation is fully completed (i.e. the implementation knows the result of the send operation to all the recipients), the onsuccess method of the successCallback will be invoked if the message is successfully sent to all the recipients.

If any of the input parameters is not compatible with the expected type for that parameter a DeviceAPIError with code TYPE_MISMATCH_ERR MUST be synchronously thrown.

If the operation fails for any other reason, the errorCallback will be invoked with an appropriate error code amongst the following:

INVALID_VALUES_ERR: If any of the input paramters contains an invalid value. E.g. successCallback or message is null, message contains invalid values in any of its attributes... Please note that in order to allow developer ignore errors errorCallback accepts null as a valid value.

NOT_SUPPORTED_ERR: If the specified messaging technology is not supported.

SECURITY_ERR: If the operation is not allowed.

UNKNOWN_ERR: In any other error case.

If the errorCallback does not contain a valid function (e.g. null), in case of any error that should be returned in the errorCallback (see above), the implementation MUST silently fail and no further action is required (i.e. the error is not notified to the developer).

Parameters

- **successCallback**

- **Optional:** No.
- **Nullable**: No
- **Type:** SuccessCallbackSuccessCallbackSuccessCallback
- **Description:** To be invoked if the message is successfully sent.

- **errorCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ErrorCallbackErrorCallbackErrorCallback
  - **Description:** To be invoked in case the sending request fails.
- **message**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Message
  - **Description:** The message to be sent.

Return value

PendingOperation to cancel the asynchronous call

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

Code example

```
// Define the success callback
function messageSent() {
alert("The SMS has been sent to all the recipients");
}

// Define the error callback
function messageFailed(error) {
alert("The SMS could not be sent " + error.message);
}

// SMS sending example
var msg = deviceapis.messaging.createMessage(deviceapis.messaging.TYPE_SMS);
msg.body = "I will arrive in 10 minutes";
msg.to = ["+34666666666", "+34888888888"];

// Send request
deviceapis.messaging.sendMessage(messageSent, messageFailed, msg);
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`sendMessage`

Attempt to send the specified message with per-recipient notification

Signature

```
PendingOperation sendMessage(in MessageSendCallback successCallback, in
ErrorCallback errorCallback, in Message message);
```

If the message type is set to email and the user has multiple email accounts set up, the runtime SHOULD use the default e-mail account. If no account has been set up, the runtime MAY either provide respective mechanisms to create a new one or throw the given ErrorCallback back to the requesting widget.

Only the parameters supported by a specific technology and that can be set up by the developer (see Message interface attribute definition) are sent as specified in the following table (the rest are ignored):

Attribute	SMS	MMS	Email	IM
to	Yes	Yes	Yes	Yes
body	Yes	Yes	Yes	Yes
subject	No	Yes	Yes	No
attachments	No	Yes	Yes	No
cc	No	No	Yes	Yes
bcc	No	No	Yes	Yes
priority	No	No	Yes	No

When a message has been successfully or unsuccessfully sent, it will be stored in the relevant folder (e.g. sent folder if successfully sent). Please not that some platforms may store multiple copies of the message if multiple recipients were included.

For every individual recipient in the destination list, when the message is successfully sent to it the method onmessagesendsuccess of the successCallback argument MUST be invoked. If the message cannot be sent to that recipient, the onmessagesenderror of the successCallback argument MUST be invoked with the recipient and the error code as input parameters. The following error codes may be passed depending on the the error conditions:

INVALID_VALUES_ERR: If any of the input paramters contains an invalid value. E.g. successCallback or message is null, message contains invalid values in any of its attributes... Please note that in order to allow developer ignore errors errorCallback accepts null as a valid value.

NOT_SUPPORTED_ERR: If the specified messaging technology is not supported.

SECURITY_ERR: If the operation is not allowed.

UNKNOWN_ERR: In any other error case.

When the operation is fully completed (i.e. the implementation knows the result of the send operation to all the recipients), the onsuccess method of the successCallback will be invoked if the message is successfully sent to all the recipients.

In case of any error different to a TYPE_MISMATCH_ERR, the errorCallback will be invoked with an appropriate error code amongst the following:

INVALID_VALUES_ERR: If any of the input paramters contains an invalid value. E.g. successCallback or message is null, message contains invalid values in any of its attributes... Please note that in order to allow developer ignore errors errorCallback accepts null as a valid value.

NOT_SUPPORTED_ERR: If the specified messaging technology is not supported.

SECURITY_ERR: If the operation is not allowed.

UNKNOWN_ERR: In any other error case.

If the errorCallback does not contain a valid function (e.g. null), in case of any error that should be returned in the errorCallback (see above), the implementation MUST silently fail and no further action is required (i.e. the error is not notified to the developer).

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** MessageSendCallback
  - **Description:** Contains the methods for individual notifications.
- **errorCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ErrorCallbackErrorCallbackErrorCallback
  - **Description:** To be invoked in case the sending request fails.
- **message**

- **Optional:** No.
- **Nullable**: No
- **Type:** Message
- **Description:** The message to be sent.

Return value

PendingOperation to cancel the asynchronous call

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

Code example

```
// Define the send callback
var messageSendCallback = {
      onsuccess: function() {
        alert("The SMS has been sent to all the recipients");},
      onmessagesendsuccess: function(recipient) {
        alert("The SMS has been sent to " + recipient);},
      onmessagesenderror: function(error, recipient) {
        alert("The SMS has not been sent to " + recipient +
              " error " + error);}
};

// Define the error callback
function messageFailed(error) {
alert("The SMS could not be sent " + error.message);
}

// SMS sending example
var msg = deviceapis.messaging.createMessage(deviceapis.messaging.TYPE_SMS);
msg.body = "I will arrive in 10 minutes";
msg.to = ["+34666666666", "+34888888888"];

// Send request
deviceapis.messaging.sendMessage(messageSendCallback, messageFailed, msg);
```

`findMessages`

Gets an array of messages from the message folders matching the selected filter.

Signature

```
PendingOperation findMessages(in FindMessagesSuccessCallback successCallback, in optional ErrorCallback errorCallback, in optional MessageFilter filter);
```

If any of the input parameters is not compatible with the expected type for that parameter a DeviceAPIError with code TYPE_MISMATCH_ERR MUST be synchronously thrown.

If the this feature is not supported, a DeviceAPIError with code NOT_SUPPORTED_ERR MUST be returned in the errorCallback. If this functionality is not allowed the errorCallback MUST be invoked with a DeviceAPIError with code SECURITY_ERR.

If the successCallback does not contain a Function (i.e. it is null), a DeviceAPIError with code INVALID_VALUES_ERR MUST be returned.

If the filter is passed and contains valid values, only those values in the message lists that matches the filter criteria as specified in the MessageFilter interface will be returned in the successCallback. If no filter is passed, it is null or undefined, or contains any invalid value, the implementation MUST return the full list of messages in the successCallback. If no messages are available in the lists or no one matches the filter criteria, the successCallback will be invoked with an empty array.

If any other error occurs, while trying to retrieve the messages, the errorCallback function that was passed in the invocation MUST be called including a DeviceAPIError object with code UNKNOWN_ERR.

In any of the cases in which the errorCallback should be invoked, if the developer has not passed an ErrorCallback or it is null, no action is required (i.e. the error is not notified to the developer).

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** FindMessagesSuccessCallback
  - **Description:** function called when the invocation ends successfully.
- **errorCallback**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** ErrorCallbackErrorCallbackErrorCallback
  - **Description:** function called when an error occurs
- **filter**
  - **Optional:** Yes.
  - **Nullable**: No

- **Type:** MessageFilter
- **Description:** message data to be used when filtering

Return value

PendingOperation to cancel the asynchronous call

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

Code example

```
   var msg = { type:[deviceapis.messaging.TYPE_SMS], body:"first messa%" };

deviceapis.messaging.findMessages(
   function (messages) {
     alert(messages.length + " message(s) found!");
     for (var i=0; imessages.length; i++) {
       alert(i + ". message from " + messages[i].from);
     }
   },
  null,
  msg);
```

onSMS

Registers the function to be notified on incoming new SMSs

Signature

```
unsigned long onSMS(in OnIncomingMessage messageHandler);
```

When this method is invoked, the implementation MUST register the function passed in the messageHandler argument as the handler for being notified whenever an incoming SMS arrives to the device. That function will be invoked every time an incoming SMS arrives, unless the unsubscribe method with the handler identifier is invoked in order to cancel the subscription.

If the subscription is successfully created, an identifier for the handler is created and returned so that it is possible to cancel the subscription. If the subscription cannot be created, a DeviceAPIError is synchronously thrown with an error code that describes the reason for the error.If any of the input parameters is not compatible with the expected type for that parameter a DeviceAPIError with code TYPE_MISMATCH_ERR MUST be synchronously thrown.

Parameters

- **messageHandler**
  - **Optional:** No.

- **Nullable**: No
- **Type:** OnIncomingMessage
- **Description:** The function to be invoked on incoming SMSs

Return value

Subscription identifier

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

  INVALID_VALUES_ERR if the messageHandler is null or undefined.

  NOT_SUPPORTED_ERR if this feature is not supported.

  SECURITY_ERR if this operation is not allowed.

Code example

```
// function to receive new SMS notifications
function incomingSMS(message)
{
  alert("New incoming SMS from " + message.from);

  // The subscription is cancelled to prevent further notifications
  if (mySMSListener != null)
    deviceapis.messaging.unsubscribe(mySMSListener);
}
```

`onMMS`

Registers the function to be notified on incoming new MMSs

Signature

```
unsigned long onMMS(in OnIncomingMessage messageHandler);
```

When this method is invoked, the implementation MUST register the function passed in the messageHandler argument as the handler for being notified whenever an incoming MMS arrives to the device. That function will be invoked every time an incoming MMS arrives, unless the unsubscribe method with the handler identifier is invoked in order to cancel the subscription.

If the subscription is successfully created, an identifier for the handler is created and returned so that it is possible to cancel the subscription. If the subscription cannot be created, a DeviceAPIError is synchronously thrown with an error code that describes the reason for the error.

Parameters

- **messageHandler**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** OnIncomingMessage
  - **Description:** The function to be invoked on incoming MMSs

Return value

Subscription identifier

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

  INVALID_VALUES_ERR if the messageHandler is null or undefined.

  NOT_SUPPORTED_ERR if this feature is not supported.

  SECURITY_ERR if this operation is not allowed.

Code example

```
// function to receive new MMS notifications
function incomingMMS(message)
{
  alert("New incoming MMS from " + message.from);

  // The subscription is cancelled to prevent further notifications
  if (myMMSListener != null)
    deviceapis.messaging.unsubscribe(myMMSListener);
}
```

`onEmail`

Registers the function to be notified on incoming new Email

Signature

```
unsigned long onEmail(in OnIncomingMessage messageHandler);
```

When this method is invoked, the implementation MUST register the function passed in the messageHandler argument as the handler for being notified whenever an incoming Email arrives to the device. That function will be invoked every time an incoming Email arrives, unless the unsubscribe method with the handler identifier is invoked in order to cancel the subscription.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

If the subscription is successfully created, an identifier for the handler is created and returned so that it is possible to cancel the subscription. If the subscription cannot be created, a DeviceAPIError is synchronously thrown with an error code that describes the reason for the error.

Parameters

- **messageHandler**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** OnIncomingMessage
    - **Description:** he function to be invoked on incoming emails

Return value

Subscription identifier

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

  INVALID_VALUES_ERR if the messageHandler is null or undefined.

  NOT_SUPPORTED_ERR if this feature is not supported.

  SECURITY_ERR if this operation is not allowed.

Code example

```
// function to receive new Email notifications
 function incomingEmail(message)
 {
   alert("New incoming Email from " + message.from);

   // The subscription is cancelled to prevent further notifications
   if (myEmailListener != null)
     deviceapis.messaging.unsubscribe(myEmailListener);
 }

 // Register listener for new Email events
 var myEmailListener = null;
 myEmailListener = deviceapis.messaging.onEmail(incomingEmail);
```

`onIM`

Registers the function to be notified on incoming new Instant Message

Signature

```
unsigned long onIM(in OnIncomingMessage messageHandler);
```

When this method is invoked, the implementation MUST register the function passed in the messageHandler argument as the handler for being notified whenever an incoming instant message arrives to the device. That function will be invoked every time an incoming Instant Message arrives, unless the unsubscribe method with the handler identifier is invoked in order to cancel the subscription.

If the subscription is successfully created, an identifier for the handler is created and returned so that it is possible to cancel the subscription. If the subscription cannot be created, a DeviceAPIError is synchronously thrown with an error code that describes the reason for the error.

Parameters

- **messageHandler**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** OnIncomingMessage
  - **Description:** he function to be invoked on incoming instant message

Return value

Subscription identifier

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

  INVALID_VALUES_ERR if the messageHandler is null or undefined.

  NOT_SUPPORTED_ERR if this feature is not supported.

  SECURITY_ERR if this operation is not allowed.

Code example

```
// function to receive new Instant Messaging notifications
 function incomingIM(message)
 {
   alert("New incoming Instant Message from " + message.from);

   // The subscription is cancelled to prevent further notifications
   if (myIMListener != null)
     deviceapis.messaging.unsubscribe(myIMistener);
 }

 // Register listener for new Instant Messaging events
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
var myIMistener = null;
myIMistener = deviceapis.messaging.onIM(incomingIM);
```

`unsubscribe`

Cancels a messaging subscription

Signature

```
void unsubscribe(in unsigned long subscriptionHandler);
```

If the subscriptionHandler argument is valid and corresponds to a subscription already in place the subscription process MUST immediately stop and no further message notifications MUST be invoked. If the subscriptionHandler argument does not correspond to a valid subscription, the method should return without any further action.

Parameters

- **subscriptionHandler**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** unsigned long
  - **Description:** identifier of the subscription returned by the onSMS(), onMMS(), onEmail() or onIM() methods.

Return value

void

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

2.3. Message

Defines the content and attributes of a message

```
[NoInterfaceObject] interface Message {
readonly attribute DOMString id;
attribute short type;
attribute short folder;
readonly attribute Date timestamp;
readonly attribute DOMString from;
attribute StringArray to;
attribute StringArray cc;
attribute StringArray bcc;
attribute DOMString body;
attribute boolean isRead;
attribute boolean priority;
```

```
    attribute DOMString subject;
    attribute FileArray attachments;
    PendingOperation update(in UpdateMessageSuccessCallback successCallback,
                           in    optional    ErrorCallbackErrorCallbackErrorCallback
errorCallback)
                           raises(DeviceAPIError);
  };
```

This interface allows a Web application to define the set of properties linked to a message previously created through the createMessage() method in the Messaging Interface.

Additionally, it also allows an application to retrieve the content of a message through the findMessages, onSMS, onMMS and onEmail methods. In those cases, the implementation MAY return in some situations only part of the body because of its size. In those situations the implementation MUST allow the developer to retrieve the remaining part of the message through the use of the sync method member of the Synchronisable interface implemented by Message.

Additionally, for the same reason, the implementation MAY decide to provide only the attachment information but not the attachment content. This is achieved by returning in the attachments attribute not a sequence of Files but a sequence of MessageAttachments (that implement the Synchronisable interface).

If the developer attempts to access an attribute not supported by the messaging technology (see attribute description or summary table in the sendMessage mehtod description), the implementation MUST ignore this attempt.

Code example

```
 var msg = deviceapis.messaging.createMessage(deviceapis.messaging.TYPE_SMS);
 msg.body = "WAC first SMS message.";
 msg.to = ["+34666666666"];
```

Attributes

`readonly DOMString id`

Message unique identifier.

A unique indicator for identifying a message.

This property is a locally unique and persistent id, assigned by the device or the Web runtime environment. For new messages created using Messaging.createMessage(), the id is assigned on the first occasion that the message is processed by the underlying platform such as a call to send(). This property is unique across device power cycles.

This attribute is readonly.

`short type`

The type of the given message.

`short folder`

The folder for the given message.

For messages created through the createMessage method this property is undefined.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`readonly Date timestamp`

The timestamp of a message.

This property is set up by the device or the web runtime environment.

This attribute is readonly.

`readonly DOMString from`

The source address of a message.

This property is set up by the device or the web runtime environment. This property should only be taken into account for Email.

This attribute is readonly.

`StringArray to`

The destination of a message.

`StringArray cc`

The Cc address of a message.

`StringArray bcc`

The Bcc address of a message.

`DOMString body`

The body of a message.

`boolean isRead`

The flag "read" for this Message.

True if the message has been read and false otherwise.

`boolean priority`

The priority of a message.

True means high priority and false normal or low priority. This property should only be taken into account for Email.

`DOMString subject`

The subject of a message.

This property should only be taken into account for MMS and Email.

`FileArray attachments`

The list of message attachments.

This property should only be taken into account for Email and MMS. If the message has not been created by the developer but returned through the findMessage, onMMS or onEmail methods, the attachments will be stored in the "attachments" file system root location, that is only accessible through this API.

Methods

`update`

Updates a message retrieved with the findMessages method

Signature

```
PendingOperation update(in UpdateMessageSuccessCallback
successCallback, in optional ErrorCallback errorCallback);
```

This method is meant to transfer all changes made to the given Message object before (i.e. changed attributes) to the underlying system (e.g. native messaging database and LDAP). If any changes cannot be transferred to the system, they can be ignored by the implementation.

This method does not have effect on messages created through the createMessage method as they are not persistently stored until the send action is invoked.

For messages in the inbox (deviceapis.messaging.FOLDER_INBOX), outbox (deviceapis.messaging.FOLDER_OUTBOX) and sentbox (deviceapis.messaging.FOLDER_SENTBOX) the implementation MUST only change the isRead attribute of the Message object. For messages within the draft folder (deviceapis.messaging.FOLDER_DRAFTS) the implementation MAY update other attributes as well. However, this is up to the actual implementation and relies on the underlying system.

The implementation has to make sure that an updated Message object is provided in the success callback, which represents the current status of the message. The developer is expected to use this updated message for comparison with the former object to check which fields have or have not been transferred by the implementation.

If any of the input parameters is not compatible with the expected type for that parameter a DeviceAPIError with code TYPE_MISMATCH_ERR MUST be synchronously thrown.

If this feature is not supported, a DeviceAPIError with code NOT_SUPPORTED_ERR MUST be returned in the errorCallback. If this functionality is not allowed the errorCallback MUST be invoked with a DeviceAPIError with code SECURITY_ERR.

If the successCallback contains an invalid value (e.g. null or undefined), a DeviceAPIError with code INVALID_VALUES_ERR MUST be returned.

If any other error occurs, while trying to update the messages, the errorCallback function that was passed in the invocation MUST be called including a DeviceAPIError object with code UNKNOWN_ERR.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

If the errorCallback does not contain a valid function (e.g. null), in case of any error that should be returned in the errorCallback (see above), the implementation MUST silently fail and no further action is required (i.e. the error is not notified to the developer).

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** UpdateMessageSuccessCallback
  - **Description:** Function to call on successful update
- **errorCallback**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** ErrorCallbackErrorCallbackErrorCallback
  - **Description:** Function to call on unsuccessful update

Return value

PendingOperation to cancel the asynchronous call

Exceptions

- DeviceAPIError:

  TYPE_MISMATCH_ERR if any input parameter is not compatible with the expected type for that parameter.

2.4. MessageFilter

Filter to restrict the items returned by the findMessages method

```
[Callback, NoInterfaceObject] interface MessageFilter {
  attribute DOMString id;
  attribute ShortArray type;
  attribute ShortArray folder;
  attribute Date startTimestamp;
  attribute Date endTimestamp;
  attribute DOMString from;
  attribute StringArray to;
  attribute StringArray cc;
  attribute StringArray bcc;
  attribute DOMString body;
  attribute boolean isRead;
  attribute boolean messagePriority;
  attribute DOMString subject;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

When used this filter in the findMessages operation, the result-set of the search MUST only contain the Message entries that match the filter values.

An entry matches the filter, if the attributes of the entry matches all the attributes of the filter with values different to undefined or null. I.e. the search is performed in a similar manner to a SQL "AND" operation.

An attribute of the Message entry matches the filter value according to the following rules:

For filter attributes of type DOMString an entry matches this value if its corresponding attribute is exactly the same than the filter one unless the filter contains U+0025 'PERCENT SIGN' wildcard character(s). If wildcards are used, the behavior is similar to the LIKE condition in SQL ('%' matches any string of any length - including zero length). In order to specify that a 'PERCENT SIGN' character is to be considered literally instead of interpreting it as a wildcard, developers may escape it with the backslash character.

For filter attributes of type StringArray the same rules as for filter attributes of type DOMString apply for each of the fields within the given Array separately. The search for all included fields is performed similar to a SQL "AND" operation in the end without taking into account the (possible) difference in ordering between Message fields as well as MessageFilter fields.

For filter attributes of an array of WebIDL numeric type (type), an entry matches it only if the corresponding entry attribute has exactly the same value as any of the array elements.

For filter attributes of any WebIDL boolean type (isRead, messagePriority) an entry matches it only if the corresponding entry attribute has exactly the same state (i.e. true or false).

For message attributes of type Date (i.e. timestamp), a couple of filter attributes are included (initial and end), in order to allow looking for messages between two dates. If both initial and end dates are different to null, a message matches the filter if its corresponding attribute is between initial and end dates (including them). If only the initial date contains a value different to null, a message matches the filter if its corresponding attribute is later than or equal to the initial one. If only the end date contains a value different to null, a message matches the filter if its corresponding attribute is earlier than or equal to the end date.

Attributes

`DOMString id`

> Used for filtering the Message id attribute.
>
> Messages which id corresponds with this attribute (either exactly or with the specified wildcards) match this filtering criteria.

`ShortArray type`

> Used for filtering the Message type attribute.
>
> Messages with type equals to one of the values in this array match the filtering criteria.

`ShortArray folder`

> Used for filtering the Message folder attribute.

Messages with folder equals to one of the values in this array match the filtering criteria.

`Date startTimestamp`

Used for filtering the Message timestamp attribute.

Messages with date later than or equal to this attribute match the filtering criteria.

`Date endTimestamp`

Used for filtering the Message timestamp attribute.

Messages with date earlier than or equal to this attribute match the filtering criteria.

`DOMString from`

Used for filtering the Message from attribute.

Messages which from corresponds with this attribute (either exactly or with the specified wildcards) match this filtering criteria.

`StringArray to`

Used for filtering the Message to attribute.

Messages which elements in the to array that correspond to all the elements of this attribute (either exactly or with the specified wildcards) match this filtering criteria.

`StringArray cc`

Used for filtering the Message cc attribute.

Messages which elements in the cc array that correspond to all the elements of this attribute (either exactly or with the specified wildcards) match this filtering criteria.

`StringArray bcc`

Used for filtering the Message bcc attribute.

Messages which elements in the bcc array that correspond to all the elements of this attribute (either exactly or with the specified wildcards) match this filtering criteria.

`DOMString body`

Used for filtering the Message body attribute.

Messages which body corresponds with this attribute (either exactly or with the specified wildcards) match this filtering criteria.

`boolean isRead`

Used for filtering the Message isRead attribute.

Messages which isRead corresponds exactly with this attribute match this filtering criteria.

`boolean messagePriority`

Used for filtering the Message messagePriority attribute.

Messages which messagePriority corresponds exactly with this attribute match this filtering criteria.

`DOMString subject`

Used for filtering the Message subject attribute.

Messages which subject corresponds with this attribute (either exactly or with the specified wildcards) match this filtering criteria.

2.5. MessageAttachment

Describes a message attachement

```
    interface MessageAttachment : File {
    readonly attribute DOMString MIMEType;
  };
```

This attribute extends the File interface (from W3C File reader (http://dev.w3.org/2006/webapi/FileAPI/#file)) by concurrently implementing the synchronizable interface. It allows attachments to be downloaded only if the user requests them by the use of the sync method specified in the Synchronizable interface.

Attributes

`readonly DOMString MIMEType`

Describes the mime type of the attachment, e.g. "text/html".

This attribute is readonly.

2.6. FindMessagesSuccessCallback

findMessages specific success callback.

```
  [Callback=FunctionOnly, NoInterfaceObject] interface FindMessagesSuccessCallback {
    void onsuccess(in MessageArray messages);
  };
```

This callback interface specifies a success callback with a function taking a list of messages as input argument. It is used in the findMessages asynchronous operation.

Methods

`onsuccess`

Method invoked when the asynchronous call completes successfully

Signature

```
void onsuccess(in MessageArray messages);
```

Parameters

- **messages**

- **Optional:** No.
- **Nullable**: No
- **Type:** MessageArray
- **Description:** The list of messages that correspond to the find criteria

Return value

void

2.7. UpdateMessageSuccessCallback

update specific success callback.

```
[Callback=FunctionOnly, NoInterfaceObject] interface UpdateMessageSuccessCallback {
  void onsuccess(in Message message);
};
```

This callback interface specifies a success callback with a function that will provide a message object that is meant to represent the actual status of a message after an update has been triggered. It is used in the update asynchronous operation on the Message interface.

Methods

onsuccess

Method invoked when the asynchronous call completes successfully

Signature

```
void onsuccess(in Message message);
```

Parameters

- **message**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Message
  - **Description:** The new message representing the actual updated status

Return value

void

2.8. OnIncomingMessage

Interface for specifying the method called on new incoming message events.

```
[Callback=FunctionOnly, NoInterfaceObject] interface OnIncomingMessage {
   void onevent (in Message message);
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

This interface specifies a function that will provide a message object that represents the received message. It is used in the onSMS(), onMMS(), onEmail() method invocation.

Methods

onevent

Method invoked when an incoming message is received

Signature

```
void onevent(in Message message);
```

Parameters

- **message**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Message
  - **Description:** The message received

Return value

void

2.9. MessageSendCallback

Interface for specifying the methods to be called for message send results for each recipient.

```
[Callback, NoInterfaceObject] interface MessageSendCallback {
   void onsuccess();
   void onmessagesendsuccess(in DOMString recipient);
   void onmessagesenderror(in DeviceAPIError error, in DOMString recipient);
};
```

This interface specifies a set of functions that will be invoked every time the result of the send operation to a recipient is obtained or when the message is successfully sent to all the recipients.

Methods

onsuccess

Method invoked when the message is successfully sent to all the recipients

Signature

```
void onsuccess();
```

Return value

void

onmessagesendsuccess

Method invoked when the message is successfully sent to a recipient

Signature
```
void onmessagesendsuccess(in DOMString recipient);
```

Parameters

- **recipient**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** The recipient which the message has been sent to

Return value
void

`onmessagesenderror`

Method invoked when the message is unsuccessfully sent to a recipient

Signature
```
void onmessagesenderror(in DeviceAPIError error, in DOMString
recipient);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DeviceAPIError
  - **Description:** The error code that identifies the reason of the failure
- **recipient**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** The recipient which the message has been sent to

Return value
void

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

2.10. PendingOperation

The PendingOperation interface

```
[NoInterfaceObject] interface PendingOperation {
  void cancel ();
};
```

The PendingOperation interface describes operation of cancellable aynchronous methods. Cancellable asynchronous methods return PendingOperation objects exporting methods for cancelling the operation.

Methods

`cancel`

Cancel method for cancelling asynchronous operation

Signature

```
void cancel();
```

Cancel ongoing asynchronous method call. Upon calling this method the runtime must immediately stop the pending operation and return.

3. Type Definitions

3.1. MessageArray

Sequence of Message objects

```
typedef sequence<Message> MessageArray;
```

3.2. FileArray

Array of files

```
typedef File[]  FileArray;
```

4. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/messaging

Access to the full Messaging module except the methods Messaging.sendMessage and Messaging.find and the attribute Message.attachments

http://webinos.org/api/messaging.send

Access to the Messaging.sendMessage() method

http://webinos.org/api/messaging.find

Access to the Messaging.findMessages method http://webinos.org/api/messaging.subscribe

Access to the Messaging.onSMS, Messaging.onMMS, Messaging.onEmail, Messaging.onIM methods

http://webinos.org/api/messaging.attach

Access to the Message.attachments attribute.

5. Full WebIDL

```
module messaging {
  typedef sequence<Message> MessageArray;
   typedef File[]  FileArray;

  [NoInterfaceObject] interface DeviceapisMessaging {
    readonly attribute Messaging messaging;
  };
  Deviceapis implements DeviceapisMessaging;

  [NoInterfaceObject] interface Messaging {
    const short TYPE_SMS = 1;
    const short TYPE_MMS = 2;
    const short TYPE_EMAIL = 3;
    const short TYPE_IM = 3;
    const unsigned short FOLDER_INBOX = 1;
    const unsigned short FOLDER_OUTBOX = 2;
    const unsigned short FOLDER_DRAFTS = 3;
    const unsigned short FOLDER_SENTBOX = 4;

    Message createMessage(in short type)
                          raises(DeviceAPIError);
    PendingOperation    sendMessage(in    SuccessCallbackSuccessCallbackSuccessCallback
successCallback,
                                 in            ErrorCallbackErrorCallbackErrorCallback
errorCallback,
                                 in Message message)
                                 raises(DeviceAPIError);
    PendingOperation sendMessage(in MessageSendCallback successCallback,
                                 in            ErrorCallbackErrorCallbackErrorCallback
errorCallback,
                                 in Message message)
                                 raises(DeviceAPIError);
    PendingOperation findMessages(in FindMessagesSuccessCallback successCallback,
                                  in  optional  ErrorCallbackErrorCallbackErrorCallback
errorCallback,
                                  in optional MessageFilter filter)
                                  raises(DeviceAPIError);
    unsigned long onSMS(in OnIncomingMessage messageHandler)
                        raises(DeviceAPIError);
    unsigned long onMMS(in OnIncomingMessage messageHandler)
                        raises(DeviceAPIError);
    unsigned long onEmail(in OnIncomingMessage messageHandler)
                          raises(DeviceAPIError);
    unsigned long onIM(in OnIncomingMessage messageHandler)
                          raises(DeviceAPIError);
    void unsubscribe(in unsigned long subscriptionHandler)
```

```
                    raises(DeviceAPIError);
  };
    [NoInterfaceObject] interface Message {
    readonly attribute DOMString id;
    attribute short type;
    attribute short folder;
    readonly attribute Date timestamp;
    readonly attribute DOMString from;
    attribute StringArray to;
    attribute StringArray cc;
    attribute StringArray bcc;
    attribute DOMString body;
    attribute boolean isRead;
    attribute boolean priority;
    attribute DOMString subject;
    attribute FileArray attachments;
    PendingOperation update(in UpdateMessageSuccessCallback successCallback,
                           in     optional     ErrorCallbackErrorCallbackErrorCallback
errorCallback)
                           raises(DeviceAPIError);
  };
  Message implements Synchronisable;
  [Callback, NoInterfaceObject] interface MessageFilter {
    attribute DOMString id;
    attribute ShortArray type;
    attribute ShortArray folder;
    attribute Date startTimestamp;
    attribute Date endTimestamp;
    attribute DOMString from;
    attribute StringArray to;
    attribute StringArray cc;
    attribute StringArray bcc;
    attribute DOMString body;
    attribute boolean isRead;
    attribute boolean messagePriority;
    attribute DOMString subject;
  };
    interface MessageAttachment : File {
    readonly attribute DOMString MIMEType;
  };
  [Callback=FunctionOnly, NoInterfaceObject] interface FindMessagesSuccessCallback {
    void onsuccess(in MessageArray messages);
  };
  [Callback=FunctionOnly, NoInterfaceObject] interface UpdateMessageSuccessCallback {
    void onsuccess(in Message message);
  };
  [Callback=FunctionOnly, NoInterfaceObject] interface OnIncomingMessage {
     void onevent (in Message message);
  };
  [Callback, NoInterfaceObject] interface MessageSendCallback {
     void onsuccess();
     void onmessagesendsuccess(in DOMString recipient);
     void onmessagesenderror(in DeviceAPIError error, in DOMString recipient);
  };
     [NoInterfaceObject] interface PendingOperation {
        void cancel ();
     };
};
```

APIs: The nfc module

Webinos API Specifications

29 Jun 2011

Authors

- Hans Myrhaug (AmbieSense Ltd.) <hans@ambiesense.com>
- Stefano Vercelli (Telecom Italia) <stefano.vercelli@telecomitalia.it>

Abstract

Near Field Communication (NFC) is an international standard (ISO/IEC 18092) that specifies an interface and protocol for simple wireless interconnection of closely coupled devices operating at 13.56 MHz. (http://www.nfc-forum.org/specs/spec_list/). There are three groups of application scenarios for NFC: The first one is to hold a device close to a wireless tag to exchange some digital information or data. The second is to hold two devices close to each other in order to exchange some information or data between them. The third one is to make payments by holding mobile phones close to point of sales terminals instead of swiping smart cards.

Summary of Methods

Interface	Method
PendingOperation	boolean cancel()
NFCTag	void initNFCTagEvent(DOMString type, boolean bubbles, boolean cancelable, ByteArray tagId, unsigned short technologyType, unsigned short ndefType, unsigned short ndefRecType, DOMString? ndefRecordTextPayload, ByteArray ndefRecordBinaryPayload)
NFCTagTechnology	void connect() void close()
NFCTagTechnologyNdef	PendingOperation makeReadOnly(SuccessCallback successCallback, ErrorCallback? errorCallback) NdefMessage readCachedNdefMessage()

Interface	Method
	PendingOperation readNdefMessage(NdefSuccessCallback successCallback, ErrorCallback? errorCallback) PendingOperation writeNdefMessage(SuccessCallback successCallback, ErrorCallback? errorCallback, NdefMessage message) NdefMessage createNdefMessage()
NdefMessage	void addTextNdefRecord(unsigned short type, DOMString payload) void addBinaryNdefRecord(unsigned short type, ByteArray payload)
NdefRecord	
NfcError	
SuccessCallback	void onsuccess()
ErrorCallback	void onError(NfcError error)
NdefSuccessCallback	void onSuccess(NdefMessage obj)

1. Introduction

Near Field Communication is a kind of radio-frequency identification (RFID) technology that uses short-hold wireless communication to transfer messages between wireless NFC devices and NFC tags. The wireless tags are physically attached onto/ mounted nearby a physical object.

The most common use case is for an NFC device to read the identifier and/or the contents of an NFC tag. Another quite common use case is for an NFC device to write content to an NFC tag, if the NFC tag allows this. The webinos NFC module supports both. There is also a third use case where NFC devices pretend to be contactless smart cards e.g. for payment or ticketing purposes. In general, our objective is with the webinos NFC module is to enable free competition for NFC applications. This means that the goal is to allow any application developers to fully operate on any of these modes.

The purpose of an NFC tag is to provide a small amount of digital data about the physical object that it is associated with - or to help perform a task for the user. One can use either the tag identifier, or some data stored on the tag, to achieve this.

An NFC tag can be viewed as a wireless bar code. It can be read by NFC devices within a range of up to 10 centimetres. The amount of power and resources needed by the NFC device to read and write to an NFC tag is very low.

- NFC APPLICATIONS AND USE CASES

NFC technology is increasingly taking part in every day activities. NFC enables digital data to be associated with real world objects. NFC tags are more advanced than printed bar codes, because one can store a few kilobytes of data on each NFC tag in addition to using its identifier.

NFC is a wireless standard where messages must be exchanged and communicated in a standardised way. This makes NFC very suitable for a wide range of ubiquitous applications. Example applications are within logistics, health care, social media, infotainment, gaming, mobile payments, access to places, system access, inventory control, exchange of business cards, email addresses, web links, images, and so on.

- NFC CORE CONCEPTS

The core concepts of the NFC standard are:

* The NFC devices and NFC tags * The NFC Data Exchange Format

NFC devices are are typically mobile phones or computers with some NFC hardware and a driver installed. Such devices are typically active and try to detect tags, or other devices, nearby. The devices and tags can be provided with identifiers, and these can be both fixed or dynamic depending on the application.

NFC tags can be worn by people or attached to objects in the environment. It enables identification and exchange of a small amount of data in the form of standardised messages that can contain up to several data records each. Each data record contains a header identifying the data followed by the actual data itself. It is completely up to application developers to identify the data, to understand it, and to use the data. Please note that sometimes the term payload is also being used in NFC. Payload in an NFC context simply refers to the raw data or information being stored.

- THE NFC DATA EXCHANGE FORMAT

The NFC Forum has specified the NFC Data Exchange Format (NDEF) to enable interoperability when exchanging data between NFC devices and NFC tags. NDEF is a standard that specifies the NDEF data structure format along with rules on how to compose an NDEF message as a complete collection of NDEF records. An NDEF message is a lightweight, binary message format.

It defines how to package application data as NDEF records. NDEF only specified the data structure format to exchange application specific data in an interoperable way. It does not define any record types in detail. Providing the record header and the actual record data is completely up to the application developer to do.

NDEF is a compact and lightweight binary format. It can contain any data such as web links, business cards, tiny applications, images, and so on. It is up the application developers to define and fill it with literally any data suitable or needed for the application. It is the capabilities of the NFC target that matters. Most NFC tags have generally a few kilobytes of available memory to store the data (payload). NFC devices typically have much more memory available. There are currently four types of NFC tags defined in the standard: type1, type 2, type 3, and type 4. The type vary because of computing capabilities and low level commands, but all NFC targets have to support the exchange of NDEF messages and records.

The advantage of NDEF is that it abstracts away from the specific NFC tags/ targets. An NDEF message consists of a set of NDEF records. Each record must carry data/ payload. The type of data can for instance be web links, MIME media types, or pre-defined NFC data types. An NDEF record consists of two parts: 1) the header part, and 2) the data (payload) part. The compact header part specifies the: i) type of data, ii) the length of the data in terms of octets, and iii) an optional data identifier. The optional data identifier could for instance be used by applications to nest data and records, or for other purposes such as signing. i) and ii) are mandatory to specify in the header of a record. Also, the data (payload) part needs to be provided to the record. The approach to specifying headers allows for compact the identification of standardised data formats across NFC applications. It also allows for the identification of new and custom data formats for any future NFC application.

The NDEF message format can accommodate literally any information or data of known and initially unknown sizes. It allows an arbitrary set of information and data to be grouped together into a single message. It also allows for the compact encapsulation of well-known data such as web links. An NDEF message is not a general message description or document format like MIME media types, HTML, XML and so on. Rather, the purpose is to enable applications to take advantage of such descriptions and formats by encapsulating any of these as NDEF messages and records.

The data (payload) length is an unsigned integer indicating the number of octets in the payload. A compact, short-record layout is provided for very small payloads. The optional payload identifier enables association of multiple data (payloads) and cross-referencing between them. NDEF payloads may include nested NDEF messages or chains of linked chunks of unknown length at the time the data is generated.

- BRIEFLY HOW NFC RADIOS AND MODULES WORK

NFC tags differ from many other RFID tags mainly because of the signal range of the NFC transceivers. Some RFID tags can be read from 100 meters, which is the case when you drive a car through a toll ring with an RFID tag. Such long range wireless tag needs an embedded battery to be able to broadcast the identifier/ data back to the reader. However, because the signal reach of NFC tags and devices is only a few centimetres, the actual NFC tags require no battery. Instead, the active NFC device activates the passive NFC tag with an electro-magnetic field. This field is sufficient to power the NFC chip and drive the data exchange.

- PLANNED OPERATING MODES FOR THE WEBINOS NFC

NFC devices can run in either reader-writer mode, peer-to-peer mode, or card emulation mode. These three modes are based on the ISO/IEC 18092 NFC IP-1 and ISO/IEC 14443 contactless smart card standards. Webinos aim to provide all three modes:

1. NFC reader/writer mode - the NFC device can read and write data to NFC tags. This is the original intention of the NFC technologies. We foresee that most NFC applications will be using this mode in the beginning. One use case is to hold a mobile towards a smart poster to obtain information about a concert.

2. NFC peer to peer mode - two NFC devices exchange data with each other when held close. The devices can connect and share any data/ files through the NFC Logical Link Control Protocol (LLCP). This capability was added to the NFC standard because of the introduction of NFC adapters to mobile phones.

3. NFC card emulation mode - an NFC device appears to another NFC device as a contactless smart card. It makes the NFC device appear as a contactless smart card for payment/ ticketing to other NFC devices. A contactless card an NFC tag with a tiny, secure application embedded on

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

it. Therefore, one needs to emulate cards through the execution of tiny, secure payment applications towards the NFC module.

The implementation priority for the webinos NFC module is in the order if the above three modes: We first will provide the NFC reader/writer mode, then we will deliver the NFC peer to peer mode. Finally, we aim to get the NFC card emulation mode implemented, however, at the moment this latter mode has a lower priority in webinos.

This specification provides a new DOM event ("nfctag") to discover when a nfc tag enters the field of the device.

Code example

```
 window.addEventListener("nfctag", nfcListener, true);

 function nfcListener(event)
 {
   var techSelected = null;
   var techSupported = event.tag.techList;
   if(techSupported.length > 0) {
     for (var index=0; index < techSupported.length; index++) {
       if(techSupported[index].type == techSupported[index].TECH_NDEF) {
         techSelected = techSupported[index];
       }
     }
   }
   if(techSelected) {
     var ndefMessage = techSelected.readCachedNdefMessage();
     for (var index=0; index < ndefMessage.ndefRecords.length) {
       if(ndefMessage.ndefRecords[index].type = ndefMessage.NDEFRECTYPE_URI)
                          alert("uri                                              found:
"+ndefMessage.ndefRecords[index].textPayload);
     }
   }
 }
```

2. Interfaces

2.1. PendingOperation

Definition of pending op.

```
        [NoInterfaceObject] interface PendingOperation {
                boolean cancel();
        };
```

Methods

cancel

Cancel the async op.

Signature

```
boolean cancel();
```

2.2. NFCTag

NFC tag event.

```
        interface NFCTag : Event {
                readonly attribute ByteArray tagId;
                readonly attribute NFCTagTechnologyArray techList;
                void initNFCTagEvent(in DOMString type,
                                     in boolean bubbles,
                                     in boolean cancelable,
                                     in ByteArray tagId,
                                     in unsigned short technologyType,
                                     in unsigned short ndefType,
                                     in unsigned short ndefRecType,
                                     in DOMString? ndefRecordTextPayload,
                                     in optional ByteArray ndefRecordBinaryPayload
                                     );
        };
```

Attributes

`readonly ByteArray tagId`

The identifier of the tag.

This attribute is readonly.

`readonly NFCTagTechnologyArray techList`

The list of technologies supported by the tag.

This attribute is readonly.

Methods

`initNFCTagEvent`

Method to set initial values of NFCTag event.

Signature

```
void initNFCTagEvent(in DOMString type, in boolean bubbles, in boolean cancelable, in ByteArray tagId, in unsigned short technologyType, in unsigned short ndefType, in unsigned short ndefRecType, in DOMString? ndefRecordTextPayload, in optional ByteArray ndefRecordBinaryPayload);
```

The initNFCTagEvent() method must initialize the event in a manner analogous to the initEvent() method in http://www.w3.org/TR/2010/WD-DOM-Level-3-Events-20100907/. The method can for example be used with document.createEvent() and EventTarget.dispatchEvent() to simulate a specific event. At the moment it should simulate a read only ndef Tag containing a ndefMessage that includes a single ndefRecord.

Parameters

- **type**
    - **Optional:** No.

- **Nullable**: No
- **Type:** DOMString
- **Description:** Event type i.e. 'sensor'

- **bubbles**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event bubbles
- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event cancelable
- **tagId**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ByteArray
  - **Description:** id of the tag
- **technologyType**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** unsigned short
  - **Description:** type of technology supported by the tag; at the moment it must be ndef.
- **ndefType**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** unsigned short
  - **Description:** type of ndef supported (see constants defined in NFCTagTechnologyNdef)

- **ndefRecType**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** unsigned short
  - **Description:** type of ndef record (see constants defined in NdefMessage)
- **ndefRecordTextPayload**
  - **Optional:** No.
  - **Nullable**: Yes
  - **Type:** DOMString
  - **Description:** textual payload of the ndef record
- **ndefRecordBinaryPayload**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** ByteArray
  - **Description:** binary payload of the ndef record

2.3. NFCTagTechnology

NFC technology.

```
        [NoInterfaceObject] interface NFCTagTechnology {
                const unsigned short TECH_OTHERS = 0;
                const unsigned short TECH_NFCA = 1;
                const unsigned short TECH_NFCB = 2;
                const unsigned short TECH_NFCF = 3;
                const unsigned short TECH_NFCV = 4;
                const unsigned short TECH_ISODEP = 5;
                const unsigned short TECH_NDEF = 6;
                readonly attribute unsigned short type;
                readonly attribute boolean isConnected;
                void connect();
                void close();
        };
```

Constants

`unsigned short TECH_OTHERS`

Constant identifying a non supported technology.

`unsigned short TECH_NFCA`

Constant identifying a NfcA technology.

`unsigned short TECH_NFCB`

Constant identifying a NfcB technology.

`unsigned short TECH_NFCF`

Constant identifying a NfcF technology.

`unsigned short TECH_NFCV`

Constant identifying a NfcV technology.

`unsigned short TECH_ISODEP`

Constant identifying a IsoDep technology.

`unsigned short TECH_NDEF`

Constant identifying a Ndef technology.

Attributes

`readonly unsigned short type`

Type of technology.

This attribute is readonly.

`readonly boolean isConnected`

Attribute indicating if the tag is connected or not.

This attribute is readonly.

Methods

`connect`

Connects to the tag.

Signature

```
void connect();
```

`close`

Closes connection to the tag.

Signature

```
void close();
```

2.4. NFCTagTechnologyNdef

Ndef technology.

```
[NoInterfaceObject] interface NFCTagTechnologyNdef : NFCTagTechnology {
        const unsigned short NDEFTYPE_OTHERS = 0;
        const unsigned short NDEFTYPE_NFCFORUMTYPE1 = 1;
        const unsigned short NDEFTYPE_NFCFORUMTYPE2 = 2;
```

```
                const unsigned short NDEFTYPE_NFCFORUMTYPE3 = 3;
                const unsigned short NDEFTYPE_NFCFORUMTYPE4 = 4;
                const unsigned short NDEFTYPE_MIFARECLASSIC = 5;
                readonly attribute unsigned short ndefType;
                readonly attribute boolean isWritable;
                readonly attribute unsigned long maxNdefMessageSize;
                PendingOperation  makeReadOnly(in  SuccessCallback  successCallback,  in
optional ErrorCallback? errorCallback)
                        raises(NfcException);
                NdefMessage readCachedNdefMessage();
                PendingOperation        readNdefMessage(in        NdefSuccessCallback
successCallback, in optional ErrorCallback? errorCallback)
                        raises(NfcException);
                PendingOperation  writeNdefMessage(in  SuccessCallback  successCallback,
in ErrorCallback? errorCallback, NdefMessage message)
                        raises(NfcException);
                NdefMessage createNdefMessage();
        };
```

Code example

```
 window.addEventListener("nfctag", nfcListener, true);

 function nfcListener(event)
 {
   var techSelected = null;
   var techSupported = event.tag.techList;
   if(techSupported.length > 0) {
     for (var index=0; index < techSupported.length; index++) {
       if(techSupported[index].type == techSupported[index].TECH_NDEF) {
         techSelected = techSupported[index];
       }
     }
   }
   // write to the tag
   if(techSelected) {
     var newMsg = techSelected.createNdefMessage();
     newMsg.addTextNdefRecord(newMsg.NDEFRECTYPE_URI, "http://webinos.org");
     techSelected.writeNdefMessage(wSuccess, wError, newMsg);
   }
 }

 function wSuccess()
 {
   alert("write successfull");
 }

 function wError()
 {
   alert("write error");
 }
```

 Constants

`unsigned short NDEFTYPE_OTHERS`

Constant identifying a non supported Ndef format.

`unsigned short NDEFTYPE_NFCFORUMTYPE1`

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Constant identifying a Nfc forum type 1 Ndef tag.

`unsigned short NDEFTYPE_NFCFORUMTYPE2`

Constant identifying a Nfc forum type 2 Ndef tag.

`unsigned short NDEFTYPE_NFCFORUMTYPE3`

Constant identifying a Nfc forum type 3 Ndef tag.

`unsigned short NDEFTYPE_NFCFORUMTYPE4`

Constant identifying a Nfc forum type 4 Ndef tag.

`unsigned short NDEFTYPE_MIFARECLASSIC`

Constant identifying a Mifare classic Ndef formatted tag.

Attributes

`readonly unsigned short ndefType`

Attribute indicating the type of Ndef tag.

This attribute is readonly.

`readonly boolean isWritable`

Attribute indicating if the tag is writable or not.

This attribute is readonly.

`readonly unsigned long maxNdefMessageSize`

Attribute indicating the maximum size of Ndef messages.

This attribute is readonly.

Methods

`makeReadOnly`

This method makes a tag read-only.

Signature

```
PendingOperation makeReadOnly(in SuccessCallback successCallback, in
optional ErrorCallback? errorCallback);
```

When the operation is fully completed the onsuccess method of the successCallback is called. Otherwise, the errorCallback will be invoked with an appropriate error code amongst the following:

- IO_ERR: if the write operation fails.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** SuccessCallback
  - **Description:** function to be invoked in case of success.
- **errorCallback**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** ErrorCallback
  - **Description:** function to be invoked in case of failure.

Return value

PendingOperation to cancel the asynchronous call

Exceptions

- NfcException:

  with error code INVALID_ARGUMENT_ERR if parameters are of the wrong type

`readCachedNdefMessage`

Retrieves the Ndef message received at discovery time.

Signature

```
NdefMessage readCachedNdefMessage();
```

Return value

The Ndef message received at discovery time.

`readNdefMessage`

This method reads a Ndef message.

Signature

```
PendingOperation         readNdefMessage(in         NdefSuccessCallback
successCallback, in optional ErrorCallback? errorCallback);
```

When the operation is fully completed the onsuccess method of the successCallback is called. Otherwise, the errorCallback will be invoked with an appropriate error code amongst the following:

- IO_ERR: if the read operation fails.

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** NdefSuccessCallback
  - **Description:** function to be invoked in case of success.
- **errorCallback**
  - **Optional:** Yes.
  - **Nullable**: Yes
  - **Type:** ErrorCallback
  - **Description:** function to be invoked in case of failure.

Return value

PendingOperation to cancel the asynchronous call

Exceptions

- NfcException:

  with error code INVALID_ARGUMENT_ERR if parameters are of the wrong type

`writeNdefMessage`

Writes a Ndef message to the tag.

Signature

```
PendingOperation writeNdefMessage(in SuccessCallback successCallback,
in ErrorCallback? errorCallback, NdefMessage message);
```

When the operation is fully completed the onsuccess method of the successCallback is called. Otherwise, the errorCallback will be invoked with an appropriate error code amongst the following:

- IO_ERR: if the write operation fails.

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No

- **Type:** SuccessCallback
- **Description:** function to be invoked in case of success.

- **errorCallback**
    - **Optional:** No.
    - **Nullable**: Yes
    - **Type:** ErrorCallback
    - **Description:** function to be invoked in case of failure.
- **message**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** NdefMessage
    - **Description:** The message to be written.

Return value

PendingOperation to cancel the asynchronous call

Exceptions

- NfcException:

  with error code INVALID_ARGUMENT_ERR if parameters are of the wrong type

`createNdefMessage`

Create a new Ndef message.

Signature

```
NdefMessage createNdefMessage();
```

Return value

The new Ndef message; it is empty, that is does not contain ndef records.

2.5. NdefMessage

Ndef message.

```
[NoInterfaceObject] interface NdefMessage {
        const unsigned short NDEFRECTYPE_UNKNOWN = 0;
        const unsigned short NDEFRECTYPE_URI = 1;
        const unsigned short NDEFRECTYPE_MEDIA = 2;
        const unsigned short NDEFRECTYPE_EMPTY = 3;
        const unsigned short NDEFRECTYPE_RTD = 4;
        const unsigned short NDEFRECTYPE_EXTERNALRTD = 5;
```

```
                readonly attribute NdefRecordArray ndefRecords;
                void addTextNdefRecord(in unsigned short type, in optional DOMString
payload)
                        raises(NfcException);
                void addBinaryNdefRecord(in unsigned short type, in ByteArray payload)
                        raises(NfcException);
        };
```

Constants

`unsigned short NDEFRECTYPE_UNKNOWN`

Constant identifying an unknown Ndef record type.

`unsigned short NDEFRECTYPE_URI`

Constant identifying a uri Ndef record type.

`unsigned short NDEFRECTYPE_MEDIA`

Constant identifying a media Ndef record type.

`unsigned short NDEFRECTYPE_EMPTY`

Constant identifying an empty Ndef record type.

`unsigned short NDEFRECTYPE_RTD`

Constant identifying a RTD Ndef record type.

`unsigned short NDEFRECTYPE_EXTERNALRTD`

Constant identifying an external RTD Ndef record type.

Attributes

`readonly NdefRecordArray ndefRecords`

List of Ndef records.

This attribute is readonly.

Methods

`addTextNdefRecord`

Adds a text record to the Ndef message.

Signature

```
void addTextNdefRecord(in unsigned short type, in optional DOMString
payload);
```

TODO add exceptions

Parameters

- **type**

- **Optional:** No.
- **Nullable**: No
- **Type:** unsigned short
- **Description:** The type of the ndef record

- **payload**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** The text payload of the record

Exceptions

- NfcException:

  with error code INVALID_ARGUMENT_ERR if parameters are of the wrong type

`addBinaryNdefRecord`

Adds a binary record to the Ndef message.

Signature

```
void addBinaryNdefRecord(in unsigned short type, in ByteArray payload);
```

TODO add exceptions

Parameters

- **type**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** unsigned short
  - **Description:** The type of the ndef record
- **payload**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ByteArray
  - **Description:** The binary payload of the record

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Exceptions

- NfcException:

  with error code INVALID_ARGUMENT_ERR if parameters are of the wrong type

2.6. NdefRecord

Ndef record.

```
[NoInterfaceObject] interface NdefRecord {
        readonly attribute unsigned short type;
        readonly attribute DOMString textPayload;
        readonly attribute ByteArray binaryPayload;
};
```

Chunk of records are assembled by the underlying implementation and returned as a single Ndef record.

Attributes

`readonly unsigned short type`

The type of the record.

This attribute is readonly.

`readonly DOMString textPayload`

The textual payload of the record.

This attribute is readonly.

`readonly ByteArray binaryPayload`

The binary payload of the record.

This attribute is readonly.

2.7. NfcError

Interface for reporting Nfc specific errors.

```
[NoInterfaceObject] interface NfcError {
        readonly attribute unsigned short code;
        readonly attribute DOMString message;
        const unsigned short UNKNOWN_ERR = 0;
        const unsigned short IO_ERR = 1;
};
```

Constants

`unsigned short UNKNOWN_ERR`

Unknown error.

`unsigned short IO_ERR`

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

I/O error.

Attributes

`readonly unsigned short code`

Code assigned when an error has occurred in Nfc API processing.

This attribute is readonly.

`readonly DOMString message`

Human readable message assigned when an error has occurred in Nfc API processing.

This attribute is readonly.

2.8. SuccessCallback

Callback to be invoked in case of success.

```
[Callback=FunctionOnly, NoInterfaceObject] interface SuccessCallback {
        void onsuccess();
};
```

2.9. ErrorCallback

Callback to be invoked when an error occurs.

```
[Callback=FunctionOnly, NoInterfaceObject] interface ErrorCallback {
        void onError(in NfcError error);
};
```

2.10. NdefSuccessCallback

Callback to be invoked when reading a Ndef message.

```
[Callback=FunctionOnly, NoInterfaceObject] interface NdefSuccessCallback {
        void onSuccess(in NdefMessage obj);
};
```

3. Type Definitions

3.1. NFCTagTechnologyArray

Array of NFCTagTechnology.

```
typedef NFCTagTechnology[]      NFCTagTechnologyArray;
```

3.2. NdefRecordArray

Array of NdefRecord.

```
typedef NdefRecord[]      NdefRecordArray;
```

3.3. ByteArray

Array of 8-bit unsigned integer values.

```
typedef octet[] ByteArray;
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

4. Exceptions

4.1. NfcException

```
        exception NfcException {
                const unsigned short UNKNOWN_ERR = 0;
                const unsigned short INVALID_ARGUMENT_ERR       = 1;
                unsigned short code;
                DOMString message;
        };
```

Field

unsigned short code

Error code.

DOMString message

Error message.

5. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/nfc

Acccess to all the module. This feature provides access to the whole API.

http://webinos.org/api/nfc.read

Acccess to all the module except write operations.

6. Full WebIDL

```
module nfc {
        typedef NFCTagTechnology[]      NFCTagTechnologyArray;
        typedef NdefRecord[]      NdefRecordArray;
        typedef octet[] ByteArray;

        [NoInterfaceObject] interface PendingOperation {
                boolean cancel();
        };

        interface NFCTag : Event {
                readonly attribute ByteArray tagId;
                readonly attribute NFCTagTechnologyArray techList;
                void initNFCTagEvent(in DOMString type,
                                     in boolean bubbles,
                                     in boolean cancelable,
                                     in ByteArray tagId,
                                     in unsigned short technologyType,
                                     in unsigned short ndefType,
                                     in unsigned short ndefRecType,
                                     in DOMString? ndefRecordTextPayload,
                                     in optional ByteArray ndefRecordBinaryPayload
```

```
                                  );
                        };

        [NoInterfaceObject] interface NFCTagTechnology {
                const unsigned short TECH_OTHERS = 0;
                const unsigned short TECH_NFCA = 1;
                const unsigned short TECH_NFCB = 2;
                const unsigned short TECH_NFCF = 3;
                const unsigned short TECH_NFCV = 4;
                const unsigned short TECH_ISODEP = 5;
                const unsigned short TECH_NDEF = 6;
                readonly attribute unsigned short type;
                readonly attribute boolean isConnected;
                void connect();
                void close();
        };

        [NoInterfaceObject] interface NFCTagTechnologyNdef : NFCTagTechnology {
                const unsigned short NDEFTYPE_OTHERS = 0;
                const unsigned short NDEFTYPE_NFCFORUMTYPE1 = 1;
                const unsigned short NDEFTYPE_NFCFORUMTYPE2 = 2;
                const unsigned short NDEFTYPE_NFCFORUMTYPE3 = 3;
                const unsigned short NDEFTYPE_NFCFORUMTYPE4 = 4;
                const unsigned short NDEFTYPE_MIFARECLASSIC = 5;
                readonly attribute unsigned short ndefType;
                readonly attribute boolean isWritable;
                readonly attribute unsigned long maxNdefMessageSize;

                PendingOperation makeReadOnly(in SuccessCallback successCallback, in
optional ErrorCallback? errorCallback)
                        raises(NfcException);
                NdefMessage readCachedNdefMessage();
                PendingOperation        readNdefMessage(in        NdefSuccessCallback
successCallback, in optional ErrorCallback? errorCallback)
                        raises(NfcException);
                PendingOperation writeNdefMessage(in SuccessCallback successCallback,
in ErrorCallback? errorCallback, NdefMessage message)
                        raises(NfcException);
                NdefMessage createNdefMessage();
        };

        [NoInterfaceObject] interface NdefMessage {
                const unsigned short NDEFRECTYPE_UNKNOWN = 0;
                const unsigned short NDEFRECTYPE_URI = 1;
                const unsigned short NDEFRECTYPE_MEDIA = 2;
                const unsigned short NDEFRECTYPE_EMPTY = 3;
                const unsigned short NDEFRECTYPE_RTD = 4;
                const unsigned short NDEFRECTYPE_EXTERNALRTD = 5;
                readonly attribute NdefRecordArray ndefRecords;
                void addTextNdefRecord(in unsigned short type, in optional DOMString
payload)
                        raises(NfcException);
                void addBinaryNdefRecord(in unsigned short type, in ByteArray payload)
                        raises(NfcException);
        };

                [NoInterfaceObject] interface NdefRecord {
                        readonly attribute unsigned short type;
                        readonly attribute DOMString textPayload;
                        readonly attribute ByteArray binaryPayload;
                };

        [NoInterfaceObject] interface NfcError {
                readonly attribute unsigned short code;
                readonly attribute DOMString message;
                const unsigned short UNKNOWN_ERR = 0;
                const unsigned short IO_ERR = 1;
        };

        exception NfcException {
                const unsigned short UNKNOWN_ERR = 0;
                const unsigned short INVALID_ARGUMENT_ERR       = 1;
                unsigned short code;
                DOMString message;
        };

        [Callback=FunctionOnly, NoInterfaceObject] interface SuccessCallback {
                void onsuccess();
        };

        [Callback=FunctionOnly, NoInterfaceObject] interface ErrorCallback {
                void onError(in NfcError error);
        };

        [Callback=FunctionOnly, NoInterfaceObject] interface NdefSuccessCallback {
                void onSuccess(in NdefMessage obj);
        };
};
```

APIs: The payment module

Webinos API Specifications

29 Jun 2011

Authors

- Christian Fuhrhop <christian.fuhrhop@fokus.fraunhofer.de>

Abstract

Interface for Payment functions.

Summary of Methods

Interface	Method
WebinosPayment	
Payment	PendingOperation createShoppingBasket(SuccessShoppingBasketCallback successCallback, PaymentErrorCB errorCallback, DOMString serviceProviderID, DOMString customerID, DOMString shopID)
ShoppingBasket	PendingOperation addItem(PaymentSuccessCB successCallback, PaymentErrorCB errorCallback, ShoppingItem item) PendingOperation update(PaymentSuccessCB successCallback, PaymentErrorCB errorCallback) PendingOperation checkout(PaymentSuccessCB successCallback, PaymentErrorCB errorCallback) void release()
ShoppingItem	
SuccessShoppingBasketCallback	void onSuccess(ShoppingBasket basket)

Interface	Method
PaymentSuccessCB	void onSuccess()
PaymentErrorCB	void onError(PaymentError error)
PendingOperation	void cancel()
PaymentError	

1. Introduction

This API provides generic shopping basket functionality to provide in-app payment.

It is not linked to a specific payment service provider and is designed to be sufficiently generic to be mapable to various payment services like GSMA OneAPI, Andoid Payment API or PayPal.

2. Interfaces

2.1. WebinosPayment

The WebinosPayment interface describes the part of the payment API accessible through the webinos object.

```
   [NoInterfaceObject] interface WebinosPayment {
      readonly attribute Payment payment;
   };
     webinoscore::Webinos implements WebinosPayment;
```

Attributes

`readonly Payment payment`

webinos.payment object.

This attribute is readonly.

2.2. Payment

The Payment interface

```
[NoInterfaceObject] interface Payment {
        PendingOperation     createShoppingBasket(in     SuccessShoppingBasketCallback
successCallback, in PaymentErrorCB errorCallback,
        in DOMString serviceProviderID, in DOMString customerID, in DOMString shopID)
        raises(PaymentException);
  };
```

The Payment interface provides access to payment functionality.

The API supports creation of a shopping basket, adding items to the shopping basket, proceeding to checkout and releasing the shopping basket.

This essentially echoes the usual 'shopping basket' system used on many web sites.

The code example below refunds the user for a returned CD and charges for the deluxe edition of that CD, demonstarting charging and refunding payments.

Code example

```
        webinons.payment.createShoppingBasket(openBasketSuccess,       paymentFailure,
"PayPal", "mymail@provider.com", "ShopName12345");
        var myBasket = null;

        // Define the openBasketSuccess success callback.
        function openBasketSuccess(basket) {
                alert("Shopping basket was opened successfully");
                myBasket = basket;
                // refound for a CD
                myBasket.addItem(CD2346278, paymentFailure,
                    {  productID: 'DCD2346233',
                       description: 'Best of Ladytron 00-10 by Ladytron (Audio CD -
2011)',
                       currency: 'EUR',
                       itemPrice: -14.99,
                       itemCount: 1}
        }

        // Define the refundItemSuccess success callback.
        function refundSuccess() {
                alert("Adding of refunding item was handled successfully");
                // charge for the deluxe CD
                myBasket.addItem(addItemSuccess, paymentFailure,
                    {  productID: 'DCD2346233',
                       description:  'Best  of  Ladytron  00-10  (Deluxe  Edition)  by
Ladytron (Audio CD - 2011)',
                       currency: 'EUR',
                       itemPrice: 17.98,
                       itemCount: 1}
        }

        // Define the addItemSuccess success callback.
        function addItemSuccess() {
                alert("Adding of new item was handled successfully");
                // now close the bill and perform the actual payment
                myBasket.update(updateSuccess, paymentFailure);
        }

        // Define the updateSuccess success callback.
        function updateSuccess() {
                alert("Total    amount    is:    "myBasket.totalAmount+"    Tax    is
"+myBasket.tax);
                // now close the bill and perform the actual payment
                myBasket.checkout(checkoutSuccess, paymentFailure);
        }

        // Define the checkoutSuccess success callback.
        function checkoutSuccess() {
                alert("Checkout handled successfully - payment was performed.");
                 if (myBasket != null) myBasket.release();
        }
```

```
// Define the paymentFailure failure callback.
function paymentFailure(e) {
        alert("Failure occured during payment.");
         if (myBasket != null) myBasket.release();
}
```

Methods

`createShoppingBasket`

Creates a shopping basket

Signature

```
PendingOperation createShoppingBasket(in SuccessShoppingBasketCallback successCallback, in PaymentErrorCB errorCallback, in DOMString serviceProviderID, in DOMString customerID, in DOMString shopID);
```

Parameters

- **successCallback**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** SuccessShoppingBasketCallback
    - **Description:** Callback issued when the shopping basket is created
- **errorCallback**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** PaymentErrorCB
    - **Description:** Callback issued if an error occurs during the creation of the shopping basket
- **serviceProviderID**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** is the name of the payment provider to be used
- **customerID**

  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** is identification of the person making the payment as known to the payment provider

- **shopID**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** is the identification of the shop the payment is made to

Return value

PendingOperation enabling the requester to cancel this request.

Exceptions

- PaymentException:

  INVALID_ARGUMENT_ERROR if an invalid argument is passed

2.3. ShoppingBasket

The ShoppingBasket interface provides access to a shopping basket

```
[NoInterfaceObject] interface ShoppingBasket {
        readonly attribute ShoppingItem[] items;
        readonly attribute ShoppingItem[] extras;
        readonly attribute float totalBill;

        PendingOperation    addItem(in    PaymentSuccessCB    successCallback,    in
PaymentErrorCB errorCallback, in ShoppingItem item)
        raises(PaymentException);

        PendingOperation update(in PaymentSuccessCB successCallback, in PaymentErrorCB
errorCallback)
        raises(PaymentException);

        PendingOperation    checkout(in    PaymentSuccessCB    successCallback,    in
PaymentErrorCB errorCallback)
        raises(PaymentException);

        void release();
  };
```

The shopping basket represents a current payment action and allows to add a number of items to the basket before proceeding to checkout.

Attributes

`readonly ShoppingItem [] items`

List of items currently in the shopping basket.

These are the items that have been added with addItem.

No exceptions

This attribute is readonly.

`readonly ShoppingItem [] extras`

Automatically generated extra items, typically rebates, taxes and shipping costs.

These items are automatically added to the shopping basket by update() (or after the addition of an item to the basket).

These items can contain such 'virtual' items as payback schemes, rebates, taxes, shipping costs and other items that are calculated on the basis of the regular items added.

No exceptions

This attribute is readonly.

`readonly float totalBill`

The total amount that will be charged to the user on checkout.

Will be updated by update(), may be updated by addItem().

No exceptions

This attribute is readonly.

Methods

`addItem`

Adds an item to a shopping basket.

Signature

```
PendingOperation addItem(in PaymentSuccessCB successCallback, in PaymentErrorCB errorCallback, in ShoppingItem item);
```

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** PaymentSuccessCB

  - **Description:** Callback issued when the adding of the item to the shopping basket is correctly finished.

- **errorCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** PaymentErrorCB
  - **Description:** Callback issued if an error occurs during adding the amount

- **item**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ShoppingItem
  - **Description:** the item to purchase

Return value

PendingOperation enabling the requester to cancel this request.

Exceptions

- PaymentException:

  INVALID_ARGUMENT_ERROR if an invalid argument is passed

`update`

Updates the shopping basket

Signature

```
PendingOperation update(in PaymentSuccessCB successCallback, in
PaymentErrorCB errorCallback);
```

The update function updates the values in the shopping baskets, based on the added items. Such updates may include taxes, calculating the total amount, shipping costs or rebate calculations.

While this, preferably, is internally updated after the adding of each item, such an update might require communication with the payment service provider and it might be undesireable in specific implementations to perform such a query after each individual item, so a specifc update function is provided to force such an update.

The checkout function will always perform an update internally before payment.

Parameters

- **successCallback**

- **Optional:** No.
- **Nullable**: No
- **Type:** PaymentSuccessCB
- **Description:** Callback issued when the update is performed

- **errorCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** PaymentErrorCB
  - **Description:** Callback issued if an error occurs during update

Return value

PendingOperation enabling the requester to cancel this request.

Exceptions

- PaymentException:

  INVALID_ARGUMENT_ERROR if an invalid argument is passed

`checkout`

Performs the checkout of the shopping basket.

Signature

```
PendingOperation checkout(in PaymentSuccessCB successCallback, in
PaymentErrorCB errorCallback);
```

The items in the shopping basket will be charged to the shopper.

Depending on the implementation of the actual payment service, this function might cause the checkout screen of the payment service provider to be displayed.

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** PaymentSuccessCB
  - **Description:** Callback issued when the checkout is performed and payment is made
- **errorCallback**

- **Optional:** No.
- **Nullable**: No
- **Type:** PaymentErrorCB
- **Description:** Callback issued if an error occurs during adding the amount

Return value

PendingOperation enabling the requester to cancel this request.

Exceptions

- PaymentException:

  INVALID_ARGUMENT_ERROR if an invalid argument is passed

`release`

Releases a shopping basket.

Signature

```
void release();
```

The current shopping basket will be released.

If no checkout has been performed, the initiated shopping transaction will be cancelled.

Return value

void

2.4. ShoppingItem

The ShoppingItem captures the attributes of a single shopping product

```
[NoInterfaceObject] interface ShoppingItem {
      attribute DOMString productID;
      attribute DOMString description;
      attribute DOMString currency;
      attribute float itemPrice;
      attribute unsigned long itemCount;
      readonly attribute unsigned long itemsPrice;
 };
```

The shopping basket represents a current payment action and allows to add a number of items to the basket before proceeding to checkout.

Attributes

`DOMString productID`

An id that allows the shop to identify the purchased item

No exceptions

`DOMString description`

A human-readable text to appear on the bill, so the user can easily see what they bought.

No exceptions

`DOMString currency`

The 3-figure code as per ISO 4217.

No exceptions

`float itemPrice`

The price per individual item in the currency given above, a negative number represents a refund.

No exceptions

`unsigned long itemCount`

The number of identical items purchased

No exceptions

`readonly unsigned long itemsPrice`

Price for all products in this shopping item.

Typically this is itemPrice*itemCount, but special '3 for 2' rebates might apply.

Updated by the shopping basket update function.

No exceptions

This attribute is readonly.

2.5. SuccessShoppingBasketCallback

Callback for successful creation of a shopping basket

```
[Callback=FunctionOnly, NoInterfaceObject]
interface SuccessShoppingBasketCallback {
    void onSuccess  (ShoppingBasket basket);
};
```

Methods

`onSuccess`

Callback for successful creation of a shopping basket

Signature

```
void onSuccess(ShoppingBasket basket);
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Parameters

- **basket**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** ShoppingBasket
    - **Description:** The shopping basket to which items can be added.

Return value
void

2.6. PaymentSuccessCB

Callback for successful payment related functions

```
[Callback=FunctionOnly, NoInterfaceObject]
interface PaymentSuccessCB {
    void onSuccess  ();
};
```

Methods
`onSuccess`

Callback for successful of payment related functions

Signature
`void onSuccess();`

Return value
void

2.7. PaymentErrorCB

Callback for errors during payment related functions

```
[Callback=FunctionOnly, NoInterfaceObject]
interface PaymentErrorCB {
    void onError (in PaymentError error);
};
```

Methods
`onError`

Callback for errors during payment related functions

Signature
`void onError(in PaymentError error);`

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** PaymentError
  - **Description:** The Payment API related error object of an unsuccessful asynchronous operation.

Return value

void

2.8. PendingOperation

The PendingOperation interface

```
[NoInterfaceObject] interface PendingOperation {
   void cancel ();
};
```

The PendingOperation interface describes objects that are returned by asynchronous methods that are cancellable. It makes it possible to bring these operations to a stop if they haven't produced a result within a desired time or before a given event, thereby possibly reclaiming resources.

Methods

`cancel`

Method Cancel

Signature

```
void cancel();
```

Cancel the pending asynchronous operation. When this method is called, the user agent must immediately bring the operation to a stop and return. No success or error callback for the pending operation will be invoked.

2.9. PaymentError

Payment specific errors.

```
interface PaymentError {
        const unsigned short PAYMENT_SHOPPING_BASKET_OPEN_ERROR = 1;
        const unsigned short PAYMENT_SHOPPING_BASKET_NOT_OPEN_ERROR = 2;
        const unsigned short PAYMENT_CHARGE_FAILED = 3;
        const unsigned short PAYMENT_REFUND_NOT_SUPPORTED = 4;
        const unsigned short PAYMENT_REFUND_FAILED = 5;
        const unsigned short PAYMENT_CHARGEABLE_EXCEEDED = 6;
        const unsigned short PAYMENT_AUTHENTICATION_FAILED = 7;
     readonly attribute unsigned short code;
     readonly attribute DOMString message;
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
  };
```

The PaymentError interface encapsulates all errors in the manipulation of payments objects in the Payment API.

Constants

`unsigned short PAYMENT_SHOPPING_BASKET_OPEN_ERROR`

Bill is already open

`unsigned short PAYMENT_SHOPPING_BASKET_NOT_OPEN_ERROR`

Bill is not open

`unsigned short PAYMENT_CHARGE_FAILED`

Charging operation failed, the charge was not applied

`unsigned short PAYMENT_REFUND_NOT_SUPPORTED`

Refunds not supported

`unsigned short PAYMENT_REFUND_FAILED`

Refund failed

`unsigned short PAYMENT_CHARGEABLE_EXCEEDED`

Chargeable amount exceeded

`unsigned short PAYMENT_AUTHENTICATION_FAILED`

Chargeable Authentication failed. Payment credentials are incorrect.

Attributes

`readonly unsigned short code`

An error code assigned by an implementation when an error has occurred in Payment processing.

No exceptions.

This attribute is readonly.

`readonly DOMString message`

A text describing an error occuring in the Payment in human readable form.

No exceptions.

This attribute is readonly.

3. Exceptions

3.1. PaymentException

Payment API specific errors.

```
         exception PaymentException {
                const unsigned short INVALID_ARGUMENT_ERROR = 1;
        unsigned short code;
        DOMString message;
  };
```

The PaymentException interface encapsulates all errors in calling of the Payment API.

Field

`unsigned short code`

An error code assigned by an implementation when an error has occurred in Payment API processing.

`DOMString message`

A text describing an error occuring in the Payment API in human readable form.

4. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/payment

Identifies all payment interactions.

5. Full WebIDL

```
module payment {
   [NoInterfaceObject] interface WebinosPayment {
      readonly attribute Payment payment;
   };

[NoInterfaceObject] interface Payment {
        PendingOperation createShoppingBasket(in SuccessShoppingBasketCallback
successCallback, in PaymentErrorCB errorCallback,
        in DOMString serviceProviderID, in DOMString customerID, in DOMString shopID)
        raises(PaymentException);
  };

[NoInterfaceObject] interface ShoppingBasket {
        readonly attribute ShoppingItem[] items;
        readonly attribute ShoppingItem[] extras;
        readonly attribute float totalBill;

        PendingOperation addItem(in PaymentSuccessCB successCallback, in
PaymentErrorCB errorCallback, in ShoppingItem item)
        raises(PaymentException);
        PendingOperation update(in PaymentSuccessCB successCallback, in PaymentErrorCB
errorCallback)
        raises(PaymentException);
        PendingOperation    checkout(in    PaymentSuccessCB    successCallback,    in
PaymentErrorCB errorCallback)
        raises(PaymentException);
        void release();
  };

   [NoInterfaceObject] interface ShoppingItem {
         attribute DOMString productID;
         attribute DOMString description;
         attribute DOMString currency;
         attribute float itemPrice;
         attribute unsigned long itemCount;
         readonly attribute unsigned long itemsPrice;
    };

        [Callback=FunctionOnly, NoInterfaceObject]
        interface SuccessShoppingBasketCallback {
            void onSuccess  (ShoppingBasket basket);
        };

        [Callback=FunctionOnly, NoInterfaceObject]
        interface PaymentSuccessCB {
            void onSuccess  ();
        };

        [Callback=FunctionOnly, NoInterfaceObject]
        interface PaymentErrorCB {
            void onError (in PaymentError error);
        };

     [NoInterfaceObject] interface PendingOperation {
        void cancel ();
     };

        interface PaymentError {
                const unsigned short PAYMENT_SHOPPING_BASKET_OPEN_ERROR = 1;
                const unsigned short PAYMENT_SHOPPING_BASKET_NOT_OPEN_ERROR = 2;
                const unsigned short PAYMENT_CHARGE_FAILED = 3;
                const unsigned short PAYMENT_REFUND_NOT_SUPPORTED = 4;
                const unsigned short PAYMENT_REFUND_FAILED = 5;
                const unsigned short PAYMENT_CHARGEABLE_EXCEEDED = 6;
                const unsigned short PAYMENT_AUTHENTICATION_FAILED = 7;
             readonly attribute unsigned short code;
             readonly attribute DOMString message;
  };

         exception PaymentException {
                const unsigned short INVALID_ARGUMENT_ERROR = 1;
        unsigned short code;
        DOMString message;
  };
     webinoscore::Webinos implements WebinosPayment;

};
```

APIs: The sensors module

Webinos API Specifications

30 Jun 2011

Authors

- Claes Nilsson <claes1.nilsson@sonyericsson.com>

Abstract

The Generic Sensor API

Summary of Methods

Interface	Method
Sensor	PendingOp configureSensor(ConfigureSensorOptions options, ConfigureSensorCB successCB, SensorErrorCB errorCB)
ConfigureSensorCB	void onSuccess()
SensorErrorCB	void onErrror(SensorCBError error)
SensorCBError	
ConfigureSensorOptions	
PendingOp	void cancel()
SensorEvent	void initSensorEvent(DOMString type, boolean bubbles, boolean cancelable, DOMString sensorType, DOMString sensorId, unsigned short accuracy, unsigned short rate, boolean interrupt, float [] sensorValues)

1. Introduction

The Webinos Generic Sensor API provides web applications with an API to access data from sensors in the device, connected to the device or in another device.

The API is agnostic to underlying low level methods for sensor discovery and communication with sensors. However, the sensor API should be used in combination with the general Webinos service discovery methods findServices() and bind(). The sensors services can be located in the user's personal zone or be shared on the current network.

The API consists of two interfaces:
- A sensor interface that provides attributes for the sensors and a method to configure a selected sensor.
- A DOM level 3 event that provides sensor data.

Currently 5 different sensor types are defined but the API could easily be extended with additional sensor types.

This is an experimental API and security and privacy issues are not specifically addressed in the specification. If access to security or privacy sensitive sensors are provided the user agent must either acquire access permission through a user interface or control access through a prearranged trust relationship with users.

2. Interfaces

2.1. Sensor

This interface defines sensor properties. It is a sensor specific extension to the interface Service in the ServiceDiscovery module. The added attributes correspond to Android sensor API attributes.

```
    [NoInterfaceObject] interface Sensor : Service {
        readonly attribute float?          maximumRange;
        readonly attribute unsigned long?   minDelay;
        readonly attribute float?         power;
        readonly attribute float?         resolution;
        readonly attribute DOMString?      vendor;
        readonly attribute unsigned long?  version;

        PendingOp    configureSensor    (in    ConfigureSensorOptions    options,    in
ConfigureSensorCB successCB, in optional SensorErrorCB errorCB)
            raises (SensorException);
     };
```

Code example

```
        //  Handle  that  can  be  used  to  cancel  the  ongoing  asynchronous  discovery
process.
        var findHandle = 0;

        // Handle from service.bind.
        var sensorHandle = 0;

        // Array of found temperature sensors object.
        var availableTempSensors = {};

        // Callback method that display a list of found sensors in a selection list
        //  The  selection  list  is  dynamically  extended  every  time  a  new  sensor  is
discovered.
        function sensorFoundCB(sensor) {
                var selectlist = document.getElementById('sensorlist');
                var option = document.createElement('option');
                option.value = sensor.id;
                option.appendChild(document.createTextNode(sensor.displayName));
                availableTempSensors [sensor.id] = sensor;
                selectlist.appendChild(option);

        }


       // Callback when bind has been successfully executed on the service object. The
Sensor is authorized and ready to use
        function bindCB(mySensor) {

                alert('Sensor ' + mySensor.displayName + ' with ID: ' + mySensor.id +
' selected');


                // Configure the sensor.
                mySensor.configureSensor ( {timeout: 120, rate: SENSOR_DELAY_NORMAL,
interrupt: False},
                                           successHandler () {alert('Sensor ' +
mySensor.displayName + ' with ID: ' + mySensor.id +
                                                                   ' is configured')
},
                                           errorHandler (error) {alert('Sensor ' +
mySensor.displayName + ' with ID: ' + mySensor.id +
                                                                      ' configuration
failed' + ' with error: ' + error.message)} );

                // Start listening to sensor events and log values.
                mySensor.addEventListener('sensor', function (event) {
                          console.log(event.sensorValues[0]);

                          var temp = document.getElementById('temp');
                          temp.innerHTML = "Current temperature is: " +
event.sensorValues[0];
                }, true);
        }

       // Callback method that is invoked when user selects an option in the sensor
selection list
        function sensorSelected(sensor) {

                // Stops the findServices operation
                findHandle.cancel();

                // Binds to the sensor API to initiate an authorized objects used to
                // invoke services.
                sensorHandle = sensor.bind({onBind:bindCB});
        }

        // Get list of temperature sensors registered in the device through the
Service Discovery findServices() method
        findHandle                                                                  =
window.webinos.discovery.findServices({api:'http://webinos.org/api/sensors.temperature
'}, {onFound:sensorFoundCB});
```

```
        // Handle user selection of sensor
         var sensorlist = document.getElementById('sensorlist');
         sensorlist.addEventListener("change", function (e) {
                                var sensor = availableTempSensors[e.target.value];
                                if (sensor) {
                                    sensorSelected(sensor);
                                }
         }, false);
```

Attributes

`readonly float? maximumRange`

Max range of sensor in the sensors unit.

This attribute is readonly.

`readonly unsigned long? minDelay`

Min delay of sensor allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes.

This attribute is readonly.

`readonly float? power`

Power consumption of sensor in mA used by this sensor while in use.

This attribute is readonly.

`readonly float? resolution`

Resolution of the sensor in the sensors unit.

This attribute is readonly.

`readonly DOMString? vendor`

Vendor string of this sensor.

This attribute is readonly.

`readonly unsigned long? version`

Version of the sensors module.

This attribute is readonly.

Methods

`configureSensor`

Configures a sensor.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Signature

```
PendingOp configureSensor(in ConfigureSensorOptions options, in
ConfigureSensorCB successCB, in optional SensorErrorCB errorCB);
```

Question: Do we need the ability to specify high and low thresholds? This is for example not supported by Android sensor API.

Parameters

- **options**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ConfigureSensorOptions
  - **Description:** Sensor configuration options.
- **successCB**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ConfigureSensorCB
  - **Description:** Callback issued when sensor configuration succeeded.
- **errorCB**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** SensorErrorCB
  - **Description:** Callback issued if sensor configuration fails.

Return value

A pending operation object making it possible to cancel the configureSensor operation

Exceptions

- SensorException:

  with appropriate error code.

2.2. ConfigureSensorCB

ConfigureSensorCB interface definition

```
[Callback=FunctionOnly, NoInterfaceObject] interface ConfigureSensorCB {
   void onSuccess();
};
```

Methods

`onSuccess`

onSuccess The onSuccess method is called when configuration of a sensor succeeded.

Signature

```
void onSuccess();
```

2.3. SensorErrorCB

SensorErrorCB interface definition

```
[Callback=FunctionOnly, NoInterfaceObject] interface SensorErrorCB {
   void onErrror(in SensorCBError error);
};
```

Methods

`onErrror`

onError The onError method is called if an error occurs during the configureSensor() process.

Signature

```
void onErrror(in SensorCBError error);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** SensorCBError
  - **Description:** The error object of an unsuccessful configureSensor() asynchronous operation.

2.4. SensorCBError

SensorCBError interface definition

```
[NoInterfaceObject] interface SensorCBError {
   const unsigned short UNKNOWN_ERROR = 0;
   const unsigned short TIMEOUT_ERROR = 1;
   const unsigned short ILLEGAL_SENSOR_TYPE_ERROR = 2;
   const unsigned short SENSOR_TYPE_NOT_SUPPORTED_ERROR = 3;
   const unsigned short ILLEGAL_SENSOR_ID_ERROR = 4;
   const unsigned short OTHER_ILLEGAL_INPUT_ARGUMENT_ERROR = 5;
   const unsigned short REQUESTED_RATE_NOT_SUPPORTED_ERROR = 6;
   const unsigned short REQUESTED_INTERRUPTMODE_NOT_SUPPORTED_ERROR = 7;
   const unsigned short PERMISSION_DENIED_ERROR = 50;
   readonly attribute unsigned short code;
   readonly attribute DOMString message;
};
```

Constants

`unsigned short UNKNOWN_ERROR`

Uknown error

`unsigned short TIMEOUT_ERROR`

No success callback within timeout period.

`unsigned short ILLEGAL_SENSOR_TYPE_ERROR`

Illegal sensor type

`unsigned short SENSOR_TYPE_NOT_SUPPORTED_ERROR`

Illegal sensor type

`unsigned short ILLEGAL_SENSOR_ID_ERROR`

Illegal sensor id

`unsigned short OTHER_ILLEGAL_INPUT_ARGUMENT_ERROR`

Other illegal input arguments

`unsigned short REQUESTED_RATE_NOT_SUPPORTED_ERROR`

Sensor rate requested through configureSensor() not supported

`unsigned short REQUESTED_INTERRUPTMODE_NOT_SUPPORTED_ERROR`

Interrupt mode requested through configureSensor() not supported

`unsigned short PERMISSION_DENIED_ERROR`

Permission denied

Attributes

`readonly unsigned short code`

Error code assigned when an error has occurred in configureSensor() processing.

This attribute is readonly.

`readonly DOMString message`

Human readable message assigned when an error has occurred in configureSensor() processing.

This attribute is readonly.

2.5. ConfigureSensorOptions

ConfigureSensorOptions interface definition

```
    [NoInterfaceObject] interface ConfigureSensorOptions {
        const unsigned short INFINITE = 0;
        const unsigned short SENSOR_DELAY_FASTEST = 0;
        const unsigned short SENSOR_DELAY_GAME = 1;
        const unsigned short SENSOR_DELAY_UI = 2;
        const unsigned short SENSOR_DELAY_NORMAL = 3;
        attribute unsigned short timeout;
        attribute unsigned short rate;
        attribute boolean interrupt;
      };
```

Constants

`unsigned short INFINITE`

INFINITE Timeout Value

`unsigned short SENSOR_DELAY_FASTEST`

The sensor is reporting data as fast as possible (rate attribute).

`unsigned short SENSOR_DELAY_GAME`

The sensor is reporting data with a rate suitable for games (rate attribute).

`unsigned short SENSOR_DELAY_UI`

The sensor is reporting data with a rate suitable for user interface (rate attribute).

`unsigned short SENSOR_DELAY_NORMAL`

The sensor is reporting data with a normal rate, e.g. suitable for screen orientation changes (rate attribute).

Attributes

`unsigned short timeout`

A timeout value for when configureSensor() is canceled in seconds between 0-65535. Default value is 120 seconds.

`unsigned short rate`

The requested rate of the sensor data.

`boolean interrupt`

The requested Interrupt mode of the sensor.
False = INTERRUPT_DISABLED (events fired with a fixed time interval)
True = INTERRUPT_ENABLED (events fired when value changes)

2.6. PendingOp

The PendingOp interface

```
      [NoInterfaceObject] interface PendingOp {
         void cancel ();
      };
```

The PendingOp interface describes objects that are returned by asynchronous methods that are cancellable. It makes it possible to bring these operations to a stop if they haven't produced a result within a desired time or before a given event, thereby possibly reclaiming resources.
TBD: Elaborate on cancel of ongoing configureSensor() operation...

Methods

`cancel`

Method Cancel

Signature

```
void cancel();
```

Cancel the pending asynchronous operation. When this method is called, the user agent must immediately bring the operation to a stop and return. No success or error callback for the pending operation will be invoked.

2.7. SensorEvent

This interface defines the "genericsensor" event type.

```
interface SensorEvent : Event {
 const unsigned short SENSOR_STATUS_ACCURACY_HIGH = 4;
 const unsigned short SENSOR_STATUS_ACCURACY_MEDIUM = 3;
 const unsigned short SENSOR_STATUS_ACCURACY_LOW = 2;
 const unsigned short SENSOR_STATUS_UNRELIABLE = 1;
 const unsigned short SENSOR_STATUS_UNAVAILABLE = 0;
 readonly attribute DOMString sensorType;
 readonly attribute DOMString sensorId;
 readonly attribute unsigned short accuracy;
 readonly attribute unsigned short rate;
 readonly attribute boolean interrupt;
 readonly attribute float[] sensorValues;
 void initSensorEvent(in DOMString type,
                      in boolean bubbles,
                      in boolean cancelable,
                      in DOMString sensorType,
                      in DOMString sensorId,
                      in unsigned short accuracy,
                      in unsigned short rate,
                      in boolean interrupt,
                      in float[] sensorValues);
 };
```

Registration for generic sensor events is achieved by calling addEventListener instantiated on the selected sensor object with event type set to "sensor" (see code example in the beginning of this specification)

Constants

`unsigned short SENSOR_STATUS_ACCURACY_HIGH`

A constant describing that the sensor is reporting data with maximum accuracy.

`unsigned short SENSOR_STATUS_ACCURACY_MEDIUM`

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

A constant describing that the sensor is reporting data with an average level of accuracy, calibrating with the environment may improve the reading.

`unsigned short SENSOR_STATUS_ACCURACY_LOW`

A constant describing that the sensor is reporting with low accuracy, calibrating with the environment is needed.

`unsigned short SENSOR_STATUS_UNRELIABLE`

A constant describing that the sensor data cannot be trusted, calibrating is needed or the environment does not allow reading.

`unsigned short SENSOR_STATUS_UNAVAILABLE`

A constant describing that the sensor is not available and no sensor data can be provided. This accuracy attribute will for example take this value when contact is lost with a sensor using Bluetooth communication.

Attributes

`readonly DOMString sensorType`

The type of sensor. This is a URI defining the sensor type according to the defined sensor "feature" URI strings. See section "Features".

For the defined sensor types the sensorValues array contains the following data:

http://webinos.org/api/sensors.light:
sensorValue[0] = the measured ambient light level around the device in SI lux units.
sensorValue[1] = A normalized value between 0 and 1.

http://webinos.org/api/sensors.noise:
sensorValue[0] = the measured ambient noise around the device, in DB(SPL).
sensorValue[1] = A normalized value between 0 and 1.

http://webinos.org/api/sensors.temperature:
sensorValue[0] = the measured ambient temperature around the device, in degrees Celsius.
sensorValue[1] = A normalized value between 0 and 1.

http://webinos.org/api/sensors.pressure:
sensorValue[0] = the measured atmospheric pressure around the device in hPa (millibar)
sensorValue[1] = A normalized value between 0 and 1.

http://webinos.org/api/sensors.proximity:
sensorValue[0] = Proximity sensor distance measured in centimeters.
sensorValue[1] = A normalized value between 0 and 1.Some sensor can only state "near" (0) and "far" (1)

This attribute is readonly.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`readonly DOMString sensorId`

The unique identity of the of the specific sensor

This attribute is readonly.

`readonly unsigned short accuracy`

The accuracy of the sensor

This attribute is readonly.

`readonly unsigned short rate`

The rate of the sensor data

This attribute is readonly.

`readonly boolean interrupt`

Interrupt mode of the sensor. The value is one of false = INTERRUPT_DISABLED (events fired with a fixed time interval) true = INTERRUPT_ENABLED (events fired when value changes)

This attribute is readonly.

`readonly float [] sensorValues`

Array of sensor values

This attribute is readonly.

Methods

`initSensorEvent`

Method to set initial values of sensor event.

Signature

```
void initSensorEvent(in DOMString type, in boolean bubbles, in boolean
cancelable, in DOMString sensorType, in DOMString sensorId, in unsigned
short accuracy, in unsigned short rate, in boolean interrupt, in

                  float

                [] sensorValues);
```

The initSensorEvent() method must initialize the event in a manner analogous to the initEvent() method in http://www.w3.org/TR/2010/WD-DOM-Level-3-Events-20100907/. The method can for example be used with document.createEvent() and EventTarget.dispatchEvent() to simulate a specific event. The sensorType, sensorId, accuracy, rate, interrupt and sensorvalues arguments must initialize the attributes with the same names.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Parameters

- **type**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** Event type i.e. 'sensor'
- **bubbles**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event bubbles
- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event cancelable
- **sensorType**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** Sensor type as a URI
- **sensorId**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** The unique identity of the specific sensor
- **accuracy**
  - **Optional:** No.
  - **Nullable**: No

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

  - **Type:** unsigned short
  - **Description:** Accuracy of sensor data
- **rate**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** unsigned short
  - **Description:** Rate
- **interrupt**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** Interrupt mode
- **sensorValues**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** array
  - **Description:** Array of sensor values

3. Exceptions

3.1. SensorException

Defines the error codes for this module

```
exception SensorException {
    const unsigned short INVALID_INPUT_ARGUMENT = 0;
    unsigned short code;
    DOMString message;
};
```

Field

`unsigned short code`

Exception code

`DOMString message`

Exception message

4. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/sensors

Identifies all the sensor types.

http://webinos.org/api/sensors.light

Identifies the light sensor type.

http://webinos.org/api/sensors.noise

Identifies the noise sensor type.

http://webinos.org/api/sensors.temperature

Identifies the temperature sensor type.

http://webinos.org/api/sensors.pressure

Identifies the pressure sensor type.

5. Full WebIDL

```
module sensors {

   exception SensorException {
       const unsigned short INVALID_INPUT_ARGUMENT = 0;
       unsigned short code;
       DOMString message;
   };

      [NoInterfaceObject] interface Sensor : Service {
        readonly attribute float?          maximumRange;
        readonly attribute unsigned long?   minDelay;
        readonly attribute float?         power;
        readonly attribute float?         resolution;
        readonly attribute DOMString?      vendor;
        readonly attribute unsigned long?  version;

        PendingOp    configureSensor    (in    ConfigureSensorOptions    options,    in
ConfigureSensorCB successCB, in optional SensorErrorCB errorCB)
            raises (SensorException);

     };

    [Callback=FunctionOnly, NoInterfaceObject] interface ConfigureSensorCB {
       void onSuccess();
    };

    [Callback=FunctionOnly, NoInterfaceObject] interface SensorErrorCB {
       void onErrror(in SensorCBError error);
    };

    [NoInterfaceObject] interface SensorCBError {
       const unsigned short UNKNOWN_ERROR = 0;
       const unsigned short TIMEOUT_ERROR = 1;
       const unsigned short ILLEGAL_SENSOR_TYPE_ERROR = 2;
       const unsigned short SENSOR_TYPE_NOT_SUPPORTED_ERROR = 3;
       const unsigned short ILLEGAL_SENSOR_ID_ERROR = 4;
       const unsigned short OTHER_ILLEGAL_INPUT_ARGUMENT_ERROR = 5;
       const unsigned short REQUESTED_RATE_NOT_SUPPORTED_ERROR = 6;
       const unsigned short REQUESTED_INTERRUPTMODE_NOT_SUPPORTED_ERROR = 7;
       const unsigned short PERMISSION_DENIED_ERROR = 50;
       readonly attribute unsigned short code;
       readonly attribute DOMString message;
    };

    [NoInterfaceObject] interface ConfigureSensorOptions {
       const unsigned short INFINITE = 0;
       const unsigned short SENSOR_DELAY_FASTEST = 0;
       const unsigned short SENSOR_DELAY_GAME = 1;
       const unsigned short SENSOR_DELAY_UI = 2;
       const unsigned short SENSOR_DELAY_NORMAL = 3;
       attribute unsigned short timeout;
       attribute unsigned short rate;
       attribute boolean interrupt;
     };

     [NoInterfaceObject] interface PendingOp {
        void cancel ();
     };

   interface SensorEvent : Event {
    const unsigned short SENSOR_STATUS_ACCURACY_HIGH = 4;
    const unsigned short SENSOR_STATUS_ACCURACY_MEDIUM = 3;
    const unsigned short SENSOR_STATUS_ACCURACY_LOW = 2;
    const unsigned short SENSOR_STATUS_UNRELIABLE = 1;
    const unsigned short SENSOR_STATUS_UNAVAILABLE = 0;
    readonly attribute DOMString sensorType;
    readonly attribute DOMString sensorId;
    readonly attribute unsigned short accuracy;
    readonly attribute unsigned short rate;
    readonly attribute boolean interrupt;
    readonly attribute float[] sensorValues;
    void initSensorEvent(in DOMString type,
                         in boolean bubbles,
                         in boolean cancelable,
                         in DOMString sensorType,
                         in DOMString sensorId,
                         in unsigned short accuracy,
                         in unsigned short rate,
                         in boolean interrupt,
                         in float[] sensorValues);
    };
};
```

APIs: The discovery module

Webinos API Specifications

30 Jun 2011

Authors

- Anders Isberg <anders.isberg@sonyericsson.com>

Abstract

Webinos Discovery API

Summary of Methods

Interface	Method
DiscoveryInterface	PendingOperation findServices(ServiceType serviceType, FindCallBack findCallBack, Options options, Filter filter) DOMString getServiceId(DOMString serviceType) Service createService()
ServiceType	
FindCallBack	void onFound(Service service) void onLost(Service service) void onError(DiscoveryError error)
Service	PendingOperation bind(BindCallBack bindCallBack, DOMString serviceId) void unbind()
BindCallBack	void onBind(Service service) void onUnbind(Service service) void onServiceAvailable(Service service) void onServiceUnavailable(Service service) void onError(DiscoveryError error)

Interface	Method
Options	
Filter	
ServiceLocation	
PendingOperation	void cancel()
DiscoveryError	
WebinosDiscovery	

1. Introduction

The Webinos Discovery API provide web applications with an API to discover services without any previous knowledge of the service. The Discovery API is not limited to discovery of local services but also enables discovery of remote services.

The API enables discovery of services that is exposed either:

1. in the device
2. by entities directly connected to the device,
3. by entities available on the same local IP network
4. by trusted services registered in a personal zone.

Once a service is found the API will provide a service object that is used to bind to a service and monitor the availability of the service. The binding to a service will make sure that the user is authorized to use the service, create an implementation of the API and establish a communication path to the remote peer providing the service. The service object hides the complexity of communicating over different bearers, do cross network addressing, traversing NAT/Firewalls and connection management.

Prerequisites for the Discovery API is that the web application using the API is installed, trusted and that the user of of the device is authenticated and authorized to use the API.

2. Interfaces

2.1. DiscoveryInterface

The DiscoveryInterface interface provides functionality for discovery of services. The API supports the possibility to discover services based on a given service type either in a personal zone of trusted services or via other legacy discovery methods such as Bluetooth SD, DNS SD, mDNS or UPnP. When searching for a service type the operation can be restricted by providing certain constraints and/or context information via a filter interface.

```
[NoInterfaceObject] interface DiscoveryInterface  {

        PendingOperation findServices(
               in ServiceType serviceType,
               in FindCallBack findCallBack,
               in optional Options options,
               in optional Filter filter)
        raises(DiscoveryExceptions);

        DOMString getServiceId(in DOMString serviceType)
        raises(DiscoveryExceptions);

        Service createService()
        raises(DiscoveryExceptions);
};
```

The code example below shows how an application initiates a search query to find a geolocation service. Whenever a service is found, a new HTML selection item is added to an HTML option list. Once the user selects a service, the usage of the service is authorized and an implementation of the API is instantiated by binding to the service.

Code example

```
        var findHandle = 0;
        var serviceHandle = 0;
        var geoServices = {};
        var serviceId;

        // Callback that displays a list of found services in a HTML selection list
        // The selection list is dynamically extended every time a new service is
discovered.
        function serviceFoundCB(service) {
                var selectlist = document.getElementById('servicelist');
                var option = document.createElement('option');

                option.value = service.id;
                option.id = service.id;
                option.appendChild(document.createTextNode(service.displayName));
                geoServices[service.id] = service;
                selectlist.appendChild(option);
        }

        // Callback that removes a service from the selection list of found services,
        // when the service is not available any longer.
        function serviceLostCB(service) {
                var option = document.getElementById(service.id);

                geoServices[service.id] = NULL;
                option.parentNode.removeChild(option);
        }

        // Success callback when bind has been successfully executed on the service
object.
        function bindCB(myLocationService) {
                alert('Service ' + myLocationService.displayName + ' ready to use');
                myLocationService.navigator.geolocation.getCurrentPosition(showMap);
        }

        // Event listner that is called when the 'change' event is dispatched on the
HTML selection list.
        function serviceSelected(service) {
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
                // Stops the findServices operation
                findHandle.cancel();

                // Binds to the service to initiate an authorized object used to
                // invoke services.
                serviceHandle = service.bind({onBind:bindCB});
        }

        if (serviceId) {
                // If serviceId is known, bind to the service directly. Assumes
                // that serviceId is stored persistently or received via an out
                // of band channel.
                serviceHandle                                                    =
window.webinos.discovery.createService().bind({onBind:bindCB}, serviceId);
        }
        else {
                // Initiate a search query for a service of the type geolocation
                findHandle = window.webinos.discovery.findServices(
                        {api:'http://www.w3.org/ns/api-perms/geolocation'},
                        {onFound:serviceFoundCB, onLost:serviceLostCB});

                var selectlist = document.getElementById('servicelist');
                selectlist.addEventListener("change", function (e) {
                        var service = geoServices[e.target.value];
                        if (service) {
                                serviceSelected(service);
                        }
                }, false);
        }
```

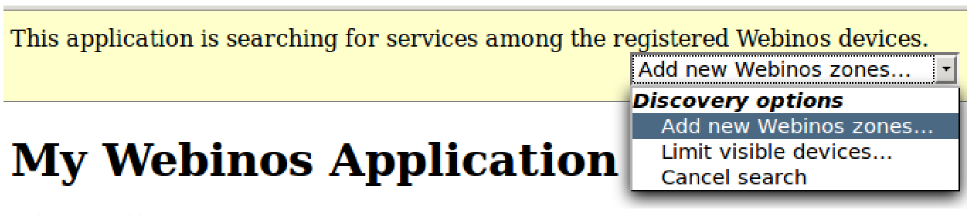
Methods

`findServices`

The findServices method initiates an asynchronous search query for services matching the requested serviceType and filter parameter. The method continues to search for services until the findServices method is canceled by the application or when the maximum search timer expires. The zones in which services are to be searched are expected to be managed by the Webinos runtime engine (rather than the application developer). Below is an example on how this zone management can be presented by a runtime engine.

Signature

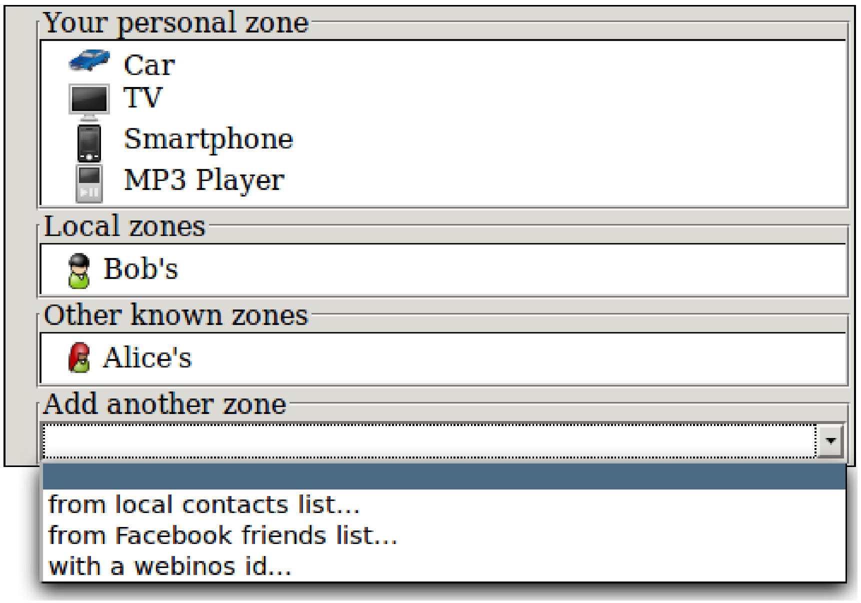
`PendingOperation findServices(in ServiceType serviceType, in FindCallBack findCallBack, in optional Options options, in optional Filter filter);`


This application is searching for services among the registered Webinos devices.

Infobar offering to add new personal zones

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

This infobar would lead to a personal zones management UI:


Your personal zone
Car
TV
Smartphone
MP3 Player
Local zones
Bob's
Other known zones
Alice's
Add another zone
from local contacts list...
from Facebook friends list...
with a webinos id...

Personal zone management UI

Parameters

- **serviceType**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ServiceType
  - **Description:** An input argument that defines which type of API that is requested. The serviceType is an URI that uniquely identifies the API.
- **findCallBack**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** FindCallBack
  - **Description:** Callback interface used to report the outcome of the search process.
- **options**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** Options
  - **Description:** Defines search options.

- **filter**
    - **Optional:** Yes.
    - **Nullable**: No
    - **Type:** Filter
    - **Description:** Defines a filter that be used to limit the service operation to certain constraints and context information.

Exceptions

- DiscoveryExceptions:

getServiceId

The getServiceId method generates a service identity that can be shared with other peers to establish a binding without invoking a findServices operation. If no matching API is found the method will return Null.

Signature

```
DOMString getServiceId(in DOMString serviceType);
```

Parameters

- **serviceType**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** URI identifying which type of API that shall be exposed. The URI shall be must be declared in the manifest in the api-name attribute of the webinos:shared-api element.

Exceptions

- DiscoveryExceptions:

createService

The createService method creates an instance of a Service object that can be used to establish a service binding directly if the service identity is already known. This is for example applicable if the service identity is stored persistently in a database or received via some kind of out of band channel.

Signature

```
Service createService();
```

Exceptions

- DiscoveryExceptions:

2.2. ServiceType

The Service Type interface is used to define which type of service that is requested.

```
[NoInterfaceObject] interface ServiceType {
        attribute DOMString api;
};
```

Attributes

`DOMString api`

URI used to identify which type of API that is requested. The URI could either be:

1. W3C DAP API URI as defined in http://www.w3.org/TR/2010/WD-api-perms-20101005/, for example http://www.w3.org/ns/api-perms/geolocation,
2. Webinos Feature URI that is defined for each API by Webinos, for example http://webinos.org/api/sensors.temperature,
3. WAC Feature URI that is defined for each API by WAC, for example http://waclists.org/api/camera,
4. an unique URI identifying an API exposed by a web application. The URI shall be the same URI as exposed by the web application manifest in the api-name attribute of the webinos:shared-api element.

2.3. FindCallBack

FindCallBack interface definition

```
[Callback, NoInterfaceObject] interface FindCallBack {
         void onFound(in Service service);
         void onLost(in Service service);
         void onError(in DiscoveryError error);
};
```

Methods

`onFound`

Asynchronous callback used whenever a new service is found.

Signature

```
void onFound(in Service service);
```

Parameters

- **service**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Service

  - **Description:** An input argument representing the found service.

`onLost`

Asynchronous callback used whenever the state of a service change from available to unavailable.

Signature
```
void onLost(in Service service);
```

Parameters

- **service**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Service
  - **Description:** An input argument representing the lost service.

`onError`

Asynchronous error callback.

Signature
```
void onError(in DiscoveryError error);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DiscoveryError
  - **Description:** Error code.

2.4. Service

The Service interface provides an API to bind to a specific service and monitor the availability of a bound service in an asynchronous manner. The process of binding to a service involves:

1. mutual authentication between the service and the personal zone
2. in case of cross zone personal interworking, mutual authentication between the zones
3. agreement on data handling obligations as set out in the service's privacy policy
4. verifying access privileges and checks the need for elevated privileges
5. instantiate an implementation of the API that can be used by applications to request services from the requested API.

Once the service object is instantiated, the service object will act as a proxy to the remote peer that will be able to invoke methods associated with API type under the window object of the remote peer. For example an application that successfully binds to a "http://webinos.org/api/tv" API will be able to invoke methods from the remote peer as described in the code example below.

```
        [NoInterfaceObject] interface Service {
                const unsigned short SERVICE_INITATING = 0;
                const unsigned short SERVICE_AVAILABLE = 1;
                const unsigned short SERVICE_UNAVAILABLE = 2;
                readonly attribute unsigned short state;
                readonly attribute DOMString api;
                readonly attribute DOMString id;
                readonly attribute DOMString displayName;
                readonly attribute DOMString description;
                readonly attribute DOMString icon;

                PendingOperation  bind(in  BindCallBack  bindCallBack,  in  optional
DOMString serviceId)
                raises(DiscoveryExceptions);

                void unbind()
                raises(DiscoveryExceptions);
        };
```

Code example

```
        succesBindCallBack(tvService) {
                // Invoke a remote method.
                tvService.webinos.tv.display.setChannel(channel, success);
                //  Register  an  event  listner  from  event  orginating  from  the  remote
peer.
                tvService.addEventListener('channelchange', success);
        }
```

Constants

`unsigned short SERVICE_INITATING`

A constant describing the service is in the process of binding to the service.

`unsigned short SERVICE_AVAILABLE`

A constant describing the service is available and is ready to be used by the application.

`unsigned short SERVICE_UNAVAILABLE`

A constant describing the service is unavailable.

Attributes

`readonly unsigned short state`

Current service state.

This attribute is readonly.

`readonly DOMString api`

API is a global unique URI identifying the type of API provided by the service.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

This attribute is readonly.

`readonly DOMString id`

Id is a globally unique id representing the binding to the service. The id can be used to resume the binding again to the service without invoking the findServices process again. The id can be stored persistently to be able to resume a binding to a service across power cycles.

This attribute is readonly.

`readonly DOMString displayName`

A human readable name of the service.

This attribute is readonly.

`readonly DOMString description`

An URL referring to a detailed description of the service.

This attribute is readonly.

`readonly DOMString icon`

Icon is an URL referring to an icon that represents the service.

This attribute is readonly.

Methods

`bind`

bind Binds to the service uniquely identified by the service identity.

Signature

```
PendingOperation bind(in BindCallBack bindCallBack, in optional DOMString serviceId);
```

Parameters

- **bindCallBack**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** BindCallBack
  - **Description:** Asynchrounous callback to report the states of the bind operation and the availability of the service.
- **serviceId**
  - **Optional:** Yes.

- **Nullable**: No
- **Type:** DOMString
- **Description:** Unique id of the binding to the particular service. If no serviceId is provided as an in parameter, the id attribute in the Service interface will be used to bind the service.

Exceptions

- DiscoveryExceptions:

unbind

unbind Releases all resources and connections allocated by the service object.

Signature

```
void unbind();
```

Exceptions

- DiscoveryExceptions:

2.5. BindCallBack

Bind success callback interface definition

```
    [Callback, NoInterfaceObject] interface BindCallBack {
            void onBind(in Service service);
            void onUnbind(in Service service);
            void onServiceAvailable(in Service service);
            void onServiceUnavailable(in Service service);
            void onError(in DiscoveryError error);
   };
```

Methods

onBind

Asynchronous success callback.

Signature

```
void onBind(in Service service);
```

Parameters

- **service**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Service
  - **Description:** An input argument representing the service.

onUnbind

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Asynchronous callback used when a service is unbound.

Signature
```
void onUnbind(in Service service);
```

Parameters

- **service**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Service
  - **Description:** An input argument representing the service.

`onServiceAvailable`

Asynchronous callback indicating that the service is available again.

Signature
```
void onServiceAvailable(in Service service);
```

Parameters

- **service**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Service
  - **Description:** An input argument representing the service.

`onServiceUnavailable`

Asynchronous callback indicating the service is temporarily unavailable.

Signature
```
void onServiceUnavailable(in Service service);
```

Parameters

- **service**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Service
  - **Description:** An input argument representing the service.

`onError`

Asynchronous error callback.

Signature

```
void onError(in DiscoveryError error);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DiscoveryError
  - **Description:** Error code.

2.6. Options

Option interface definition

```
[NoInterfaceObject] interface Options {
        attribute unsigned short timeout;
};
```

Attributes

`unsigned short timeout`

A timeout value for the findService operation in seconds between 0-65535. Default value is 120 seconds. It is possible to disable the timeout by setting the timeout value to Number.POSITIVE_INFINITY.

2.7. Filter

Filter interface

```
[NoInterfaceObject] interface Filter {
        attribute DOMString[] zoneId;
        attribute boolean remoteServices;
        attribute ServiceLocation? serviceLocation;
};
```

Attributes

`DOMString [] zoneId`

Identities of personal zones that will be used to search for services in addition to the person zone of the user logged into to the device and all personal zones that has been defined via the personal zone management UI.

`boolean remoteServices`

If remoteService is false the findServices method will limit the search for services that are connected directly to the device or to the same local IP network. If remoteServices is true, the findServices method will extend the search for services outside the local IP network. Default value is false.

`ServiceLocation? serviceLocation`

With the serviceLocation attribute it is possible to indicate where the service shall be located. If the service location is Null, the location of the service is not considered during the findServices process.

2.8. ServiceLocation

ServiceLocation interface

```
[NoInterfaceObject] interface ServiceLocation {
        attribute double? latitude;
        attribute double? longitude;
        attribute double accuracy;
};
```

Attributes

`double? latitude`

The latitude attribute is the geographic coordinate specified in decimal degrees. If the latitude is Null the latitude of the device invoking the findServices method will be used.

`double? longitude`

The longitude attribute is the geographic coordinate specified in decimal degrees. If the longitude is Null the longitude of the device invoking the findServices method will be used.

`double accuracy`

The accuracy denotes the accuracy level of the latitude and longitude coordinates in meters. This is used to limit the geographical area for finding services.

2.9. PendingOperation

Pending Operation interface

```
[NoInterfaceObject] interface PendingOperation {
        void cancel();
};
```

Methods

`cancel`

Cancels the pending asynchronous operation and allocated resources are released.

Signature
```
void cancel();
```

2.10. DiscoveryError

Discovery specific errors.

```
[NoInterfaceObject] interface DiscoveryError {
        const unsigned short FIND_SERVICE_CANCELED = 101;
        const unsigned short FIND_SERVICE_TIMEOUT = 102;
```

```
                const unsigned short PERMISSION_DENIED_ERROR  = 103;
                readonly attribute unsigned short code;
                readonly attribute DOMString message;
        };
```

Constants

`unsigned short FIND_SERVICE_CANCELED`

The discovery process was canceled by the application

`unsigned short FIND_SERVICE_TIMEOUT`

The discovery process was canceled since the timeout timer expired.

`unsigned short PERMISSION_DENIED_ERROR`

Not Authorized to use the service.

Attributes

`readonly unsigned short code`

Error code assigned when an error has occurred in during the discovery process.

This attribute is readonly.

`readonly DOMString message`

Human readable message assigned when an error has occurred the discovery process.

This attribute is readonly.

2.11. WebinosDiscovery

The WebinosDiscovery interface describes the part of the Discovery API accessible through the webinos object.

```
        [NoInterfaceObject] interface WebinosDiscovery {
                readonly attribute DiscoveryInterface discovery;
        };
        webinoscore::Webinos implements WebinosDiscovery;
```

Attributes

`readonly DiscoveryInterface discovery`

webinos.discovery object.

This attribute is readonly.

3. Exceptions

3.1. DiscoveryExceptions

Discovery specific exceptions.

```
        exception DiscoveryExceptions {
                const unsigned short INVALID_ARGUMENT_ERROR = 101;
                unsigned short code;
                DOMString message;
        };
```

Field

`unsigned short code`

Exception code.

`DOMString message`

Human readable exception message.

4. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/discovery

5. Full WebIDL

```
module discovery {

        [NoInterfaceObject] interface DiscoveryInterface  {

                 PendingOperation findServices(
                        in ServiceType serviceType,
                        in FindCallBack findCallBack,
                        in optional Options options,
                        in optional Filter filter)
                 raises(DiscoveryExceptions);

                 DOMString getServiceId(in DOMString serviceType)
                 raises(DiscoveryExceptions);

                 Service createService()
                 raises(DiscoveryExceptions);
        };

        [NoInterfaceObject] interface ServiceType {
                attribute DOMString api;
        };


        [Callback, NoInterfaceObject] interface FindCallBack {
                 void onFound(in Service service);
                 void onLost(in Service service);
                 void onError(in DiscoveryError error);
        };

        [NoInterfaceObject] interface Service {
                const unsigned short SERVICE_INITATING = 0;
                const unsigned short SERVICE_AVAILABLE = 1;
                const unsigned short SERVICE_UNAVAILABLE = 2;
                readonly attribute unsigned short state;
                readonly attribute DOMString api;
                readonly attribute DOMString id;
                readonly attribute DOMString displayName;
                readonly attribute DOMString description;
                readonly attribute DOMString icon;

                PendingOperation  bind(in  BindCallBack  bindCallBack,  in  optional
DOMString serviceId)
                raises(DiscoveryExceptions);

                void unbind()
                raises(DiscoveryExceptions);
        };

         [Callback, NoInterfaceObject] interface BindCallBack {
                 void onBind(in Service service);
                 void onUnbind(in Service service);
                 void onServiceAvailable(in Service service);
                 void onServiceUnavailable(in Service service);
                 void onError(in DiscoveryError error);
        };

        [NoInterfaceObject] interface Options {
                attribute unsigned short timeout;
        };

        [NoInterfaceObject] interface Filter {
                attribute DOMString[] zoneId;
                attribute boolean remoteServices;
                attribute ServiceLocation? serviceLocation;
        };

        [NoInterfaceObject] interface ServiceLocation {
                attribute double? latitude;
                attribute double? longitude;
                attribute double accuracy;
        };

        [NoInterfaceObject] interface PendingOperation {
                void cancel();
        };

        [NoInterfaceObject] interface DiscoveryError {
                const unsigned short FIND_SERVICE_CANCELED = 101;
                const unsigned short FIND_SERVICE_TIMEOUT = 102;
                const unsigned short PERMISSION_DENIED_ERROR  = 103;
                readonly attribute unsigned short code;
                readonly attribute DOMString message;
        };

        exception DiscoveryExceptions {
                const unsigned short INVALID_ARGUMENT_ERROR = 101;
                unsigned short code;
                DOMString message;
        };
        [NoInterfaceObject] interface WebinosDiscovery {
                readonly attribute DiscoveryInterface discovery;
        };
        webinoscore::Webinos implements WebinosDiscovery;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The tv module

Webinos API Specifications

1 Jul 2011

Authors

- Fraunhofer FOKUS, Alexander Futász <alexander.futasz@fokus.fraunhofer.de>
- Dominique Hazael-Massieux

Abstract

Interface for TV control and managment.

Summary of Methods

Interface	Method
WebinosTV	
TVManager	
TVDisplayManager	void setChannel(Channel channel, TVDisplaySuccessCB successCallback, TVErrorCB errorCallback)
TVDisplaySuccessCB	void onSuccess(Channel channel)
TVTunerManager	void getTVSources(TVSuccessCB successCallback, TVErrorCB errorCallback)
TVSuccessCB	void onSuccess(TVSource [] sources)
TVErrorCB	void onError(TVError error)
TVError	
TVSource	

Interface	Method
Channel	
ChannelChangeEvent	void initChannelChangeEvent(DOMString type, boolean bubbles, boolean cancelable, Channel channel)

1. Introduction

The interface provides means to acquire a list of tv sources, channels and their streams.

The TV channel streams can be displayed in HTMLVideoElement object (http://dev.w3.org/html5/spec/video.html). Alternatively the API provides means to control channel management of the native hardware TV, by allowing to set a channel or watch for channel changes that are invoked otherwise.

The tv object is made available under the webinos namespace, i.e. webinos.tv.

2. Interfaces

2.1. WebinosTV

Creates tv object.

```
[NoInterfaceObject]
interface WebinosTV {
  readonly attribute TVManager tv;
};
webinoscore::Webinos implements WebinosTV;
```

2.2. TVManager

Access to tuner and display managers.

```
[NoInterfaceObject]
interface TVManager {
  readonly attribute TVDisplayManager display;
  readonly attribute TVTunerManager tuner;
};
```

2.3. TVDisplayManager

Interface to manage what's currently displayed on TV screen.

```
[NoInterfaceObject]
interface TVDisplayManager {
  void  setChannel(Channel  channel,  TVDisplaySuccessCB  successCallback,  optional
TVErrorCB errorCallback);
};
```

This interface is useful when an app doesn't want to show the broadcast itself, but let the TV natively handle playback, i.e. not in a web context. Useful to build an control interface that allows channel switching.

Code example

```
  <p>Currently shown on TV: <span id='tv'>Undetermined</span></p>
  <script>
  var channel; // holding a previously obtained channel object.
  webinos.tv.display.setChannel(channel, success);
  var ontv = document.getElementById('tv');
  function success(channel) {
    ontv.normalize();
    ontv.removeChild(ontv.childNodes[0]);
    ontv.appendChild(document.createTextNode(channel.name    +    '    (source:    '    +
channel.tvsource.name + ')'));
  }
  </script>
```

Methods

`setChannel`

Switches the channel natively on the TV (same as when a hardware remote control would be used).

Signature

```
void setChannel(Channel channel, TVDisplaySuccessCB successCallback,
optional TVErrorCB errorCallback);
```

Parameters

- **channel**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Channel
  - **Description:** The TV channel to switch to.
- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** TVDisplaySuccessCB
  - **Description:** The callback to notify the caller that the channel change succeeded.
- **errorCallback**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** TVErrorCB

- **Description:** The callback called in case the channel could not be switched and an error occured.

2.4. TVDisplaySuccessCB

Callback function when current channel changed successfully.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface TVDisplaySuccessCB {
  void onSuccess(Channel channel);
};
```

2.5. TVTunerManager

Get a list of all available TV tuners.

```
[NoInterfaceObject]
interface TVTunerManager {
  void getTVSources(TVSuccessCB successCallback, optional TVErrorCB errorCallback);
};
```

Code example

```
  <label>Pick a TV Source: <select id='source'>
  <option>None</option>
  </select></label>
  <label>Pick a  channel: <select id='channel'>
  <option>None</option>
  </select></label>
  <video id='display' width='640' height='400' poster='nochannel.png'></video>
  <script>
  webinos.tv.tuner.getTVSources(successCB);
  var tvsourceselector = document.getElementById('source');
  var channelselector = document.getElementById('channel');
  var v = document.getElementById('display');
  var currentTVSource;
  var tvsources = [];
  function successCB(sources) {
    tvsources = sources;
    for (var i in sources) {
      var o = document.createElement('option');
      o.value = i;
      o.appendChild(document.createTextNode(sources[i].name);
      tvsourceselector.appendChild(o);
    }
  }
  tvsourceselector.addEventListener('change', function (e) {
    currentTVSource = tvsources[e.target.value];
    // start showing first channel
    if (currentTVSource.channelList.length) {
      v.src = currentTVSource.channelList[0].stream;
      for (var i in currentTVSource.channelList) {
          var channel = currentTVSource.channelList[i];
          var o = document.createElement('option');
          o.appendChild(document.createTextNode(channel.name);
          o.value = i;
          channelselector.appendChild(o);
    }
  }, false);
  channelselector.addEventListener('change', function (e) {
     if (e.target.value) {
       v.src = currentTVSource.channelList[e.target.value].stream;
     }
  }, false);
  </script>
```

Methods

`getTVSources`

Get a list of all available TV tuners.

Signature

```
void getTVSources(TVSuccessCB successCallback, optional TVErrorCB
errorCallback);
```

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** TVSuccessCB
  - **Description:** Callback that receives all available TV sources.
- **errorCallback**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** TVErrorCB
  - **Description:** Callback called in case something goes wrong.

2.6. TVSuccessCB

Callback for found TV tuners.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface TVSuccessCB {
  void onSuccess(TVSource[] sources);
};
```

Methods

`onSuccess`

Callback that is called with the found TV sources.

Signature

```
void onSuccess(
                TVSource
              [] sources);
```

Parameters

- **sources**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** array
  - **Description:** An array of TVSource objects representing available tuners.

2.7. TVErrorCB

Error callback for errors when trying to get TV tuners.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface TVErrorCB {
  void onError(TVError error);
};
```

Methods

onError

Callback that is called when an error occures while getting TV sources

Signature

```
void onError(TVError error);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** TVError
  - **Description:** Error object detailing what went wrong.

2.8. TVError

Error codes.

```
[NoInterfaceObject]
interface TVError {
  const unsigned short UNKNOWN_ERROR = 0;

  const unsigned short ILLEGAL_CHANNEL_ERROR = 1;

  readonly attribute unsigned short code;
};
```

Constants

`unsigned short UNKNOWN_ERROR`

An unknown error.

`unsigned short ILLEGAL_CHANNEL_ERROR`

Invalid input channel.

Attributes

`readonly unsigned short code`

Code.

This attribute is readonly.

2.9. TVSource

TV source: a list of channels with a name.

```
[NoInterfaceObject]
interface TVSource {
  readonly attribute DOMString name;

  readonly attribute Channel[] channelList;
};
```

Attributes

`readonly DOMString name`

The name of the source.

The name should describe the kind of tuner this source represents, e.g. DVB-T, DVB-C.

This attribute is readonly.

`readonly Channel [] channelList`

List of channels for this source.

This attribute is readonly.

2.10. Channel

The Channel Interface

```
[NoInterfaceObject]
interface Channel {
  const unsigned short TYPE_TV = 0;
  const unsigned short TYPE_RADIO = 1;
  readonly attribute unsigned short channelType;
  readonly attribute DOMString name;
  readonly attribute DOMString longName;
  readonly attribute Stream stream;
  readonly attribute TVSource tvsource;
};
```

Channel objects provide access to the video stream.

Constants

`unsigned short TYPE_TV`

Indicates a TV channel.

`unsigned short TYPE_RADIO`

Indicates a radio channel.

Attributes

`readonly unsigned short channelType`

The type of channel.

Type of channel is defined by one of the TYPE_* constants defined above.

This attribute is readonly.

`readonly DOMString name`

The name of the channel.

The name of the channel will typically be the call sign of the station.

This attribute is readonly.

`readonly DOMString longName`

The long name of the channel.

The long name of the channel if transmitted. Can be undefined if not available.

This attribute is readonly.

`readonly Stream stream`

The video stream.

This stream is a represents a valid source for a HTMLVideoElement.

This attribute is readonly.

`readonly TVSource tvsource`

The source this channels belongs too.

This attribute is readonly.

2.11. ChannelChangeEvent

Event that fires when the channel is changed.

```
interface ChannelChangeEvent : Event {
  readonly attribute Channel channel;
  void initChannelChangeEvent(DOMString type, boolean bubbles, boolean cancelable,
Channel channel);
};
```

Changing channels could also be invoked by other parties, e.g. a hardware remote control. A ChannelChange event will be fire in these cases which provides the channel that was switched to.

Code example

```
  <p>Currently shown on TV: <span id='tv'>Undetermined</span></p>
  <script>
  window.addEventListener('channelchange', success);
  var ontv = document.getElementById('tv');
  function success(channel) {
    ontv.normalize();
    ontv.removeChild(ontv.childNodes[0]);
    ontv.appendChild(document.createTextNode(channel.name + ' (source: ' +
channel.tvsource.name + ')'));
  }
  </script>
```

Attributes

`readonly Channel channel`

The new channel.

This attribute is readonly.

Methods

`initChannelChangeEvent`

Initializes a new channel change event.

Signature

```
void initChannelChangeEvent(DOMString type, boolean bubbles, boolean
cancelable, Channel channel);
```

Parameters

- **type**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** The type of event. Pass 'channelchange'.
- **bubbles**
  - **Optional:** No.

  - **Nullable**: No
  - **Type:** boolean
  - **Description:** Indicates whether the event bubbles.
- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** Indicates whether the event is cancelable.
- **channel**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Channel
  - **Description:** The channel that was changed to.

3. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/tv

4. Full WebIDL

```
module tv {

[NoInterfaceObject]
interface WebinosTV {
  readonly attribute TVManager tv;
};

webinoscore::Webinos implements WebinosTV;

[NoInterfaceObject]
interface TVManager {
  readonly attribute TVDisplayManager display;
  readonly attribute TVTunerManager tuner;
};

[NoInterfaceObject]
interface TVDisplayManager {
  void  setChannel(Channel  channel,  TVDisplaySuccessCB  successCallback,  optional
TVErrorCB errorCallback);
};
[Callback=FunctionOnly, NoInterfaceObject]
interface TVDisplaySuccessCB {
  void onSuccess(Channel channel);
};

[NoInterfaceObject]
interface TVTunerManager {
  void getTVSources(TVSuccessCB successCallback, optional TVErrorCB errorCallback);
};

[Callback=FunctionOnly, NoInterfaceObject]
interface TVSuccessCB {
  void onSuccess(TVSource[] sources);
};

[Callback=FunctionOnly, NoInterfaceObject]
interface TVErrorCB {
  void onError(TVError error);
};

[NoInterfaceObject]
interface TVError {
  const unsigned short UNKNOWN_ERROR = 0;
  const unsigned short ILLEGAL_CHANNEL_ERROR = 1;
  readonly attribute unsigned short code;
};

[NoInterfaceObject]
interface TVSource {
  readonly attribute DOMString name;
  readonly attribute Channel[] channelList;
};

[NoInterfaceObject]
interface Channel {
  const unsigned short TYPE_TV = 0;
  const unsigned short TYPE_RADIO = 1;
  readonly attribute unsigned short channelType;
  readonly attribute DOMString name;
  readonly attribute DOMString longName;
  readonly attribute Stream stream;
  readonly attribute TVSource tvsource;
};

interface ChannelChangeEvent : Event {
  readonly attribute Channel channel;

  void  initChannelChangeEvent(DOMString  type,  boolean  bubbles,  boolean  cancelable,
Channel channel);

};
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The userprofile module

Webinos API Specifications

28 Jun 2011

Authors

- WIDL version for webinos created by Ronny Gräfe <ronny.graefe@t-systems.com>

Abstract

The webinos userprofile API to access user information.

Summary of Methods

Interface	Method
UserProfileInterface	void find(DOMString [] fields, UserProfileFindCB successCB, UserProfileErrorCB errorCB, UserProfileFindOptions options) void createUserProfile(UserProfile userProfile, SuccessCB successCallBack, UserProfileErrorCB errorCallback) void replaceUserProfile(DOMString id, UserProfile userProfile, SuccessCB successCallBack, UserProfileErrorCB errorCallback) void deleteUserProfile(DOMString id, SuccessCB successCallBack, UserProfileErrorCB errorCallback)
UserProfile	
SocialNetworkProfile	
UserProfileFindOptions	
UserProfileError	
UserProfileErrorCB	void onerror(UserProfileError error)

Interface	Method
UserProfileFindCB	void onsuccess(UserProfile [] userProfileObjs)
WebinosUserProfile	

1. Introduction

This API offers access to information of the user. UserProfile API is an extension of webinos Contact API to gather basic information about the user (e.g. name, nickname, gender, birthday, etc.) and extends it with social network attributes from Portablecontacts from August 5, 2008 (http://portablecontacts.net/draft-spec.html). These social network attributes are a simple pointer where the webinos user has non-webinos profiles. These information could be used by an application to query an external API for an additional information (e.g. query the Facebook Graph API for the buddylist).

2. Interfaces

2.1. UserProfileInterface

The UserProfileInterface interface provides methods to find, create, replace and delete a userprofile. There could be several userprofiles assigned to one sepcific webinos user.

```
        interface UserProfileInterface {

                caller void find(DOMString[] fields, UserProfileFindCB successCB,
optional UserProfileErrorCB errorCB, optional UserProfileFindOptions options);

                void createUserProfile(in UserProfile userProfile, optional SuccessCB
successCallBack, in UserProfileErrorCB errorCallback);

                void replaceUserProfile(in DOMString id, in UserProfile userProfile,
optional SuccessCB successCallBack, in UserProfileErrorCB errorCallback);

                void deleteUserProfile(in DOMString id, optional SuccessCB
successCallBack, in UserProfileErrorCB errorCallback);
        };
```

Methods

find

find() method

Signature

```
caller void find(

               DOMString

             [] fields, UserProfileFindCB successCB, optional
UserProfileErrorCB errorCB, optional UserProfileFindOptions options);
```

Find a userprofile in the webinos system according to the find user process detailed below.

This method takes two, three or four arguments. When called, it starts the following find userprofile process:

1. Let successCallback be the callback indicated by the method's second argument.

2. Let errorCallback be the callback indicated by the method's third argument, if any, or null otherwise.

3. If successCallback is null, then throw a TypeError (as defined in the WEBIDL Sepcification - http://dev.w3.org/2006/webapi/WebIDL/).

4. If there is a task from the device task source in one of the task queues (e.g. an existing find() operation is still pending a response), run these substeps:

4.1 If errorCallback is not null, let error be a UserProfileError object whose code attribute has the value PENDING_OPERATION_ERROR and queue a task to invoke errorCallback with error as its argument.

4.2 Abort this operation.

5. Return, and run the remaining steps asynchronously.

6. Let results be the array of UserPofile objects obtained by searching userprofiles in the webinos system according to the rules defined in UserProfile Search Processing, or null if the search has failed.

7. If results is null, run these substeps:

7.1 If errorCallback is not null, let error be a UserProfileError object whose code attribute has its value set according to the type of failure that occurred and queue a task to invoke errorCallback with error as its argument.

7.2 Abort this operation.

8. Queue a task to invoke successCallback with results as its argument.

Parameters

- **fields**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** array
  - **Description:**
- **successCB**
  - **Optional:** No.

- **Nullable**: No
- **Type:** UserProfileFindCB
- **Description:**

- **errorCB**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** UserProfileErrorCB
  - **Description:**
- **options**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** UserProfileFindOptions
  - **Description:**

`createUserProfile`

createuserProfile() method - Creates a new user profile in the webinos system.

Signature

```
void createUserProfile(in UserProfile userProfile, optional SuccessCB
successCallBack, in UserProfileErrorCB errorCallback);
```

Parameters

- **userProfile**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** UserProfile
  - **Description:** A new UserProfile object.
- **successCallBack**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** SuccessCB
  - **Description:** Callback issued when the creating of the user is correctly finished.
- **errorCallback**

- **Optional:** No.
- **Nullable**: No
- **Type:** UserProfileErrorCB
- **Description:** Callback issued if an error occurs during the processing time.

Code example

```
//create a new userProfile
var userProfile = new Object();

//add attributes
userProfile.displayName = 'John Smith';
userProfile.nickname = 'johnny2011';
...
userProfile.timezone = 'CET';

//creates a new userprofile in the webinos system
webinos.userprofile.createuserProfile(userProfile, successCB, errorCB);
```

`replaceUserProfile`

The replaceUserProfile() method - Replaces a userprofile. This method should be used to update a userprofile.

Signature

```
void replaceUserProfile(in DOMString id, in UserProfile userProfile, optional SuccessCB successCallBack, in UserProfileErrorCB errorCallback);
```

Three steps are necessary to use this method.

1. Get the existing user profile object

2. Update attributes

3. Use replaceUserProfile() method to update the existing user profile with new attributes by providing the entire user profile object to the method.

Parameters

- **id**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:**

- **userProfile**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** UserProfile
  - **Description:** The UserProfile object which should be replaced in the webinos system.
- **successCallBack**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** SuccessCB
  - **Description:** Callback issued when the creating of the user is correctly finished.
- **errorCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** UserProfileErrorCB
  - **Description:** Callback issued if an error occurs during the opening. E.g. the userprofile id does not exist.

Code example

```
 // Obtain a single existing UserProfile object resulting from webinos.userprofile.find()
 var existingUserProfileObj = ...;

 // Modify some parameters as required. e.g. add a new phone number
 existingUserProfileObj.phoneNumbers.push({
 type: 'home',
 value: '654321'
 });

 //update the userprofile
 webinos.userprofile.replaceUserProfile(existingUserProfileObj, successCB, errorCB);
```

deleteUserProfile

The deleteUserProfile() method - Deletes an existing userprofile from the user in the webinos system.

Signature

```
void deleteUserProfile(in DOMString id, optional SuccessCB successCallBack, in UserProfileErrorCB errorCallback);
```

Parameters

- **id**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** The id of the existing object
- **successCallBack**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** SuccessCB
  - **Description:** Callback issued when the creating of the user is correctly finished.
- **errorCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** UserProfileErrorCB
  - **Description:** Callback issued if an error occurs during the opening. E.g. the userprofile id does not exist.

Code example

```
//delete the userprofile
webinos.userprofile.deleteUserProfile('xxx', successCB, errorCB);
```

2.2. UserProfile

The UserProfile interface. It is a userprofile specific extension to the interface Contact in the Contacts module.

```
        interface UserProfile : Contact {
                attribute DOMString? preferredUsername;
                attribute SocialNetworkProfile[]? socialProfiles;
        };
```

Attributes

`DOMString? preferredUsername`

preferredUsername of type DOMString

The preferred username of this user on sites that ask for a username (e.g. jsmarr or daveman692). This field may be useful for describing the owner (i.e. the value when

/@me/@self is requested), e.g. Consumers MAY wish to use this value to pre-populate a username for this user when signing up for a new service. See [[PORT] section 7.2.1].

`SocialNetworkProfile [] socialProfiles`

socialProfiles of type array of SocialNetworkProfile

The User profile on a social network provider.

2.3. SocialNetworkProfile

The SocialNetworkProfile interface

```
[NoInterfaceObject]
interface SocialNetworkProfile {
        attribute boolean pref;
        attribute DOMString? socialNetworkProvider;
        attribute DOMString? userId;
};
```

Attributes

`boolean pref`

pref of type boolean

This attribute indicates whether this instance of the SocialNetworkProfile is the preferred, or primary, value for the user. By default, the value is false.

`DOMString? socialNetworkProvider`

socialNetworkProvider of type DOMString

The identifier of the social network provider, for the purposes of sorting and filtering.

`DOMString? userId`

userId of type DOMString

The user's IDs in the social network, that is one or more elements that can be used to uniquely identify the user (i.e. userName, social network ID number, email). Usually chosen automatically, and usually numeric but sometimes alphanumeric, e.g. "12345" or "1Z425A".

2.4. UserProfileFindOptions

The UserProfileFindOptions interface describes the options that can be applied to userprofile searching. It inherits directly from ContactFindOptions and could be used to declare an filter for userprofile attributes. When a UserProfileFindOptions parameter is provided to the UserProfile find() operation, it should be processed according to the provisions detailed in Options Processing.

```
[NoInterfaceObject]
interface UserProfileFindOptions : ContactFindOptions {
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

2.5. UserProfileError

The UserProfileError interface. It is a userprofile specific extension to the interface ContactError in the Contacts module.

```
[NoInterfaceObject]
    interface UserProfileError : ContactError {
    const unsigned short USERPROFILE_NOT_EXIST = 101;
};
```

Constants

`unsigned short USERPROFILE_NOT_EXIST`

The userprofile does not exist in the webinos system.

2.6. UserProfileErrorCB

This is the wrapper interface for callbacks indicating failure of the createUserProfile(), updateUserProfile() and deleteUserProfile() operation.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface UserProfileErrorCB  {
    void onerror(UserProfileError error);
};
```

Methods

`onerror`

Callback on failure of a find() operation

Signature
`void onerror(UserProfileError error);`

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** UserProfileError
  - **Description:** The UserProfileError object capturing the type of the error.

Return value
void

2.7. UserProfileFindCB

This is the wrapper interface for callbacks indicating success of the find() operation.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface UserProfileFindCB {
    void onsuccess (UserProfile[] userProfileObjs);
};
```

Methods

`onsuccess`

Callback on success of a find() operation

Signature

```
void onsuccess(
                  UserProfile
                [] userProfileObjs);
```

Parameters

- **userProfileObjs**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** array
  - **Description:** An array of UserProfile objects resulting from the given UserProfile find() operation.

Return value

void

2.8. WebinosUserProfile

The WebinosUserProfile interface describes the part of the user profile API accessible through the webinos object.

```
[NoInterfaceObject] interface WebinosUserProfile {
        readonly attribute UserProfileInterface userprofile;
};
webinoscore::Webinos implements WebinosUserProfile;
```

Attributes

`readonly UserProfileInterface userprofile`

webinos.userprofile object.

This attribute is readonly.

3. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/userprofile

4. Full WebIDL

```
module userprofile {

        interface UserProfileInterface {

                caller void find(DOMString[] fields, UserProfileFindCB successCB, optional UserProfileErrorCB errorCB, optional UserProfileFindOptions options);

                void createUserProfile(in UserProfile userProfile, optional SuccessCB successCallBack, in UserProfileErrorCB errorCallback);

                void replaceUserProfile(in DOMString id, in UserProfile userProfile, optional SuccessCB successCallBack, in UserProfileErrorCB errorCallback);

                void deleteUserProfile(in DOMString id, optional SuccessCB successCallBack, in UserProfileErrorCB errorCallback);
        };

        interface UserProfile : Contact {
                attribute DOMString? preferredUsername;
                attribute SocialNetworkProfile[]? socialProfiles;
        };

        [NoInterfaceObject]
        interface SocialNetworkProfile {
                        attribute boolean pref;
                        attribute DOMString? socialNetworkProvider;
                        attribute DOMString? userId;
        };

    [NoInterfaceObject]
    interface UserProfileFindOptions : ContactFindOptions {
    };

    [NoInterfaceObject]
        interface UserProfileError : ContactError {
        const unsigned short USERPROFILE_NOT_EXIST = 101;
    };

    [Callback=FunctionOnly, NoInterfaceObject]
    interface UserProfileErrorCB  {
        void onerror(UserProfileError error);
    };

    [Callback=FunctionOnly, NoInterfaceObject]
    interface UserProfileFindCB {
        void onsuccess (UserProfile[] userProfileObjs);
    };

        [NoInterfaceObject] interface WebinosUserProfile {
                readonly attribute UserProfileInterface userprofile;
        };

        webinoscore::Webinos implements WebinosUserProfile;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The vehicle module

Webinos API Specifications

1 Jul 2011

Authors

- Simon Isenberg (BMW Forschung & Technik) <Simon.Isenberg@bmw.de>

Abstract

webinos Vehicle interface.

Summary of Methods

Interface	Method
VehicleError	
Address	
VehicleEvent	
LatLng	
POI	
SuccessCallback	void onSuccess()
ErrorCallback	void onError(VehicleError error)
VehicleDataHandler	void handleVehicleData(VehicleEvent data)
VehicleInterface	void get(DOMString vehicleDataId, VehicleDataHandler handler, ErrorCallback errorCB) void requestGuidance(SuccessCallback successCallback, ErrorCallback errorCallback, POI [] destinations)

Interface	Method
	void findDestination(DestinationCallback destinationCallback, ErrorCallback errorCallback, DOMString search)
Vehicle	
DestinationCallback	void handleResults(POI [] pois)
ClimateControlEvent	void initClimateControlEvent(boolean bubbles, boolean cancelable, DOMString zone, short desiredTemperature, boolean acStatus, short ventLevel, short ventMode)
ControlEvent	void initControlEvent(boolean bubbles, boolean cancelable, DOMString controlId, boolean active)
NavigationEvent	void initNavigationEvent(boolean bubbles, boolean cancelable, DOMString navigationEventId, Address destination)
ParkSensorsEvent	void initParkSensorsEvent(boolean bubbles, boolean cancelable, DOMString position, short left, short midLeft, short midRight, short right)
ShiftEvent	void initShiftEvent(boolean bubbles, boolean cancelable, short gear)
TripComputerEvent	void initTripComputerEvent(boolean bubbles, boolean cancelable, float averageConsumption1, float averageConsumption2, float averageSpeed1, float averageSpeed2, float tripDistance, float mileage, float range)

1. Introduction

The webinos vehicle API provides access to specific vehicle data. It is derived from W3C's DOM Level 3 Events model and defines event types for retrieving information about the vehicle including trip computer data, gears or park sensors. Furthermore it offers methods for interacting with the on-board navigation system. The geolocation, speed and acceleration can be retrieved using the geolocation and device orientation API.

The API gives access to vehicle data, which is available on the infotainment vehicle bus (e.g. MOST). The infotainment bus is the only access point for the headunit to receveive vehicle data (diagram on vehicle bus architecture). Some data from other busses (high/low speed CAN) are routed into the bus over the central gateway such as speed or gear (RPM is currently not provided on the MOST).

2. Interfaces

2.1. VehicleError

The interface defines the vehicle specific error

```
        interface VehicleError : Error{
                const short ACCESS_DENIED = 1;
                const short NOT_AVAILABLE = 2;
                const short UNKNOWN = 0;
        };
```

Constants

`short ACCESS_DENIED`

Constant describes that the access to the requested vehicle feature has been denied.

`short NOT_AVAILABLE`

Constant describes that the requested vehicle feature is not available.

`short UNKNOWN`

Constant describes that an unkown error occured while requestung a vehicle feature.

2.2. Address

This interface defines the address properties, which can be passed to the navigation system using the requestGuidance() function. The Address interface defined in the v2 of the W3C Geolocation API is used for this purpose.

```
        [NoInterfaceObject]
        interface Address{
                attribute DOMString country;
                attribute DOMString? region;
                attribute DOMString? county;
                attribute DOMString city;
                attribute DOMString street;
                attribute DOMString streetNumber;
                attribute DOMString? premises;
                attribute DOMString additionalInformation;
                attribute DOMString postalCode;
        };
```

Attributes

`DOMString country`

Attribute is specified by using the two-letter [ISO 3166-1] code.

`DOMString? region`

Attribute denotes the name of a country subdivision (e.g. the state name in the US).

`DOMString? county`

Attribute denotes the name of a land area within a larger region.

`DOMString city`

Attribute reflects the name of the city.

`DOMString street`

Attribute reflects the name of the street.

`DOMString streetNumber`

Attribute describes the location's street number.

`DOMString? premises`

Attribute denotes the details of the premises, such as a building name, block of flats, etc.

`DOMString additionalInformation`

Attribute contains other address details that are not captured by the rest of the attributes in this interface. Examples include a floor number in a building, an apartment number, the name of an office occupant, etc..

`DOMString postalCode`

Attribute reflects the postal code of the location (e.g. the zip code in the US).

2.3. VehicleEvent

The interface defines a generic event for vehicle data specific events.

```
[NoInterfaceObject]
interface VehicleEvent : Event{
};
```

2.4. LatLng

This interface defines the LatLng properties, which can be passed to the navigation system using the requestGuidance() function. The format is WGS84. *Note: The coordinate interface from the Geolocation API v2 includes attributes, which are not feasible for beeing handled by the navigation system (accurancy, alitude accurency, heading, speed). In some special cases (destination is on a bridge, which crosses another street) it might make sense to add the altitude to LatLng interface at a later stage.*

```
[NoInterfaceObject]
interface LatLng{
        attribute double latitude;
        attribute double longitude;
};
```

Attributes

`double latitude`

Attribute reflect the latitude of a geolocation in WGS84.

`double longitude`

Attribute reflect the Longitude of a geolocation in WGS84.

2.5. POI

This interface defines a Point of Interest (POI). The interface contains the name of a POI and its address and/or geolocation as a LatLng object. *Note: The W3C POI WG has published a first* *working draft for the POI handling. The draft focuses on a XML representation of a POI and does not seem handy for beeing handled by a navigation system.*

```
[NoInterfaceObject]
interface POI{
        attribute DOMString? name;
        attribute LatLng? position;
        attribute Address address;
};
```

Attributes

`DOMString? name`

Attribute denotes the name of the POI.

`LatLng? position`

Attribute reflects the geolocation of the POI as LatLng object.

`Address address`

Attribute denotes the address of the POI.

2.6. SuccessCallback

The interface defines the callback for a asynchronous function call insided the vehicle module.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface SuccessCallback{
        void onSuccess();
};
```

Methods

`onSuccess`

Method is triggered, if function has been succesfully called.

Signature

`void onSuccess();`

2.7. ErrorCallback

The interface defines the callback for a failed asynchronous function call inside the vehilce module.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface ErrorCallback{
        void onError(in VehicleError error);
};
```

Methods

`onError`

Method is triggered, if asychronous function call fails.

Signature

```
void onError(in VehicleError error);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** VehicleError
  - **Description:** contains information about the error.

2.8. VehicleDataHandler

The interface defines the callback method to receive vehicle data in a non-recurring and asynchronous way.

```
        interface VehicleDataHandler{
                void handleVehicleData(in VehicleEvent data);

        };
```

2.9. VehicleInterface

The interface defines general information about the vehicle and is the object, where the event listener for vehicle related data can be registered. The interface is accessible through the webinos.vehicle object.

```
        [NoInterfaceObject]
        interface VehicleInterface : EventTarget {
                const DOMString FUEL_UNLEADED = "unleaded";
                const DOMString FUEL_PREMIUM = "premium";
                const DOMString FUEL_DIESEL = "diesel";
                const DOMString TRANSMISSION_AUTOMATIC = "automatic";
                const DOMString TRANSMISSION_MANUAL = "manual";
                readonly attribute DOMString brand;
                readonly attribute DOMString model;
                readonly attribute DOMString year;
                readonly attribute DOMString fuel;
                readonly attribute DOMString transmission;
                void  get(DOMString  vehicleDataId,  VehicleDataHandler  handler,  in
ErrorCallback errorCB);
                void   requestGuidance(in   SuccessCallback   successCallback,   in
ErrorCallback errorCallback, POI[] destinations);
                void   findDestination(DestinationCallback   destinationCallback,   in
ErrorCallback errorCallback, DOMString search);
        };
```

Constants

`DOMString FUEL_UNLEADED`

Constant defines the fuel type unleaded.

`DOMString FUEL_PREMIUM`

Constant defines the fuel type premium.

`DOMString FUEL_DIESEL`

Constant defines the fuel type diesel.

`DOMString TRANSMISSION_AUTOMATIC`

Constant defines the transmission type automatic.

`DOMString TRANSMISSION_MANUAL`

Constant defines the transmission type manual.

Attributes

`readonly DOMString brand`

Attribute denotes brand name of the vehicle.

This attribute is readonly.

`readonly DOMString model`

Attribute reflects model name of the vehicle.

This attribute is readonly.

`readonly DOMString year`

Attribute denotes production year of the vehicle.

This attribute is readonly.

`readonly DOMString fuel`

Attribute reflects fuel type of the vehicle.

This attribute is readonly.

`readonly DOMString transmission`

Attribute denotes transmission type of the vehicle.

This attribute is readonly.

Methods

`get`

Method allows to request vehicle data in a non-recurring way and is independant from value changes (cf. events). The same identifiers are used for vehicle data as well as for the different vehicle events (ClimateControlEvent, ControlEvent, NavigationEvent, ParkSensorsEvent, ShiftEvent).

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Signature

```
void   get(DOMString   vehicleDataId,   VehicleDataHandler   handler,   in
ErrorCallback errorCB);
```

Parameters

- **vehicleDataId**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** Parameter specifies the type of data, which shall be retrieved. The identifiers for the different data types are defined in the different vehicle event interfaces (ClimateControlEvent, ControlEvent, NavigationEvent, ParkSensorsEvent, ShiftEvent). The identifiers for climate control data are defined in constants CLIMATE_*, for control data in constants LIGHTS_* and WHIPER_*, for navigation data in constants DESTINATION_*, for park sensors data in constants PARKSENSORS_*, for shift data in constant SHIFT and for trip computer data in constant TRIPCOMPUTER.
- **handler**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** VehicleDataHandler
    - **Description:** Parameter specifies the function to handle the result.
- **errorCB**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** ErrorCallback
    - **Description:** Parameter specifies the callback function in case of an error.

Code example

```
webinos.vehicle.get(webinos.vehicle.ClimateControlEvent.CLIMATE_ALL, dataHandler);
function dataHandler(data){
        if(data.acStatus){
                console.log("Airconditioning is on");
                if(data.desiredTemperature < 19){
                        console.log("This is not so good for your health");
                                }
        }
}
```

`requestGuidance`

Parameter sets the given POIs as the next destinations for the build-in navigation system. The method handles intermediate stops. The last POI in the array is the final destination.

Signature

```
void    requestGuidance(in    SuccessCallback    successCallback,    in
ErrorCallback errorCallback,

               POI

             [] destinations);
```

Parameters

- **successCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** SuccessCallback
  - **Description:** callback, if the POIs are succesully transferred.
- **errorCallback**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ErrorCallback
  - **Description:** callback, if the address could not be transferred.
- **destinations**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** array
  - **Description:** in form of an POI array. The last POI in the array is the destination point. The other POIs are intermediate stops along the route.

Code example

```
        var destinations =new Array();
        destination.push({name:"BMW AG", address : {street:"Petuelring", streetNumber:
"130", postalCode: "80788", city: "MÜNCHEN", country: "DE"}});
      destination.push({name:"BMW  Forschung  und  Technik",  address:{street:  "Hanauer
Strasse", streetNumber: "46", postalCode: "80992", city: "MÜNCHEN", country: "DE"}});
      webinos.vehicle.requestGuidance(succesCB, null, destinations);

webinos.vehicle.addEventListener(webinos.vehicle.NavigationEvent.DESTINATION_REACHED,
handleDestinations, false);

        function handleDestinations(event){
```

```
    if (event.address.street == destination[0].address.street) {
         console.log("Reached the HQ");
    } else if (event.address.street == destination[1].address.street) {
          console.log("Reached the research center");
    }
  }
```

findDestination

Queries the navigation system to retrieve POIs for a given search string.

Signature

```
void findDestination(DestinationCallback destinationCallback, in ErrorCallback errorCallback, DOMString search);
```

Parameters

- **destinationCallback**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** DestinationCallback
    - **Description:** callback to handle the results to the search string.
- **errorCallback**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** ErrorCallback
    - **Description:** callback to handle errors.
- **search**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** search string for resolving a address.

Code example

```
  var destinations =new Array();
webinos.vehicle.findDestination(destinationCB, errorCB,"BMW");
function destinationCB(pois){
  if(destinations.length > 0){
  webinos.requestGuidance(successCB, errorCB, destinations);
  else{
          console.log("No POI found");
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
        }
}
```

2.10. Vehicle

The Vehicle interface describes the part of the Vehicle API accessible through the webinos object.

```
        [NoInterfaceObject] interface Vehicle {
                readonly attribute VehicleInterface vehicle;
        };
        webinoscore::Webinos implements Vehicle;
```

2.11. DestinationCallback

The interface defines the result callback for the asynchronous findDestination method.

```
        [Callback=FunctionOnly, NoInterfaceObject]
        interface DestinationCallback {
                 void handleResults(in POI[] pois);
        };
```

Methods

handleResults

Function is called, when the results for a POI search are retrieved.

Signature

```
void handleResults(in

                POI

              [] pois);
```

Parameters

- **pois**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** array
    - **Description:** Parameter provides an array of POIs.

2.12. ClimateControlEvent

The interface defines a climate control event. The event provides information about changes to the climate control system inside the vehicle.

```
        [NoInterfaceObject]
        interface ClimateControlEvent : VehicleEvent{
                const DOMString CLIMATE_ALL = "climate-all";
                const DOMString CLIMATE_DRIVER = "climate-driver";
                const DOMString CLIMATE_PASSENGER_FRONT = "climate-passenger-front";
                const DOMString CLIMATE_PASSENGER_REAR_LEFT = "climate-passenger-rear-
left";
                const  DOMString  CLIMATE_PASSENGER_REAR_RIGHT  =  "climate-passenger-
rear-right";
                readonly attribute DOMString zone;
                readonly attribute unsigned short desiredTemperature;
                readonly attribute boolean acStatus;
                readonly attribute unsigned short ventLevel;
                readonly attribute boolean ventMode;
                void  initClimateControlEvent(boolean  bubbles,  boolean  cancelable,
DOMString zone, short desiredTemperature, boolean acStatus, short ventLevel, short
ventMode);
        };
```

Code example

```
 webinos.vehicle.addEventListener("climate", climateHandler, null);
 function climateHandler(data){
                console.log(data.zone   +   "   desired   temperature   is   "   +
data.desiredTemperature + "° C");
        }
```

Constants

`DOMString CLIMATE_ALL`

Constant defines the single climate zone. This constant is used as an identifier for a ClimateControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the overall zone.

`DOMString CLIMATE_DRIVER`

Constant describes the climate zone of the driver. This constant is used as an identifier for a ClimateControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the driver zone.

`DOMString CLIMATE_PASSENGER_FRONT`

Constant defines the climate zone of the passenger seat in the front. This constant is used as an identifier for a ClimateControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the passenger zone in the front.

`DOMString CLIMATE_PASSENGER_REAR_LEFT`

Constant defines the climate zone of the rear set passenger seat on the left. This constant is used as an identifier for a ClimateControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the passenger zone in the rear on the left side.

`DOMString CLIMATE_PASSENGER_REAR_RIGHT`

Constant defines the climate zone of the rear set passenger seat on the right. This constant is used as an identifier for a ClimateControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the passenger zone in the rear on the right side.

Attributes

`readonly DOMString zone`

Attribute defines the zone of the climate control event. The value of this attribute is defined in the constants CLIMATE_*.

This attribute is readonly.

`readonly unsigned short desiredTemperature`

Attribute defines the desired temperature in degree celsius.

This attribute is readonly.

`readonly boolean acStatus`

Attribute defines, if the AC switched on or not.

This attribute is readonly.

`readonly unsigned short ventLevel`

Attribute defines the level of the vents. This value can be 1-9.

This attribute is readonly.

`readonly boolean ventMode`

Attribute defines if the vent is used in automatic mode or not.

This attribute is readonly.

Methods

`initClimateControlEvent`

Method is used to set initial values of a climate control event.

Signature

```
void initClimateControlEvent(boolean bubbles, boolean cancelable, DOMString zone, short desiredTemperature, boolean acStatus, short ventLevel, short ventMode);
```

Parameters

- **bubbles**
  - **Optional:** No.
  - **Nullable**: No

  - **Type:** boolean
  - **Description:** True if event bubbles.
- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event is cancelable.
- **zone**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** zone where event climate settings have been changed.
- **desiredTemperature**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** short
  - **Description:** desired temperature in degree celsius.
- **acStatus**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** true if the air conditioning is running.
- **ventLevel**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** short
  - **Description:** level of the vent.
- **ventMode**
  - **Optional:** No.

- **Nullable**: No
- **Type:** short
- **Description:** true, if the vent is in automatic mode.

2.13. ControlEvent

The interface defines a control event. The event signals a change for a control unit inside the vehicle (e.g. lights, wiper, etc.). The identifiers for the different control events are defined in the constants LIGHTS_* and WHIPER_*

```
        interface ControlEvent : VehicleEvent{
                const DOMString LIGHTS_FOG_FRONT = "lights-fog-front";
                const DOMString LIGHTS_FOG_REAR = "lights-fog-rear";
                const DOMString LIGHTS_SIGNAL_LEFT = "lights-signal-left";
                const DOMString LIGHTS_SIGNAL_RIGHT = "lights-signal-right";
                const DOMString LIGHTS_SIGNAL_WARN = "lights-signal-warn";
                const DOMString LIGHTS_PARKING = "lights-parking";
                const DOMString LIGHTS_HIBEAM = "lights-hibeam";
                const DOMString LIGHTS_HEAD = "lights-head";
                const DOMString WHIPER_FRONT_WASH = "whiper-front-wash";
                const DOMString WHIPER_REAR_WASH = "whiper-rear-wash";
                const DOMString WHIPER_AUTOMATIC = "whiper-automatic";
                const DOMString WHIPER_FRONT_ONCE = "whiper-front-once";
                const DOMString WHIPER_REAR_ONCE = "whiper-front-once";
                const DOMString WHIPER_FRONT_LEVEL1 = "whiper-front-level1";
                const DOMString WHIPER_FRONT_LEVEL2 = "whiper-front-level2";
                readonly attribute DOMString conrolId;
                readonly attribute boolean active;
                void  initControlEvent(boolean  bubbles,  boolean  cancelable,  DOMString
controlId, boolean active);
        };
Code example
        webinos.vehicle.addEventListener("lights-hibeam", lightHandler, false);
      function lightHandler(cEvent){
                if(cEvent.controlId == "lights-hibeam"){
                        if(cEvent.active == true){
                                console.log("Hibeam turned on");
                        }else{
                                console.log("Hibeam turned off");
                        }
                }
        }
```

Constants

`DOMString LIGHTS_FOG_FRONT`

Constant indicates a change for the fog light in the front. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of fogs light in the front.

`DOMString LIGHTS_FOG_REAR`

Constant indicates a change for the fog light in the rear. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of fogs light in the rear.

`DOMString LIGHTS_SIGNAL_LEFT`

Constant indicates a change for left turn signal. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the left signal.

`DOMString LIGHTS_SIGNAL_RIGHT`

Constant indicates a change for right turn signal. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the right signal.

`DOMString LIGHTS_SIGNAL_WARN`

Constant indicates a change for warn signal. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the warn signal.

`DOMString LIGHTS_PARKING`

Constant indicates a change for the parking lights. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the parking light.

`DOMString LIGHTS_HIBEAM`

Constant indicates a change for the hibeam. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the lights hibeam.

`DOMString LIGHTS_HEAD`

Constant indicates a change for the headlight. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the head light.

`DOMString WHIPER_FRONT_WASH`

Constant indicates front window is beeing washed. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the whiper front wash.

`DOMString WHIPER_REAR_WASH`

Constant indicates rear window is beeing washed. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the whiper rear wash.

`DOMString WHIPER_AUTOMATIC`

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Constant indicates whiper is in automatic mode. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the automatic whiper mode.

`DOMString WHIPER_FRONT_ONCE`

Constant indicates front whiper is beeing used once. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the whiper single mode in the front.

`DOMString WHIPER_REAR_ONCE`

Constant indicates rear whiper is beeing used once. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the whiper single mode in the rear.

`DOMString WHIPER_FRONT_LEVEL1`

constant indicates front whiper is on level 1. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the whiper on level 1 in the front.

`DOMString WHIPER_FRONT_LEVEL2`

constant indicates front whiper is on level 2. This constant is used as an identifier for a ControlEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the whiper on level 2 in the front.

Attributes

`readonly DOMString conrolId`

Attribute describes the source of the event. The value of the attribute is defined in the constants LIGHTS_* and WHIPER_*.

This attribute is readonly.

`readonly boolean active`

Attribute describes the status of the control unit.

This attribute is readonly.

Methods

`initControlEvent`

Method sets initial values of a control event.

Signature

```
void initControlEvent(boolean bubbles, boolean cancelable, DOMString
controlId, boolean active);
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Parameters

- **bubbles**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event bubbles.
- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event cancelable.
- **controlId**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** specifies the control unit.
- **active**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** specifies if the unit is activated or not.

2.14. NavigationEvent

The interface defines the navigation event. The identifiers for the different navigation events are defined in the constants DESTINATTION_*.

```
        interface NavigationEvent : VehicleEvent{
                const DOMString DESTINATION_REACHED = "destination-reached";
                const DOMString DESTINATION_CHANGED = "destination-changed";
                const DOMString DESTINATION_CANCELLED = "destination-cancelled";

                readonly attribute DOMString type;
                readonly attribute Address address;

                void    initNavigationEvent(boolean    bubbles,    boolean    cancelable,
DOMString navigationEventId, Address destination);
        };
```

Constants

`DOMString DESTINATION_REACHED`

Constant defines the event that the destination has been reached. This constant is used as an identifier for a NavigationEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for destination reached.

`DOMString DESTINATION_CHANGED`

Constant defines the event that a new destination has been set. This constant is used as an identifier for a NavigationEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for destination changed.

`DOMString DESTINATION_CANCELLED`

Constant defines the event that the navigation to a destination has been cancelled. This constant is used as an identifier for a NavigationEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for a cancelation of guidance to a destination.

Attributes

`readonly DOMString type`

Attribute defines the type of the navigation event. The type can either be "destination-reached", "destination-changed" or "destination-cancelled".

This attribute is readonly.

`readonly Address address`

Attribute defines for which address the event occured.

This attribute is readonly.

Methods

`initNavigationEvent`

Method sets initial values of a navigation event.

Signature

```
void initNavigationEvent(boolean bubbles, boolean cancelable, DOMString
navigationEventId, Address destination);
```

Parameters

- **bubbles**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean

- **Description:** True if event bubbles.

- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event cancelable.

- **navigationEventId**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** Sensor type.

- **destination**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** Address
  - **Description:** destination for which the event occured.

2.15. ParkSensorsEvent

This interface defines an event related to the built-in park sensors. The identifiers for the different events are defined in the constants PARKSENSENSORS_FRONT and PARKSENSENSORS_REAR. A listener can be registered by vehicle.addEventLister("parksensor-front",listener,false).

```
        interface ParkSensorsEvent : VehicleEvent{
                const DOMString PARKSENSENSORS_FRONT = "parksensors-front";
                const DOMString PARKSENSENSORS_REAR = "parksensors-rear";
                readonly attribute DOMString position;
                readonly attribute unsigned short left;
                readonly attribute unsigned short midLeft;
                readonly attribute unsigned short midRigth;
                readonly attribute unsigned short rigth;
                void   initParkSensorsEvent(boolean   bubbles,   boolean   cancelable,
DOMString position, short left, short midLeft, short midRight, short right);
        };
Code example
        webinos.vehicle.addEventListener("parksensor-front", psHandler, false);
        webinos.vehicle.addEventListener("parksensor-rear", psHandler, false);
        function psHandler(psEvent){
                if(psEvent.left == 20){
                        console.log("obstacle on the left in" + psEvent.position + "
is close");
```

```
                }
        }
```

Constants

`DOMString PARKSENSENSORS_FRONT`

Constant defines that the event was emitted by the park sensors in the front. This constant is used as an identifier for a ParkSensorEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the park sensors in the front.

`DOMString PARKSENSENSORS_REAR`

Constant defines that the event was emitted by the park sensors in the rear. This constant is used as an identifier for a ParkSensorEvent and a non-recurring vehicle data request using the method webinos.vehicle.get() for the status of the park sensors in the rear.

Attributes

`readonly DOMString position`

Attribute defines the position of the sensor. The value of the attribute is either PARKSENSENSORS_FRONT or PARKSENSENSORS_REAR.

This attribute is readonly.

`readonly unsigned short left`

Attribute reflects the destination to an object sensed by the sensor on the left side in centimeters. Minimum distance is 20 centimeters. Maximum distance is 250 centimeters. A value of -1 indiactes that no object has been sensed.

This attribute is readonly.

`readonly unsigned short midLeft`

Attribute reflects the destination to an object sensed by the sensor on the middle left side in centimeters. Minimum distance is 20 centimeters. Maximum distance is 250 centimeters. A value of -1 indiactes that no object has been sensed.

This attribute is readonly.

`readonly unsigned short midRigth`

Attribute reflects the destination to an object sensed by the sensor on the middle right side in centimeters. Minimum distance is 20 centimeters. Maximum distance is 250 centimeters. A value of -1 indiactes that no object has been sensed.

This attribute is readonly.

`readonly unsigned short rigth`

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Attribute reflects the destination to an object sensed by the sensor on the rigth side in centimeters. Minimum distance is 20 centimeters. Maximum distance is 250 centimeters. A value of -1 indiactes that no object has been sensed.

This attribute is readonly.

Methods

`initParkSensorsEvent`

Method sets initial values of a park sensors event.

Signature

```
void initParkSensorsEvent(boolean bubbles, boolean cancelable, DOMString position, short left, short midLeft, short midRight, short right);
```

Parameters

- **bubbles**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event bubbles.
- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event cancelable.
- **position**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** position of the sensors: front or rear.
- **left**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** short

- **Description:** data from the left sensor.

- **midLeft**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** short
  - **Description:** data from the middle left sensor.
- **midRight**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** short
  - **Description:** data from the middle right sensor.
- **right**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** short
  - **Description:** data from the right sensor.

2.16. ShiftEvent

This interface defines a shift event. A listener can be registered by vehicle.addEventLister("shift",listener,false).

```
        interface ShiftEvent : VehicleEvent{
                const DOMString SHIFT = "shift";
                const short GEAR_ONE = 1;
                const short GEAR_TWO = 2;
                const short GEAR_THREE = 3;
                const short GEAR_FOUR = 4;
                const short GEAR_FIFE = 5;
                const short GEAR_SIX = 6;
                const short GEAR_SEVEN = 7;
                const short GEAR_EIGHT = 8;
                const short GEAR_REVERSE = -1;
                const short GEAR_NEUTRAL = 0;
                const short GEAR_PARKING = -2;
                readonly attribute short gear;
                void initShiftEvent(boolean bubbles, boolean cancelable, short gear);
        };
```

Code example

```
// registering an Event for a Shift
webinos.vehicle.addEventListener("shift", shiftHandler);
function shiftHandler(e){
  document.getElementById("info").innerHTML = e.gear;
 webinos.vehicle.removeEventListener("shift", shiftHandler);
}
```

Constants

`DOMString SHIFT`

Constant defines shift event. This constant is used as an identifier for a shift event and a non-recurring vehicle data request using the method webinos.vehicle.get() for the gear.

`short GEAR_ONE`

Constant defines the first gear.

`short GEAR_TWO`

Constant defines the second gear.

`short GEAR_THREE`

Constant defines the third gear.

`short GEAR_FOUR`

Constant defines the fourth gear.

`short GEAR_FIFE`

Constant defines the fifth gear.

`short GEAR_SIX`

Constant defines the sixth gear.

`short GEAR_SEVEN`

Constant defines the seventh gear.

`short GEAR_EIGHT`

Constant defines the eighth gear.

`short GEAR_REVERSE`

Constant defines the reverse gear.

`short GEAR_NEUTRAL`

Constant defines the neutral gear.

`short GEAR_PARKING`

Constant defines the parking gear.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Attributes

`readonly short gear`

Attribute represents the current gear of the vehicle.

This attribute is readonly.

Methods

`initShiftEvent`

Method sets initial values a shift event.

Signature

```
void initShiftEvent(boolean bubbles, boolean cancelable, short gear);
```

Parameters

- **bubbles**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event bubbles.
- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event cancelable.
- **gear**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** short
  - **Description:** current gear of the engine.

2.17. TripComputerEvent

The interface defines a trip computer event. A listener can be registered by vehicle.addEventLister("tripcomputer",listener,false).

```
        interface TripComputerEvent : VehicleEvent{
                const DOMString TRIPCOMPUTER = "tripcomputer";
                readonly attribute float averageConsumption1;
                readonly attribute float averageConsumption2;
                readonly attribute float averageSpeed1;
                readonly attribute float averageSpeed2;
                readonly attribute float tripDistance;
                readonly attribute float milage;
                readonly attribute float range;
                void initTripComputerEvent(boolean bubbles, boolean cancelable, float
averageConsumption1, float averageConsumption2, float averageSpeed1, float
averageSpeed2, float tripDistance, float mileage, float range);
        };
```

Code example

```
var latestConsumption = 0;
webinos.vehicle.addEventListener("tripcomputer", tripDataHandler, false);

function tripDataHandler(data){
        //Calculating, if the fuel efficiency increased, since the last update
        var gap = latestConsumption - data.averageConsumption1;
        if(gap < 0){
            console.log("Thumbs down. You decreased your fuel efficiency");
        } else if (gap > 0){
            console.log("Thumbs up. You increased your fuel efficiency");
        } else {
            console.log("nothing changed");
        }
        latestConsumption = data.averageConsumption1;

        if (data.range < 20){
            console.log("You really need " + webinos.vehicle.fuel + "soon.");
        }
}
```

Constants

`DOMString TRIPCOMPUTER`

Constant defines a trip computer event. This constant is used as an identifier for a trip computer event and a non-recurring trip data request using the method webinos.vehicle.get() for trip computer data.

Attributes

`readonly float averageConsumption1`

Attrubute reflects the average consumption 1 of the vehicle in l/100kilometers (resets automatically after a trip).

This attribute is readonly.

`readonly float averageConsumption2`

Attrubute reflects the average consumption 2 of the vehicle in l/100kilometers (resets on driver's demand).

This attribute is readonly.

`readonly float averageSpeed1`

Attribute reflects the average speed of the vehicle in kilometers per hour (resets automatically after a trip).

This attribute is readonly.

`readonly float averageSpeed2`

Attrubute reflects average speed of the vehicle in kilometers per hour (resets on driver's demand).

This attribute is readonly.

`readonly float tripDistance`

Attrubute reflects trip distance in kilometers.

This attribute is readonly.

`readonly float milage`

Attrubute reflects milage in kilometers.

This attribute is readonly.

`readonly float range`

Attrubute reflects the range of the vehicle in kilometers.

This attribute is readonly.

Methods

`initTripComputerEvent`

Method sets the initial values of a trip computer event.

Signature

```
void initTripComputerEvent(boolean bubbles, boolean cancelable, float
averageConsumption1, float averageConsumption2, float averageSpeed1,
float averageSpeed2, float tripDistance, float mileage, float range);
```

Parameters

- **bubbles**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event bubbles.

- **cancelable**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** True if event cancelable.
- **averageConsumption1**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** float
  - **Description:** average consumption 1 of the vehicle in l/100kilometers.
- **averageConsumption2**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** float
  - **Description:** average consumption 2 of the vehicle in l/100kilometers.
- **averageSpeed1**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** float
  - **Description:** average speed 1 of the vehicle in kilometers per hour.
- **averageSpeed2**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** float
  - **Description:** average speed 2 of the vehicle in kilometers per hour.
- **tripDistance**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** float

    - **Description:** distance of the current trip in kilometers.

- **mileage**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** float
    - **Description:** overall driven distance in kilometers.
- **range**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** float
    - **Description:** range of the vehicle in kilometers.

3. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://webinos.org/api/vehicle

Identifies the light sensor type.

http://webinos.org/api/vehicle.climate

Identifies vehicle data related to climate controls.

http://webinos.org/api/vehicle.navigation

Identifies the navigation module of a vehicle.

http://webinos.org/api/vehicle.parksensors

Identifies park sensor modules of a vehicle.

http://webinos.org/api/vehicle.tripcomputer

Identifies the trip computer information of a vehicle.

http://webinos.org/api/vehicle.controls

Identifies control data (e.g. whipers and lights) of a vehicle

4. Full WebIDL

```
module vehicle{
        interface VehicleError : Error{
                const short ACCESS_DENIED = 1;
                const short NOT_AVAILABLE = 2;
                const short UNKNOWN = 0;
        };
        [NoInterfaceObject]
        interface Address{
                attribute DOMString country;
                attribute DOMString? region;
                attribute DOMString? county;
                attribute DOMString city;
                attribute DOMString street;
                attribute DOMString streetNumber;
                attribute DOMString? premises;
                attribute DOMString additionalInformation;
                attribute DOMString postalCode;
        };

        [NoInterfaceObject]
        interface VehicleEvent : Event{

        };
        [NoInterfaceObject]
        interface LatLng{
                attribute double latitude;
                attribute double longitude;
        };
        [NoInterfaceObject]
        interface POI{
                attribute DOMString? name;
                attribute LatLng? position;
                attribute Address address;
        };

        [Callback=FunctionOnly, NoInterfaceObject]
        interface SuccessCallback{
                void onSuccess();
        };
        [Callback=FunctionOnly, NoInterfaceObject]
        interface ErrorCallback{
                void onError(in VehicleError error);
        };

        interface VehicleDataHandler{
                void handleVehicleData(in VehicleEvent data);

        };

        [NoInterfaceObject]
        interface VehicleInterface : EventTarget {
                const DOMString FUEL_UNLEADED = "unleaded";
                const DOMString FUEL_PREMIUM = "premium";
                const DOMString FUEL_DIESEL = "diesel";
                const DOMString TRANSMISSION_AUTOMATIC = "automatic";
                const DOMString TRANSMISSION_MANUAL = "manual";
                readonly attribute DOMString brand;
                readonly attribute DOMString model;
                readonly attribute DOMString year;
                readonly attribute DOMString fuel;
                readonly attribute DOMString transmission;
                void  get(DOMString  vehicleDataId,  VehicleDataHandler  handler,  in
ErrorCallback errorCB);
                void   requestGuidance(in   SuccessCallback   successCallback,   in
ErrorCallback errorCallback, POI[] destinations);
                void   findDestination(DestinationCallback   destinationCallback,   in
ErrorCallback errorCallback, DOMString search);
        };

        [NoInterfaceObject] interface Vehicle {
                readonly attribute VehicleInterface vehicle;
        };
        webinoscore::Webinos implements Vehicle;
        [Callback=FunctionOnly, NoInterfaceObject]
        interface DestinationCallback {
                 void handleResults(in POI[] pois);
        };
        [NoInterfaceObject]
        interface ClimateControlEvent : VehicleEvent{
                const DOMString CLIMATE_ALL = "climate-all";
                const DOMString CLIMATE_DRIVER = "climate-driver";
                const DOMString CLIMATE_PASSENGER_FRONT = "climate-passenger-front";
                const DOMString CLIMATE_PASSENGER_REAR_LEFT = "climate-passenger-rear-
left";
                const  DOMString  CLIMATE_PASSENGER_REAR_RIGHT  =  "climate-passenger-
rear-right";
                readonly attribute DOMString zone;
                readonly attribute unsigned short desiredTemperature;
                readonly attribute boolean acStatus;
                readonly attribute unsigned short ventLevel;
                readonly attribute boolean ventMode;
                void  initClimateControlEvent(boolean  bubbles,  boolean  cancelable,
DOMString  zone,  short  desiredTemperature,  boolean  acStatus,  short  ventLevel,  short
ventMode);
        };
        interface ControlEvent : VehicleEvent{
                const DOMString LIGHTS_FOG_FRONT = "lights-fog-front";
                const DOMString LIGHTS_FOG_REAR = "lights-fog-rear";
                const DOMString LIGHTS_SIGNAL_LEFT = "lights-signal-left";
                const DOMString LIGHTS_SIGNAL_RIGHT = "lights-signal-right";
                const DOMString LIGHTS_SIGNAL_WARN = "lights-signal-warn";
                const DOMString LIGHTS_PARKING = "lights-parking";
                const DOMString LIGHTS_HIBEAM = "lights-hibeam";
                const DOMString LIGHTS_HEAD = "lights-head";
                const DOMString WHIPER_FRONT_WASH = "whiper-front-wash";
                const DOMString WHIPER_REAR_WASH = "whiper-rear-wash";
                const DOMString WHIPER_AUTOMATIC = "whiper-automatic";
                const DOMString WHIPER_FRONT_ONCE = "whiper-front-once";
                const DOMString WHIPER_REAR_ONCE = "whiper-front-once";
                const DOMString WHIPER_FRONT_LEVEL1 = "whiper-front-level1";
                const DOMString WHIPER_FRONT_LEVEL2 = "whiper-front-level2";
                readonly attribute DOMString conrolId;
                readonly attribute boolean active;
                void  initControlEvent(boolean  bubbles,  boolean  cancelable,  DOMString
controlId, boolean active);
        };
        interface NavigationEvent : VehicleEvent{
                const DOMString DESTINATION_REACHED = "destination-reached";
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
                const DOMString DESTINATION_CHANGED = "destination-changed";
                const DOMString DESTINATION_CANCELLED = "destination-cancelled";

                readonly attribute DOMString type;
                readonly attribute Address address;

                void   initNavigationEvent(boolean   bubbles,   boolean   cancelable,
DOMString navigationEventId, Address destination);
        };
        interface ParkSensorsEvent : VehicleEvent{
                const DOMString PARKSENSENSORS_FRONT = "parksensors-front";
                const DOMString PARKSENSENSORS_REAR = "parksensors-rear";
                readonly attribute DOMString position;
                readonly attribute unsigned short left;
                readonly attribute unsigned short midLeft;
                readonly attribute unsigned short midRigth;
                readonly attribute unsigned short rigth;
                void   initParkSensorsEvent(boolean   bubbles,   boolean   cancelable,
DOMString position, short left, short midLeft, short midRight, short right);
        };
        interface ShiftEvent : VehicleEvent{
                const DOMString SHIFT = "shift";
                const short GEAR_ONE = 1;
                const short GEAR_TWO = 2;
                const short GEAR_THREE = 3;
                const short GEAR_FOUR = 4;
                const short GEAR_FIFE = 5;
                const short GEAR_SIX = 6;
                const short GEAR_SEVEN = 7;
                const short GEAR_EIGHT = 8;
                const short GEAR_REVERSE = -1;
                const short GEAR_NEUTRAL = 0;
                const short GEAR_PARKING = -2;
                readonly attribute short gear;
                void initShiftEvent(boolean bubbles, boolean cancelable, short gear);
        };
        interface TripComputerEvent : VehicleEvent{
                const DOMString TRIPCOMPUTER = "tripcomputer";
                readonly attribute float averageConsumption1;
                readonly attribute float averageConsumption2;
                readonly attribute float averageSpeed1;
                readonly attribute float averageSpeed2;
                readonly attribute float tripDistance;
                readonly attribute float milage;
                readonly attribute float range;
                void initTripComputerEvent(boolean bubbles, boolean cancelable, float
averageConsumption1,   float   averageConsumption2,   float   averageSpeed1,   float
averageSpeed2, float tripDistance, float mileage, float range);
        };
};
```

APIs: The webinoscore module

Webinos API Specifications

17 May 2011

Authors

- Claes Nilsson <claes1.nilsson@sonyericsson.com>

Abstract

Webinos core interfaces

Summary of Methods

Interface	Method
WebinosObject	
Webinos	

1. Introduction

This specification defines the common interface from which all Webinos APIs are can be accessed as well as several interfaces that are commonly reused.

This version of the specification defines:
- The core Webinos interface. In this version this interface is part of the window global object but this has to be discussed. W3C DAP hangs the APIs on Device on Navigator.

It is to be considered if more common interfaces should be included in this specification, for example:
- A common PendingOperation interface
- Methods to retrieve lists of available and activated feature
- Generic error interface
- Generic success callback
- Generic error callback
- Common array types

2. Interfaces

2.1. WebinosObject

Webinos object

```
interface WebinosObject {
  readonly attribute Webinos webinos;
};
Window implements WebinosObject;
```

Defines that the webinos interface is part of the window global object.

2.2. Webinos

Webinos interface

```
interface Webinos {
};
```

The is the Webinos root interface and is initially defined as an empty interface on which the various Webinos APIs that are defined elsewhere graft themselves. A user agent supporting the Webinos interface must do so according to the following WebIDL [WEBIDL] definition.

3. Features

4. Full WebIDL

```
module webinoscore {


  interface WebinosObject {
    readonly attribute Webinos webinos;
  };

  Window implements WebinosObject;

  interface Webinos {
  };
};
```

APIs: The widget module

Webinos API Specifications

29 Jun 2011

Authors

- Andre Paul andre.paul@fokus.fraunhofer.de;

Abstract

Webinos widget interfaces

Summary of Methods

Interface	Method
NotifySuccessCallback	void onSuccess(DOMString id)
NotifyErrorCallback	void onError(DOMString id)
DeploymentSuccessCallback	void onSuccess(DOMString childID, DOMString serviceID)
DeploymentErrorCallback	void onError(DeploymentError error)
DeploymentError	
Widget	void exit() void hide() boolean isHidden() void notify(NotifySuccessCallback onSuccess, NotifyErrorCallback onError, DOMString title, DOMString shortDescription, DOMString id, DOMString icon) void cancelNotify(DOMString id) void onDestroy() void onBackground() void onForeground()

Interface	Method
	void onStop() void onStart() void deployChild(DeploymentSuccessCallback onSuccess, DeploymentErrorCallback onError, DOMString childApplicationID, boolean local)
WindowWidget	

1. Introduction

This specification defines the common widget interface. The webinos application packaging is based on W3C Widget Specifications, thus, the interface definition is also based on W3C. Namely W3C Widget Interface (http://www.w3.org/TR/2011/WD-widgets-apis-20110607/). This specification recaptures the W3C specification while adding webinos specific extensions.

2. Interfaces

2.1. NotifySuccessCallback

Callback for successfull notifications

```
   [NoInterfaceObject] interface NotifySuccessCallback {
     void onSuccess(in DOMString id);
  };
```

Methods

`onSuccess`

Accepted Notification.

Signature

```
void onSuccess(in DOMString id);
```

Called if an event was accepted by the user. If provided, the notification id is provided to link the success callback to a specific notification request.

Parameters

- **id**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** the optional id of the notification request or null if no id was provided.

2.2. NotifyErrorCallback

Callback for failed notifications

```
   [NoInterfaceObject] interface NotifyErrorCallback {
     void onError(in DOMString id);
 };
```

Methods

`onError`

Discarded Notification.

Signature

```
void onError(in DOMString id);
```

Called if an event was not accepted by the user. If provided, the notification id is passed in to link the error to a specific notification request.

Parameters

- **id**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** the optional id of the notification request or null if no id was provided.

2.3. DeploymentSuccessCallback

Callback for successfull installations

```
   [NoInterfaceObject] interface DeploymentSuccessCallback {
     void onSuccess(in DOMString childID, in DOMString serviceID);
   };
```

Methods

`onSuccess`

Called when an application was successfully deployed.

Signature

```
void onSuccess(in DOMString childID, in DOMString serviceID);
```

Called when an application was successfully deployed on another device using deployChild.

Parameters

- **childID**

- **Optional:** No.
- **Nullable**: No
- **Type:** DOMString
- **Description:** is the application id which was used during deployChild and declared in the manifest

- **serviceID**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** is the unique application instance id that can be used to explicitly address the deployed service within webinos service discovery

2.4. DeploymentErrorCallback

Callback for failed installations

```
[Callback=FunctionOnly, NoInterfaceObject] interface DeploymentErrorCallback {
    void onError (in DeploymentError error);
 };
```

Methods

`onError`

Failled installions.

Signature

`void onError(in DeploymentError error);`

Called if an installation was not accepted by the user or any other error occurred.

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DeploymentError
  - **Description:** The Widget API related error object of an unsuccessful application installation operation.

Return value

void

2.5. DeploymentError

Application installation specific errors.

```
    interface DeploymentError {
      const unsigned short INSTALLATION_CANCELED_BY_USER = 101;
      const unsigned short PERMISSION_DENIED_ERROR  = 102;
      const unsigned short NOT_REACHABLE  = 103;
      const unsigned short UNKNOWN_APPLICATION_ID  = 104;
      const unsigned short ALREADY_INSTALLED = 105;
      const unsigned short INSTALLATION_ERROR_OTHER = 106;
      readonly attribute unsigned short code;
      readonly attribute DOMString applicationID;
  };
```

The DeploymentError interface encapsulates all errors related to installation of applications. on the same or on other devices using the deploy function.

Constants

`unsigned short INSTALLATION_CANCELED_BY_USER`

Installation was cancelled by the user.

`unsigned short PERMISSION_DENIED_ERROR`

Not Authorized to use the service.

`unsigned short NOT_REACHABLE`

Device where the application should be installed on is not reachable. Consider retrying later.

`unsigned short UNKNOWN_APPLICATION_ID`

Device where the application should be installed on is not reachable. Consider retrying later.

`unsigned short ALREADY_INSTALLED`

Already Installed.

`unsigned short INSTALLATION_ERROR_OTHER`

Any other error.

Attributes

`readonly unsigned short code`

An error code assigned by an implementation when an error has occurred.

This attribute is readonly.

`readonly DOMString applicationID`

The application ID the error relates to.

This attribute is readonly.

2.6. Widget

Widget Interface

```
  interface Widget {

    readonly attribute DOMString     distributor;
    readonly attribute DOMString     distributorEmail;
    readonly attribute DOMString     distributorHref;
    readonly attribute DOMString     versionName;
    readonly attribute unsigned long long validfor;
    readonly attribute unsigned long long validuntil;
    readonly attribute DOMString     author;
    readonly attribute DOMString     authorEmail;
    readonly attribute DOMString     authorHref;
    readonly attribute DOMString     description;
    readonly attribute DOMString     id;
    readonly attribute DOMString     name;
    readonly attribute DOMString     shortName;
    readonly attribute Storage       preferences;
    readonly attribute DOMString     version;
    readonly attribute unsigned long height;
    readonly attribute unsigned long width;
    void exit();
    void hide();
    boolean isHidden();
    void notify(in NotifySuccessCallback onSuccess, in NotifyErrorCallback onError, in
DOMString title, in optional DOMString shortDescription, in optional DOMString id, in
optional DOMString icon);
    void cancelNotify(in DOMString id);
    void onDestroy();
    void onBackground();
    void onForeground();
    void onStop();
    void onStart();
    void deployChild(in DeploymentSuccessCallback onSuccess, in
DeploymentErrorCallback onError, in DOMString childApplicationID, in optional boolean
local);
  };
```

Defines that the webinos interface is part of the window global object.

Attributes

`readonly DOMString distributor`

An distributor attribute that represents people or an organization that distributed the widget.

This attribute is readonly.

`readonly DOMString distributorEmail`

A string attribute that represents an email address associated with the distributor.

This attribute is readonly.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`readonly DOMString distributorHref`

A string attribute that represents an email address associated with the distributor.

This attribute is readonly.

`readonly DOMString versionName`

A human readable version name.

This attribute is readonly.

`readonly unsigned long long validfor`

The validfor attributed defines a time interval until when the application is valid and can be used.

The time frame is specified in elapsed milliseconds after the first application execution.

This attribute is readonly.

`readonly unsigned long long validuntil`

The validuntil attributed defines a date and time until the application is valid and can be used.

The time frame is specified as in milliseconds whereas the date and time is encoded as milliseconds since midnight of January 1, 1970, according to universal time.

This attribute is readonly.

`readonly DOMString author`

An author attribute that represents people or an organization attributed with the creation of the widget.

This attribute is readonly.

`readonly DOMString authorEmail`

A string attribute that represents an email address associated with the author.

This attribute is readonly.

`readonly DOMString authorHref`

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.).

This attribute is readonly.

`readonly DOMString description`

The description element represents a human-readable description of the widget.

This attribute is readonly.

`readonly DOMString id`

An IRI attribute that denotes an identifier for the widget.

This attribute is readonly.

`readonly DOMString name`

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.

This attribute is readonly.

`readonly DOMString shortName`

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

This attribute is readonly.

`readonly Storage preferences`

The preference element allows authors to access preferences declared in the manifest file.

For a complete definition of the Storage attribute please read the W3C specification of the storage attribute in the Widget specification (http://www.w3.org/TR/2011/WD-widgets-apis-20110607/).

This attribute is readonly.

`readonly DOMString version`

A version attribute that specifies the version of the widget.

This attribute is readonly.

`readonly unsigned long height`

A numeric attribute greater than 0 that indicates the preferred viewport height of the instantiated custom start file in CSS pixels.

This attribute is readonly.

`readonly unsigned long width`

A numeric attribute greater than 0 that indicates the preferred viewport width of the instantiated custom start file in CSS pixels.

This attribute is readonly.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Methods

`exit`

Close the running application.

Signature

```
void exit();
```

Allows an application to trigger calling destroy from the runtime which results in stopping the application execution and closing the application.

Code example

```
//terminate the widget by its own
window.widget.exit();
```

`hide`

Hide the running application.

Signature

```
void hide();
```

Sends the application to background if possible so that it is not visible to the user anymore if possible by the platform the application execution goes on

Code example

```
//the widget is not visible anymore if possible
window.widget.hide();
```

`isHidden`

Checks visibility status.

Signature

```
boolean isHidden();
```

Asks the WRT wheather the application is currently hidden (not visible to the user) or not if the application is hidden and want to come to foreground it may notify an event to the user.

Code example

```
if (window.widget.isHidden()){
   //do things, e.g., create a notification
};
```

`notify`

Issues a notification to the user.

Signature

```
void notify(in NotifySuccessCallback onSuccess, in NotifyErrorCallback
onError, in DOMString title, in optional DOMString shortDescription, in
optional DOMString id, in optional DOMString icon);
```

Triggers the WRT to notify occurence of an event, as described using the parameters, to the user The user can click the event. If the application is in background the application must be brought to foreground. After that onSuccess is called.

Triggers the WRT to notify occurrence of an event, as described using the parameters, to the user The user can click the event or reject it. If the application is in background and the user accepted the event, e.g., by clicking on it, the application must be brought back to foreground. The notify success callback is then called after onForeground was called.

Parameters

- **onSuccess**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** NotifySuccessCallback
  - **Description:** NotifySuccessCallback issued when the user accepts the notification.
- **onError**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** NotifyErrorCallback
  - **Description:** ErrorCallback issued when the notification is discarded.
- **title**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** A short title describing the notification.

- **shortDescription**
    - **Optional:** Yes.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** A short description about the notification.
- **id**
    - **Optional:** Yes.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** An local identifier that represents the event and can be used to cancel the event or to take actions within the callbacks.
- **icon**
    - **Optional:** Yes.
    - **Nullable**: No
    - **Type:** DOMString
    - **Description:** A relative path within the application package to an icon describing the notification.

Code example

```
function error(id){
   if (id == "1"){
       //e.g, clear new e-mail list
  }
}

function success(id){
  if (id == "1"){
       //e.g, show new e-mails
  }
  else{
     if (id == "2"){
        //e.g, show new SMS messages
     }
  }
}

window.widget.notify(success, error, "New Emails", "You have 5 new E-Mails", 1);
```

`cancelNotify`

Cancels an ongoing notification.

Signature
```
void cancelNotify(in DOMString id);
```

To cancel a previous notify because it is updated or expired (if ongoing / not clicked by the user)

Parameters

- **id**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** The notification id to cancel.

Code example

```
//cancel notifications with id 1
window.widget.cancelNotify("1");
```

`onDestroy`

Asynchronous callback indicating that the application will be terminated.

Signature
```
void onDestroy();
```

Callback function which is called if the application will be shut down by the WRT. All application memory assigned to the application will be freed after returning out of this function.

`onBackground`

Asynchronous callback indicating that the application is gone to background.

Signature
```
void onBackground();
```

Callback function which is called after the application was put to background, e.g., another application goes to foreground and the application is not visible any more. After calling onBackground the application is still running but not visible anymore.

`onForeground`

Asynchronous callback indicating that the application is gone to foreground.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Signature
```
void onForeground();
```

Application goes to foreground after previously going to background.

`onStop`

Asynchronous callback indicating that application execution is going to be stopped.

Signature
```
void onStop();
```

Application execution is stopped aftern returning out of this function.

`onStart`

Asynchronous callback indicating that application execution is continued.

Signature
```
void onStart();
```

Application execution is continued after previously stopped.

`deployChild`

Requests to install an application on another device.

Signature
```
void deployChild(in DeploymentSuccessCallback onSuccess, in
DeploymentErrorCallback onError, in DOMString childApplicationID, in
optional boolean local);
```

Deploys a child application known to the WRT through the definition in the application s manifest file on another device. If local = false or not specified the WRT has to provide a list of available devices to the user where the application should be installed on, if local = true the WRT has to install the selected child on the same device as the API is bound to.

Parameters

- **onSuccess**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DeploymentSuccessCallback
  - **Description:** SuccessCallback called after successfull installation.
- **onError**
  - **Optional:** No.
  - **Nullable**: No

- **Type:** DeploymentErrorCallback
- **Description:** ErrorCallback called if installation was not possible.

- **childApplicationID**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** DOMString
  - **Description:** the application ID of the child package to be installed.
- **local**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** boolean
  - **Description:** Indicates if application should be installed on the same device as the requesting application or not.

Code example

```
function error(){
  //installation failed
}

function success(childID, serviceID){
  //application was successfully deployed
  //serviceID can be used for discovery to bind directly to this application
  //if functions are exposed by the application
}

//installing child application with name child1.wgt on the same device
window.widget.deployChild(success, error, "child1.wgt", true);
```

2.7. WindowWidget

The WindowWidget interface describes the part of the Widget API accessible through the window object.

```
 [Supplemental, NoInterfaceObject]interface WindowWidget {
    readonly attribute Widget widget;
  };
 Window implements WindowWidget;
```

3. Features

4. Full WebIDL

```
module widget {
    [NoInterfaceObject] interface NotifySuccessCallback {
     void onSuccess(in DOMString id);
   };

    [NoInterfaceObject] interface NotifyErrorCallback {
      void onError(in DOMString id);
   };

    [NoInterfaceObject] interface DeploymentSuccessCallback {
      void onSuccess(in DOMString childID, in DOMString serviceID);
    };

    [Callback=FunctionOnly, NoInterfaceObject] interface DeploymentErrorCallback {

       void onError (in DeploymentError error);
     };

     interface DeploymentError {
       const unsigned short INSTALLATION_CANCELED_BY_USER = 101;
       const unsigned short PERMISSION_DENIED_ERROR  = 102;
       const unsigned short NOT_REACHABLE  = 103;
       const unsigned short UNKNOWN_APPLICATION_ID  = 104;
       const unsigned short ALREADY_INSTALLED = 105;
       const unsigned short INSTALLATION_ERROR_OTHER = 106;
       readonly attribute unsigned short code;
       readonly attribute DOMString applicationID;
   };

   interface Widget {
     readonly attribute DOMString     distributor;
     readonly attribute DOMString     distributorEmail;
     readonly attribute DOMString     distributorHref;
     readonly attribute DOMString     versionName;
     readonly attribute unsigned long long validfor;
     readonly attribute unsigned long long validuntil;
     readonly attribute DOMString     author;
     readonly attribute DOMString     authorEmail;
     readonly attribute DOMString     authorHref;
     readonly attribute DOMString     description;
     readonly attribute DOMString     id;
     readonly attribute DOMString     name;
     readonly attribute DOMString     shortName;
     readonly attribute Storage       preferences;
     readonly attribute DOMString     version;
     readonly attribute unsigned long height;
     readonly attribute unsigned long width;
     void exit();
     void hide();
     boolean isHidden();
     void notify(in NotifySuccessCallback onSuccess, in NotifyErrorCallback onError, in
DOMString title, in optional DOMString shortDescription, in optional DOMString id, in
optional DOMString icon);
     void cancelNotify(in DOMString id);
     void onDestroy();
     void onBackground();
     void onForeground();
     void onStop();
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
    void onStart();
    void deployChild(in DeploymentSuccessCallback onSuccess, in DeploymentErrorCallback onError, in DOMString childApplicationID, in optional boolean local);
  };

  [Supplemental, NoInterfaceObject]interface WindowWidget {
     readonly attribute Widget widget;
   };

  Window implements WindowWidget;

};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

11. Referred APIs used by webinos

API Summary

Specification	Summary
The W3C calendar module	This W3C API provides access to a user calendaring service.
The W3C contacts module	This W3C API provides access to a user unified address book.
The WAC devicestatus module	This WAC API provides access to the information about the device status. The status information is organised as a tree structure utilising a vocabulary.
The devicestatus vocabulary module	The vocabulary that defines the information available in the webinos device status API.
The WAC deviceinteraction module	This WAC API allows applications the capability to access functions that allow them to interact with the end user.
The W3C DeviceOrientation Event specification	This specification defines several new DOM event types that provide information about the physical orientation and motion of a hosting device.
The W3C File API	This specification provides an API for representing file objects in web applications, as well as programmatically selecting them and accessing their data.
The W3C File API: Writer	This specification defines an API for writing to files from web applications. This API is designed to be used in conjunction with, and depends on definitions in, other APIs and elements on the web platform such as the W3C File API.
The W3C File API: Directories and System	This specification defines an API to navigate file system hierarchies, and defines a means by which a user agent may expose sandboxed sections of a user local filesystem to web applications. It builds on the File Writer API, which in turn built on the File API, each adding a different kind of functionality.
The W3C Gallery API	This specification defines an API that provides access to the media items stored in the device gallery.

Specification	**Summary**
The W3C Geolocation API	This specification defines an API that provides scripted access to geographical location information associated with the hosting device.
The W3C Media Capture API	This specification defines an API that provides access to the audio, image and video capture capabilities of the device.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The CalendarWrapper module

Webinos API Specifications

30 Jun 2011

Authors

- W3C Working Draft 19 April 2011
- Normative: W3C Calendar API
- WIDL version for webinos created by Christian Fuhrhop <christian.fuhrhop@fokus.fraunhofer.de>

Abstract

W3C based Calendar interface.

Summary of Methods

Interface	Method
ServiceCalendar	
Calendar	void findEvents(CalendarEventSuccessCB successCB, CalendarErrorCB errorCB, CalendarFindOptions options)
CalendarEvent	
CalendarRepeatRule	
CalendarFindOptions	
CalendarEventFilter	
CalendarError	

1. Introduction

This specification provides a wrapper that mandates the use of the W3C Calendar API (W3C Working draft 14th April).

The Calendar API defines a high-level interface to access Calendar information such as events, reminders, alarms and other calendar information.

The API itself is designed to be agnostic of any underlying calendaring service sources.

Note that while the W3C version, on which this specification is based on, provides only the ServiceCalendar to retrieve calendars, in webinos calendars can also be retrieved using the findServices method of the Service Discovery API.

2. Interfaces

2.1. ServiceCalendar

The ServiceCalendar interface is exposed on the Navigator interface [NAVIGATOR].

```
[NoInterfaceObject]
interface ServiceCalendar {
    readonly attribute Calendar calendar;
};
```

Attributes

`readonly Calendar calendar`

The root node from which the calendar functionality can be accessed.

No exceptions.

This attribute is readonly.

2.2. Calendar

The The Calendar interface provides a method to retrieve calendaring information from a user's calendar.

```
        [NoInterfaceObject]
        interface Calendar {
            caller void findEvents (in CalendarEventSuccessCB successCB, in optional
CalendarErrorCB errorCB, in optional CalendarFindOptions options);
        };
```

Methods

`findEvents`

Find calendar event items in the calendar based on an CalendarEventFilter object.

Signature

```
caller void findEvents(in CalendarEventSuccessCB successCB, in optional
CalendarErrorCB errorCB, in optional CalendarFindOptions options);
```

This method takes between one and three arguments. When called, it immediately returns, and then asynchronously start a find calendar event items process defined as follows:

If there are any tasks from the device task source in one of the task queues (i.e. an existing findEvents() operation is still pending a response), and the current method was invoked with a non-null errorCB argument, dispatch an error event with a PENDING_OPERATION_ERROR code value.

Search for calendar event items in the calendar according to the calendar item search processing rules.

If the attempt was successful, dispatch a success event. If the attempt fails, and the method was invoked with a non-null errorCB argument, this method must dispatch an error event with the code attribute set according to the type of failure that has occurred.

Parameters

- **successCB**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** CalendarEventSuccessCB
  - **Description:** Function to call when the asynchronous operation completes successfully.
- **errorCB**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** CalendarErrorCB
  - **Description:** Function to call when the asynchronous operation fails.
- **options**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** CalendarFindOptions
  - **Description:** The options to apply to the output of this method.

Return value

caller void

2.3. CalendarEvent

The CalendarEvent interface captures a calendar event object.

```
[NoInterfaceObject]
interface CalendarEvent {
     readonly attribute DOMString          id;
     attribute DOMString          description;
     attribute DOMString?         location;
     attribute DOMString?         summary;
     attribute DOMString          start;
     attribute DOMString?         end;
     attribute DOMString?         status;
     attribute DOMString?         transparency;
     attribute CalendarRepeatRule? recurrence;
     attribute DOMString?         reminder;
};
```

Attributes

`readonly DOMString id`

A globally unique identifier for the given CalendarEvent object. Each CalendarEvent referenced from Calendar must include a non-empty id value.

An implementation must maintain this globally unique resource identifier when a calendar event is added to, or present within, a Calendar.

An implementation may use an IANA registered identifier format. The value can also be a non-standard format.

No exceptions.

This attribute is readonly.

`DOMString description`

A description of the event.

No exceptions.

Code example

```
{description: "Meeting with Joe's team"}
```

`DOMString? location`

A plain text description of the location of the event.

No exceptions.

Code example

```
{location: 'Conf call #+4402000000001'}
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`DOMString? summary`

A summary of the event.

No exceptions.

Code example
```
 {summary: "Agenda: * Introductions* AoB"}
```

`DOMString start`

The start date and time of the event as a valid date or time string.

No exceptions.

Code example
```
 {start: '2011-03-24T09:00-08:00'} // Event starts on March 24, 2011 @
5pm (UTC)
```

`DOMString? end`

The end date and time of the event as a valid date or time string.

No exceptions.

Code example
```
 {end: '2011-03-24T10:00:00-08:00'} // Event ends on March 24, 2011 @
6pm (UTC)
```

`DOMString? status`

An indication of the user's status of the event.

This parameter may be set to one of the following constants:

No exceptions.

Code example
```
 {status: 'pending'} // Event is awaiting user action
```

`DOMString? transparency`

An indication of the display status to set for the event.

This parameter may be set to one of the following constants:

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

'transparent', 'opaque'.

No exceptions.

Code example
```
 {freebusy: 'transparent'} // Mark event as transparent in Calendar
```

`CalendarRepeatRule? recurrence`

The recurrence or repetition rule for this event

No exceptions.

Code example
```
 {recurrence: {frequency: 'daily'}}      // Event occurs every day and
never expires
```

Code example
```
 {recurrence: {frequency: 'weekly',     // Event occurs weekly...
  daysInWeek: [2, 4],      // ...every Tuesday and Thursday
  expires: '2011-06-11T12:00:00-04:00'}} // Event expires on or before June 11,
2011 @ 4pm (UTC)
```

Code example
```
 {recurrence: {frequency: 'weekly',        // Event occurs weekly...on every
Wednesday
// (if we say the 'start' attribute is March 24, 2011 @ 2pm (Wednesday) as
// shown above and no daysInWeek attribute is provided)
  expires: '2011-06-11T11:00:00-05:00'}} // Event expires on or before June 11,
2011 @ 4pm (UTC)
```

Code example
```
 {recurrence: {frequency: 'monthly',    // Event occurs monthly...
  daysInMonth: [-5],       // ...5 days before the end of each month
```

Code example
```
 {recurrence: {frequency: 'monthly',     // Event occurs monthly...on the 24th
day of every month
// (if we say the 'start' attribute is March 24, 2011 @ 2pm as
// shown above and no daysInMonth attribute is provided)
  expires: '2011-06-11T20:00:00+04:00'}} // Event expires on or before June 11,
2011 @ 4pm (UTC)
```

Code example

```
 {recurrence: {frequency: 'yearly',       // Event occurs yearly...on the 24th
day of every March
// (if we say the 'start' attribute is March 24, 2011 @ 2pm as
// shown above and no daysInMonth attribute is provided)
  expires: '2011-06-11T16:00:00+00:00'}} // Event expires on or before June 11,
2011 @ 4pm (UTC)
```

Code example

```
 {recurrence: {frequency: 'yearly',     // Event occurs yearly...
   daysInMonth: [24],       // ...every 24th day...
  monthsInYear: [3, 6],    // ...in every March and June
  expires: '2011-06-11T16:00:00Z'}} // Event expires on or before June 11, 2011
@ 4pm (UTC)
```

Code example

```
 {recurrence: {frequency: 'yearly',     // Event occurs yearly...
  daysInYear: [168],       // ...every 168th day of each year
  expires: '2011-06-11T21:45:00+05:45'}} // Event expires on or before June 11,
2011 @ 4pm (UTC)
```

`DOMString? reminder`

A reminder for the event.

This attribute can be specified as a positive valid date or time string, denoting a one-time reminder or as a negative value in milliseconds denoting a relative relationship to the start time of the calendar event.

A relative reminder is recommended for setting a reminder for recurrent events.

No exceptions.

Code example

```
 {reminder: '2011-03-24T13:00:00+00:00'}   // Remind ONCE on March 24,
2011 @ 1pm (UTC)
```

Code example

```
 {reminder: '-3600000'}        // Remind 1 hour before every occurrence
of this event
```

2.4. CalendarRepeatRule

The CalendarRepeatRule interface captures the recurrence of a calendar event item.

```
 [NoInterfaceObject]
 interface CalendarRepeatRule {
     attribute DOMString?      frequency;
     attribute unsigned short? interval;
     attribute DOMString?      expires;
     attribute DOMString[]     exceptionDates;
     attribute short[]         daysInWeek;
     attribute short[]         daysInMonth;
     attribute short[]         daysInYear;
     attribute short[]         weeksInMonth;
```

```
            attribute short[]         monthsInYear;
        };
```

Attributes

`DOMString? frequency`

The frequency of the CalendarRepeatRule.

This parameter must be set to one of the following constants: 'daily', 'weekly', 'monthly', 'yearly'.

Additional values must be ignored for this attribute.

No exceptions.

Code example

```
 {frequency: 'monthly'}  // Event repeats on a monthly basis
```

`unsigned short? interval`

A positive integer defining how often the recurrence rule must repeat.

For interval N, recurrence rule repeats every Nth frequency. Default interval value is 1, that is every day for a daily, every week for a weekly, every month for a monthly and every year for a yearly.

If this parameter is set to null the event item does not have any fixed interval and the event interal should be derived from the other CalendarRepeatRule attributes.

No exceptions.

Code example

```
 {interval: 1}
```

`DOMString? expires`

The date and time to which the CalendarRepeatRule applies as a valid date or time string.

If this parameter is set to null the event item does not have any fixed expiry date and the event is scheduled to continue indefintely.

No exceptions.

Code example

```
 {expires:  '2011-08-01T01:00:00+01:00'} //  Event  repeats  until  August
1, 2011 @ 12am (UTC)
```

`DOMString [] exceptionDates`

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

One or more dates and times to which the CalendarRepeatRule does not apply as valid date or time string strings.

If this parameter is set to null the event item does not have any exception dates and/or times.

No exceptions.

Code example

```
 {exceptionDates: ['2011-12-22', '2011-12-29']} // Event does not occur
on December 22, 2011 and December 29, 2011
```

`short [] daysInWeek`

The day or days of the week for which the CalendarRepeatRule applies. If this attribute is set to null then the day of the CalendarEvent.start value is used to derive the recurrent dates.

NOTE: This property only applies to weekly occurrences. If CalendarRepeatRule.frequency is not set to 'weekly' this property must be ignored.

The possible values are: 0 (Sunday)

1 (Monday)

2 (Tuesday)

3 (Wednesday)

4 (Thursday)

5 (Friday)

6 (Saturday)

No exceptions.

Code example

```
 {daysInWeek: [0, 6]} // A weekly event repeats every Sunday and
Saturday
```

`short [] daysInMonth`

The day or days of the month for which the CalendarRepeatRule applies. If this attribute is set to null then the day of the month of the CalendarEvent.start value is used to derive the recurrent dates.

NOTE: This property only applies to monthly occurrences. If CalendarRepeatRule.frequency is not set to 'monthly' this property must be ignored.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

The possible values are:

[1..31] (number of days from the first day of the month) [0..-30] (number of days before the last day of the month)

No exceptions.

Code example

```
 {daysInMonth: [4, -10]} // A monthly event repeats on the 4th and 10th
to last day of each month.
```

`short [] daysInYear`

The day or days of the month for which the CalendarRepeatRule applies. If this attribute is set to null then the day of the year of the CalendarEvent.start value is used to derive the recurrent dates.

NOTE: This property only applies to yearly occurrences. If CalendarRepeatRule.frequency is not set to 'yearly' this property must be ignored.

The possible values are:

[1..365] (number of days from the first day of the year) [0..-364] (number of days before the last day of the year)

No exceptions.

Code example

```
 {daysInYear: [262, -102]} // A yearly event repeats on day 262 and 102
days before the last day of each year.
```

`short [] weeksInMonth`

The week or weeks of the month for which the CalendarRepeatRule applies. If this attribute is set to null then the week of the month of the CalendarEvent.start value is used to derive the recurrent dates.

NOTE: This property only applies to monthly occurrences. If CalendarRepeatRule.frequency is not set to 'monthly' this property must be ignored. The possible values are:

[1..4] (number of weeks from the first week of the month) [0..-3] (number of weeks before the last week of the month)

No exceptions.

Code example

```
 {weeksInMonth: [1, -1]} // A monthly event repeats on the first and
last week of each month.
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`short [] monthsInYear`

The month or months of the year for which the CalendarRepeatRule applies. If this attribute is set to null then the month of the year of the CalendarEvent.start value is used to derive the recurrent dates.

NOTE: This property only applies to yearly occurrences. If CalendarRepeatRule.frequency is not set to 'yearly' this property must be ignored.

The possible values are:

1 (January)

2 (February)

3 (March)

4 (April)

5 (May)

6 (June)

7 (July)

8 (August)

9 (September)

10 (October)

11 (November)

12 (December)

No exceptions.

Code example

```
 {monthsInYear: [4, 10]} // A yearly event repeats in April and October
each year.
```

2.5. CalendarFindOptions

The CalendarFindOptions interface describes the options that can be applied to calendar searching.

```
[NoInterfaceObject]
interface CalendarFindOptions {
    attribute CalendarEventFilter? filter;
    attribute boolean?            multiple;
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Attributes

`CalendarEventFilter? filter`

A search filter with which to search and initially filter the Calendar database.

No exceptions.

`boolean? multiple`

A boolean value to indicate whether multiple Calendar objects are returnable as part of the associated Calendar findEvents() operation.

By default this option is set to true.

No exceptions.

2.6. CalendarEventFilter

The CalendarEventFilter interface captures the searchable parameters for finding calendar event items.

```
[NoInterfaceObject]
interface CalendarEventFilter : CalendarEvent {
    attribute DOMString startBefore;
    attribute DOMString startAfter;
    attribute DOMString endBefore;
    attribute DOMString endAfter;
};
```

Attributes

`DOMString startBefore`

Search for Calendar Events that start before the time provided as a valid date or time string..

No exceptions.

`DOMString startAfter`

Search for Calendar Events that start after the time provided as a valid date or time string..

No exceptions.

`DOMString endBefore`

Search for Calendar Events that end before the time provided as a valid date or time string..

No exceptions.

`DOMString endAfter`

Search for Calendar Events that end after the time provided as a valid date or time string..

No exceptions.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

2.9. CalendarError

he CalendarError interface encapsulates all errors in the manipulation of CalendarEvent objects in the Calendar API.

```
[NoInterfaceObject]
interface CalendarError {
    const unsigned short UNKNOWN_ERROR = 0;
    const unsigned short INVALID_ARGUMENT_ERROR = 1;
    const unsigned short TIMEOUT_ERROR = 2;
    const unsigned short PENDING_OPERATION_ERROR = 3;
    const unsigned short IO_ERROR = 4;
    const unsigned short NOT_SUPPORTED_ERROR = 5;
    const unsigned short PERMISSION_DENIED_ERROR = 20;
    readonly attribute unsigned short code;
};
```

Constants

`unsigned short UNKNOWN_ERROR`

An unknown error occurred.

`unsigned short INVALID_ARGUMENT_ERROR`

An invalid parameter was provided when the requested method was invoked.

`unsigned short TIMEOUT_ERROR`

The requested method timed out before it could be completed.

`unsigned short PENDING_OPERATION_ERROR`

If the user agent is currently waiting for a callback on a current findEvents() operation, as defined in this specification.

`unsigned short IO_ERROR`

An error occurred in communication with the underlying implementation that meant the requested method could not complete.

`unsigned short NOT_SUPPORTED_ERROR`

The requested method is not supported by the current implementation.

`unsigned short PERMISSION_DENIED_ERROR`

Access to the requested method was denied at the implementation or by the user.

Attributes

`readonly unsigned short code`

An error code assigned by an implementation when an error has occurred in Calendar API processing.

No exceptions.

This attribute is readonly.

3. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://www.w3.org/ns/api-perms/calendar.read

Read access to the calendar book.

4. Full WebIDL

```
module CalendarWrapper {

        [NoInterfaceObject]
        interface ServiceCalendar {
            readonly attribute Calendar calendar;
        };

        [NoInterfaceObject]
        interface Calendar {
            caller void findEvents (in CalendarEventSuccessCB successCB, in optional
CalendarErrorCB errorCB, in optional CalendarFindOptions options);
        };

        [NoInterfaceObject]
        interface CalendarEvent {
             readonly attribute DOMString          id;
             attribute DOMString          description;
             attribute DOMString?         location;
             attribute DOMString?         summary;
             attribute DOMString          start;
             attribute DOMString?         end;
             attribute DOMString?         status;
             attribute DOMString?         transparency;
             attribute CalendarRepeatRule? recurrence;
             attribute DOMString?         reminder;
        };

        [NoInterfaceObject]
        interface CalendarRepeatRule {
            attribute DOMString?      frequency;
            attribute unsigned short? interval;
            attribute DOMString?      expires;
            attribute DOMString[]     exceptionDates;
            attribute short[]         daysInWeek;
            attribute short[]         daysInMonth;
            attribute short[]         daysInYear;
            attribute short[]         weeksInMonth;
            attribute short[]         monthsInYear;
        };

        [NoInterfaceObject]
        interface CalendarFindOptions {
            attribute CalendarEventFilter? filter;
            attribute boolean?            multiple;
        };

        [NoInterfaceObject]
        interface CalendarEventFilter : CalendarEvent {
            attribute DOMString startBefore;
            attribute DOMString startAfter;
            attribute DOMString endBefore;
            attribute DOMString endAfter;
        };


        [Callback=FunctionOnly, NoInterfaceObject]
        interface CalendarEventSuccessCB {
            void onSuccess (in sequence<CalendarEvent> eventObjs);
        };

        [Callback=FunctionOnly, NoInterfaceObject]
        interface CalendarErrorCB {
            void onError (in CalendarError error);
        };

        [NoInterfaceObject]
        interface CalendarError {
            const unsigned short UNKNOWN_ERROR = 0;
            const unsigned short INVALID_ARGUMENT_ERROR = 1;
            const unsigned short TIMEOUT_ERROR = 2;
            const unsigned short PENDING_OPERATION_ERROR = 3;
            const unsigned short IO_ERROR = 4;
            const unsigned short NOT_SUPPORTED_ERROR = 5;
            const unsigned short PERMISSION_DENIED_ERROR = 20;
            readonly attribute unsigned short code;
        };
};
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The ContactsWrapper module

Webinos API Specifications

30 Jun 2011

Authors

- W3C Editor's Draft 16 June 2011
- Normative: W3C Contacts API
- WIDL version for webinos created by Christian Fuhrhop <christian.fuhrhop@fokus.fraunhofer.de>

Abstract

W3C based Contacts interface.

Summary of Methods

Interface	Method
ContactError	
ServiceContacts	
Contacts	void find(DOMString [] fields, ContactFindCB successCB, ContactErrorCB errorCB, ContactFindOptions options)
Contact	
ContactName	
ContactField	
ContactAddress	
ContactOrganization	

Interface	Method
ContactFindOptions	
ContactFindCB	void onsuccess(Contact [] contactObjs)
ContactErrorCB	void onerror(ContactError error)

1. Introduction

This specification provides a wrapper that mandates the use of the W3C Contacts API (Editor's draft 16th June).

The Contacts API defines the high-level interfaces required to obtain read access to a user's unified address book.

This API includes the following key interfaces:

A Contacts interface, which provides the method needed to access a user's unified address book. A Contact interface, which captures the individual contact information that can be returned following a successful read operation.

Note that while the W3C version, on which this specification is based on, provides only the ServiceContacts to retrieve contacts, in webinos contacts can also be retrieved using the findServices method of the Service Discovery API.

2. Interfaces

2.1. ContactError

Contacts specific errors.

```
 [NoInterfaceObject]
interface ContactError {
    const unsigned short UNKNOWN_ERROR = 0;
    const unsigned short INVALID_ARGUMENT_ERROR = 1;
    const unsigned short TIMEOUT_ERROR = 2;
    const unsigned short PENDING_OPERATION_ERROR = 3;
    const unsigned short IO_ERROR = 4;
    const unsigned short NOT_SUPPORTED_ERROR = 5;
    const unsigned short PERMISSION_DENIED_ERROR = 20;
    readonly attribute unsigned short code;
};
```

Constants

`unsigned short UNKNOWN_ERROR`

An unknown error occurred.

`unsigned short INVALID_ARGUMENT_ERROR`

An invalid parameter was provided when the requested method was invoked.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`unsigned short TIMEOUT_ERROR`

The requested method timed out before it could be completed.

`unsigned short PENDING_OPERATION_ERROR`

There is already a task in the device task source.

`unsigned short IO_ERROR`

An error occurred in communication with the underlying implementation that meant the requested method could not complete.

`unsigned short NOT_SUPPORTED_ERROR`

The requested method is not supported by the current implementation.

`unsigned short PERMISSION_DENIED_ERROR`

Access to the requested information was denied by the implementation or by the user.

Attributes

`readonly unsigned short code`

An error code assigned by an implementation when an error has occurred in Contacts API processing. No exceptions.

This attribute is readonly.

2.2. ServiceContacts

The ServiceContacts interface is exposed on the Navigator object [NAVIGATOR]. Its goal is to provide an access point to the functionality in this specification.

```
[NoInterfaceObject]
interface ServiceContacts {
            readonly attribute Contacts contacts;
};
```

Attributes

`readonly Contacts contacts`

The object through which the contacts functionality can be accessed. No exceptions.

This attribute is readonly.

2.3. Contacts

The Contacts interface exposes a database of contact information that may be retrieved.

```
        [NoInterfaceObject]
        interface Contacts {
            caller void find (DOMString[] fields, ContactFindCB successCB, optional
ContactErrorCB errorCB, optional ContactFindOptions options);
        };
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Multiple contact groups can be represented within this unified address book by specifying consistent categories values as part of individual Contact objects.

Multiple contact groups can be displayed by filtering on the required categories values via the Contacts find() operation.

The ServiceContacts interface is exposed on the Navigator object [NAVIGATOR].

Its goal is to provide an access point to the functionality in this specification.

Methods

`find`

Find contacts in the address book according to the find contacts process detailed below.

Signature

```
caller void find(

                DOMString

              [] fields, ContactFindCB successCB, optional ContactErrorCB
errorCB, optional ContactFindOptions options);
```

This method takes two, three or four arguments. When called, it starts the following find contacts process:

Let successCallback be the callback indicated by the method's second argument.

Let errorCallback be the callback indicated by the method's third argument, if any, or null otherwise.

If successCallback is null, then throw a TypeError (as defined in [WEBIDL]).

If there is a task from the device task source in one of the task queues (e.g. an existing find() operation is still pending a response), run these substeps:

If errorCallback is not null, let error be a ContactError object whose code attribute has the value PENDING_OPERATION_ERROR and queue a task to invoke errorCallback with error as its argument.

Abort this operation.

Return, and run the remaining steps asynchronously.

Let results be the array of Contact objects obtained by searching contacts in the address book according to the rules defined in Contact Search Processing, or null if the search has failed.

If results is null, run these substeps:

If errorCallback is not null, let error be a ContactError object whose code attribute has its value set according to the type of failure that occurred and queue a task to invoke errorCallback with error as its argument.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Abort this operation.

Queue a task to invoke successCallback with results as its argument.

Parameters

- **fields**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** array
  - **Description:** The search qualifier.
- **successCB**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ContactFindCB
  - **Description:** Function to call when the asynchronous operation completes successfully.
- **errorCB**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** ContactErrorCB
  - **Description:** Function to call when the asynchronous operation fails.
- **options**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** ContactFindOptions
  - **Description:** The options to apply to the output of this method.

Return value

caller void

2.4. Contact

The Contact interface captures the properties of a contact object.

```
[NoInterfaceObject]
interface Contact {
     readonly attribute DOMString             id;
```

```
            attribute DOMString?             displayName;
            attribute  ContactName           name;
            attribute DOMString?             nickname;
            attribute ContactField[]?        phoneNumbers;
            attribute ContactField[]?        emails;
            attribute ContactAddress[]?      addresses;
            attribute ContactField[]?        ims;
            attribute ContactOrganization[]? organizations;
            attribute Date?                  revision;
            attribute Date?                  birthday;
            attribute DOMString?             gender;
            attribute DOMString?             note;
            attribute ContactField[]?        photos;
            attribute DOMString[]?           categories;
            attribute ContactField[]?        urls;
            attribute DOMString?             timezone;
        };
```

All Contact objects must include all attributes supported by the implementation, regardless of whether these attributes have been assigned a null value or not. If a supported attribute has not been assigned a value by the user or the implementation, then this attribute must still be present in the resulting Contact object and must have a value of null.

Additional attributes may be included according to the provisions detailed in Extended Contact Properties and Parameters. If an extended attribute is supported by the current implementation and has not been assigned a value by the user or the implementation, then this extended attribute must still be present in the resulting Contact object and must have a value of null.

Attributes

`readonly DOMString id`

A globally unique identifier for the given Contact object.

Each Contact instance must include a non-empty id value.

No exceptions.

This attribute is readonly.

`DOMString? displayName`

This attribute contains the name of this Contact in a form that is suitable for display to the user.

Each Contact must include either a displayName or the name attribute.

No exceptions.

`ContactName name`

This attribute represents the full name of this Contact indicated by the name components associated with the ContactName object.

Each Contact must include either a displayName or the name attribute.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

No exceptions.

`DOMString? nickname`

This attribute contains the nickname (or a casual name) for this Contact.

No exceptions.

`ContactField [] phoneNumbers`

This attribute captures one or more phone numbers associated with this Contact.

No exceptions.

`ContactField [] emails`

This attribute represents one or more email addresses associated with this Contact.

No exceptions.

`ContactAddress [] addresses`

This attribute represents one or more physical addresses associated with this Contact.

No exceptions.

`ContactField [] ims`

This attribute represents one or more instant messaging identifiers associated with this Contact.

No exceptions.

`ContactOrganization [] organizations`

This attribute represents one or more organizations associated with this Contact.

No exceptions.

`Date? revision`

This attribute contains the timestamp information associated with this Contact, which represents the last known modification time. If no modification time exists, then this object contains the timestamp of the object's creation time.

No exceptions.

`Date? birthday`

This attribute contains birthday of this Contact.

The year value may be set to 0000 when the age of the Contact is private or the year is not available.

No exceptions.

`DOMString? gender`

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

This attribute contains the gender of this Contact. This attribute should have one of the following values:

male

female

undisclosed

Note however that this attribute may contain a value not listed above.

No exceptions.

`DOMString? note`

This attribute contains the personal notes (free-text) for this Contact that is managed by the user of the address book.

No exceptions.

`ContactField [] photos`

This attribute represents one or more photos associated with this Contact.

The photos must be specified in the value attribute of the ContactField object either by using a URL to an image resource or base64 encoded string of the image data.

No exceptions.

`DOMString [] categories`

This attribute contains one or more user-defined categories/tags/labels associated with this Contact. e.g. "family", "favourite", "cryptozoologists".

No exceptions.

`ContactField [] urls`

This attribute represents one or more URLs associated with this Contact e.g. personal web page, blog.

The web resources must be specified using the value attribute of the ContactField object, and its type field may be set to "blog" or "profile".

No exceptions.

`DOMString? timezone`

This attribute represents the time zone of this Contact.

It is recommended that names from the public-domain Olson database [TZDB] will be used as the value of this attribute, but this is not a restriction. For example, a value of America/New_York indicates the Contact is associated with the variable time zone of the New York region of the United States, including daylight saving time offsets experienced in that region.

It is also possible to use this attribute to express the timezone as a positive or negative difference from UTC, in the 24-hour clock, in units of hours and minutes (i.e. +hh:mm). For example, a value of +05:30 indicates the Contact is associated with a fixed time zone of GMT+05:30.

No exceptions.

2.5. ContactName

The ContactName interface describes a contact's name.

```
[NoInterfaceObject]
  interface ContactName {
      attribute DOMString? formatted;
      attribute DOMString? familyName;
      attribute DOMString? givenName;
      attribute DOMString? middleName;
      attribute DOMString? honorificPrefix;
      attribute DOMString? honorificSuffix;
  };
```

Attributes

`DOMString? formatted`

This attribute contains the full name, including all the individual components such as givenName, middleName, familyName, prefix, suffix as appropriate for the user's culture, and formatted for display (e.g. Mr. Joe Smith Jr).

No exceptions.

`DOMString? familyName`

This attribute contains the family name (also referred to as the last name) of this Contact.

No exceptions.

`DOMString? givenName`

This attribute contains the given name (also referred to as the first name) of this Contact.

No exceptions.

`DOMString? middleName`

This attribute contains the middle name of this Contact.

No exceptions.

`DOMString? honorificPrefix`

This attribute contains the honorific prefix (or title) of this Contact. E.g. Mr., Dr., Ms., Mrs.

No exceptions.

`DOMString? honorificSuffix`

This attribute contains the honorific suffix of this Contact. E.g. Jr, III, Sr.

No exceptions.

2.6. ContactField

The ContactField interface is a reusable component that is used to capture contact fields of the Contact interface that have some modicum of structure.

```
[NoInterfaceObject]
interface ContactField {
    attribute DOMString  type;
    attribute DOMString? value;
    attribute boolean    pref;
};
```

Attributes

`DOMString type`

This attribute contains the type information for this ContactField and its content varies subject to the contact property this ContactField is representing. For example, if the ContactField is representing a phoneNumber property, the type attribute can be set to home, mobile; if the ContactField is representing the ims property, the type attribute could be set to xmpp, irc, bbm, etc.

No exceptions.

`DOMString? value`

This attribute contains the value for this ContactField and its content varies subject to the contact property this ContactField is representing. For example, if the ContactField is representing an email, the value attribute could be set to JoeSmith@example.com, and if the ContactField is representing a url, the value attribute can be set to http://www.example.org/joesmith, etc.

No exceptions.

`boolean pref`

This attribute indicates whether this instance of the ContactField is the preferred, or primary, value for the contact property this ContactField is representing in the Contact interface. By default, the value is false.

No exceptions.

2.7. ContactAddress

The ContactAddress interface is a reusable component that is used to capture addresses within the Contact interface.

```
[NoInterfaceObject]
interface ContactAddress {
    attribute boolean    pref;
    attribute DOMString? type;
    attribute DOMString? formatted;
    attribute DOMString? streetAddress;
```

```
        attribute DOMString? locality;
        attribute DOMString? region;
        attribute DOMString? postalCode;
        attribute DOMString? country;
    };
```

Attributes

`boolean pref`

This attribute indicates whether this instance of the ContactAddress is the preferred, or primary, value for the contact. By default, the value is false.

No exceptions.

`DOMString? type`

This attribute contains the type of address this object is representing (e.g. work, home, premises, etc).

No exceptions.

`DOMString? formatted`

This attribute contains the full physical address including street, locality, region, postalCode, and country as appropriate, and formatted for display.

No exceptions.

`DOMString? streetAddress`

This attribute contains the street address corresponding to this ContactAddress.

No exceptions.

`DOMString? locality`

This attribute contains the locality (or city) name corresponding to this ContactAddress.

No exceptions.

`DOMString? region`

This attribute contains the region (or state/province) name corresponding to this ContactAddress.

No exceptions.

`DOMString? postalCode`

This attribute contains the postal code (or zip) corresponding to this ContactAddress.

No exceptions.

`DOMString? country`

This attribute contains the country name corresponding to this ContactAddress.

No exceptions.

2.8. ContactOrganization

The ContactOrganization interface is a reusable component that is used to support contact organisations within the Contact interface.

```
[NoInterfaceObject]
   interface ContactOrganization {
       attribute boolean    pref;
       attribute DOMString? type;
       attribute DOMString? name;
       attribute DOMString? department;
       attribute DOMString? title;
   };
```

Attributes

`boolean pref`

This attribute indicates whether this instance of the ContactOrganization is the preferred, or primary, value for the contact. By default, the value is false.

No exceptions.

`DOMString? type`

This attribute contains the type of organization this object is representing.

No exceptions.

`DOMString? name`

The name of the organisation.

No exceptions.

`DOMString? department`

The department within which this Contact works.

No exceptions.

`DOMString? title`

The job title that the Contact holds inside this organisation.

No exceptions.

2.9. ContactFindOptions

The ContactFindOptions interface describes the options that can be applied to contact searching. When a ContactFindOptions parameter is provided to the Contacts find() operation, it should be processed according to the provisions detailed in Options Processing.

```
[NoInterfaceObject]
interface ContactFindOptions {
    attribute DOMString? filter;
```

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

```
        attribute boolean?   multiple;
        attribute Date      updatedSince;
    };
```

Attributes

`DOMString? filter`

A string-based search filter which provides a hint to the user agent to facilitate contacts selection by the user.

No exceptions.

`boolean? multiple`

A boolean value to indicate whether multiple Contact objects are wanted as part of the Contacts find() operation. By default this option is set to false.

No exceptions.

`Date updatedSince`

Return only contact records that have been updated on or after the given time, specified as an ECMAScript Date object.

This filter is applied to the revision field as defined in Contact.

No exceptions.

2.10. ContactFindCB

This is the wrapper interface for callbacks indicating success of the find() operation.

```
    [Callback=FunctionOnly, NoInterfaceObject]
    interface ContactFindCB {
        void onsuccess (Contact[] contactObjs);
    };
```

Methods

`onsuccess`

Callback on success of a find() operation

Signature

```
void onsuccess(

                Contact

              [] contactObjs);
```

Parameters

- **contactObjs**
  - **Optional:** No.
  - **Nullable**: No

- **Type:** array
- **Description:** An array of Contact objects resulting from the given Contacts find() operation.

Return value

void

2.11. ContactErrorCB

This is the wrapper interface for callbacks indicating failure of the find() operation.

```
[Callback=FunctionOnly, NoInterfaceObject]
interface ContactErrorCB  {
      void onerror (ContactError error);
};
```

Methods

onerror

Callback on failure of a find() operation

Signature

```
void onerror(ContactError error);
```

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** ContactError
  - **Description:** The ContactError object capturing the type of the error.

Return value

void

3. Features

This is the list of URIs used to declare this API's features, for use in the widget config.xml and as identifier for service type in service discovery functionality. For each URI, the list of functions covered is provided.

http://www.w3.org/ns/api-perms/contacts.read

Read access to the address book.

4. Full WebIDL

```
module ContactsWrapper {
```

```
         [NoInterfaceObject]
        interface ContactError {
            const unsigned short UNKNOWN_ERROR = 0;
            const unsigned short INVALID_ARGUMENT_ERROR = 1;
            const unsigned short TIMEOUT_ERROR = 2;
            const unsigned short PENDING_OPERATION_ERROR = 3;
            const unsigned short IO_ERROR = 4;
            const unsigned short NOT_SUPPORTED_ERROR = 5;
            const unsigned short PERMISSION_DENIED_ERROR = 20;
            readonly attribute unsigned short code;
        };

        [NoInterfaceObject]
        interface ServiceContacts {
                    readonly attribute Contacts contacts;
        };

        [NoInterfaceObject]
        interface Contacts {
            caller void find (DOMString[] fields, ContactFindCB successCB, optional
ContactErrorCB errorCB, optional ContactFindOptions options);
        };

        [NoInterfaceObject]
        interface Contact {
             readonly attribute DOMString              id;
             attribute DOMString?             displayName;
             attribute  ContactName           name;
             attribute DOMString?             nickname;
             attribute ContactField[]?        phoneNumbers;
             attribute ContactField[]?        emails;
             attribute ContactAddress[]?      addresses;
             attribute ContactField[]?        ims;
             attribute ContactOrganization[]? organizations;
             attribute Date?                  revision;
             attribute Date?                  birthday;
             attribute DOMString?             gender;
             attribute DOMString?             note;
             attribute ContactField[]?        photos;
             attribute DOMString[]?           categories;
             attribute ContactField[]?        urls;
             attribute DOMString?             timezone;
        };

      [NoInterfaceObject]
        interface ContactName {
            attribute DOMString? formatted;
            attribute DOMString? familyName;
            attribute DOMString? givenName;
            attribute DOMString? middleName;
            attribute DOMString? honorificPrefix;
            attribute DOMString? honorificSuffix;
        };

        [NoInterfaceObject]
        interface ContactField {
            attribute DOMString  type;
            attribute DOMString? value;
            attribute boolean    pref;
        };

        [NoInterfaceObject]
        interface ContactAddress {
            attribute boolean    pref;
            attribute DOMString? type;
            attribute DOMString? formatted;
            attribute DOMString? streetAddress;
            attribute DOMString? locality;
            attribute DOMString? region;
            attribute DOMString? postalCode;
            attribute DOMString? country;
        };

     [NoInterfaceObject]
        interface ContactOrganization {
            attribute boolean    pref;
            attribute DOMString? type;
            attribute DOMString? name;
            attribute DOMString? department;
            attribute DOMString? title;
        };

        [NoInterfaceObject]
        interface ContactFindOptions {
            attribute DOMString? filter;
            attribute boolean?   multiple;
            attribute Date      updatedSince;
        };

        [Callback=FunctionOnly, NoInterfaceObject]
        interface ContactFindCB {
            void onsuccess (Contact[] contactObjs);
        };

        [Callback=FunctionOnly, NoInterfaceObject]
        interface ContactErrorCB  {
              void onerror (ContactError error);
        };
};
```

APIs: The device status module

Webinos Specification

June 2011

Abstract

This document describes the functionality that Webinos devices should implement for retrieving device status information.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The access to Device Status information is done through the WAC Device Status API.

2 - API

This section is NORMATIVE.

Webinos implementations MUST support WAC Device Status specification. The implementations MUST also support the vocabulary defined by webinos for accessing the device status information.

In order to use this API, access to it must be declared in the widget configuration document (i.e. config.xml). This declaration is done through the features http://wacapps.net/api/devicestatus, http://wacapps.net/api/devicestatus.deviceinfo or http://wacapps.net/api/devicestatus.networkinfo.

3 - Security

This section is NORMATIVE.

The implementation MUST NOT enable access to this API by default, but only if the declaration is present in the widget configuration document through the appropriate feature tag.

Please note that Webinos Security Framework, depending on its configuration and in the widget level of trust, MAY deny access to this API even if it is declared in the configuration document.

4 - WebIDL

This section is INFORMATIVE.

For completeness, this specification includes a copy of the WebIDL declaration included in the WAC Device Status specification. The referred specification includes all the details needed to create a conformant implementation as well as a reference to the WAC Vocabulary. Please note that webinos does not use WAC Vocabulary but a webinos specific one.

DeviceapisDeviceStatusManager Interface

```
[NoInterfaceObject] interface DeviceapisDeviceStatusManager {
    readonly attribute DeviceStatusManager devicestatus;
};
webinos implements DeviceapisDeviceStatusManager;
```

DeviceStatusManager Interface

```
[NoInterfaceObject] interface DeviceStatusManager {

    StringArray getComponents(in DOMString aspect)
        raises(DeviceAPIError);

    boolean isSupported(in DOMString aspect,
                        [TreatUndefinedAs=Null] in optional DOMString? property)
                        raises(DeviceAPIError);

    PendingOperation getPropertyValue(in PropertyValueSuccessCallback successCallback,
                                      in ErrorCallback errorCallback,
                                      in PropertyRef prop)
                                      raises(DeviceAPIError);

    long watchPropertyChange(in PropertyValueSuccessCallback successCallback,
                             in ErrorCallback errorCallback,
                             in PropertyRef prop,
                             in optional WatchOptions options)
                             raises(DeviceAPIError);

    void clearPropertyChange(in unsigned long watchHandler)
                             raises(DeviceAPIError);
  };
```

PropertyRef Interface

```
[Callback, NoInterfaceObject] interface PropertyRef {
    attribute DOMString component;
    attribute DOMString aspect;
    attribute DOMString property;
  };
```

WatchOptions Interface

```
[Callback, NoInterfaceObject] interface WatchOptions {
    attribute long minNotificationInterval;
    attribute long maxNotificationInterval;
    attribute long minChangePercent;
  };
```

DeviceAPIError Interface

```
  [NoInterfaceObject] interface DeviceAPIError {
    readonly attribute unsigned short code;
    readonly attribute DOMString message;
    const unsigned short      UNKNOWN_ERR                    = 0;
    const unsigned short      INDEX_SIZE_ERR                 = 1;
    const unsigned short      DOMSTRING_SIZE_ERR             = 2;
    const unsigned short      HIERARCHY_REQUEST_ERR          = 3;
    const unsigned short      WRONG_DOCUMENT_ERR             = 4;
    const unsigned short      INVALID_CHARACTER_ERR          = 5;
    const unsigned short      NO_DATA_ALLOWED_ERR            = 6;
    const unsigned short      NO_MODIFICATION_ALLOWED_ERR    = 7;
    const unsigned short      NOT_FOUND_ERR                  = 8;
    const unsigned short      NOT_SUPPORTED_ERR              = 9;
    const unsigned short      INUSE_ATTRIBUTE_ERR            = 10;
    const unsigned short      INVALID_STATE_ERR              = 11;
    const unsigned short      SYNTAX_ERR                     = 12;
    const unsigned short      INVALID_MODIFICATION_ERR       = 13;
    const unsigned short      NAMESPACE_ERR                  = 14;
    const unsigned short      INVALID_ACCESS_ERR             = 15;
    const unsigned short      VALIDATION_ERR                 = 16;
    const unsigned short      TYPE_MISMATCH_ERR              = 17;
    const unsigned short      SECURITY_ERR                   = 18;
    const unsigned short      NETWORK_ERR                    = 19;
    const unsigned short      ABORT_ERR                      = 20;
    const unsigned short      TIMEOUT_ERR                    = 21;
    const unsigned short      INVALID_VALUES_ERR             = 22;
    const unsigned short      NOT_AVAILABLE_ERR              = 101;
  };
```

PropertyValueSucessCallback Interface

```
 [Callback=FunctionOnly, NoInterfaceObject] interface GetPropertySuccessCallback {
    void onpropertyvalue(in Object value, in PropertyRef property);
  };
```

ErrorCallback Interface

```
[Callback=FunctionOnly, NoInterfaceObject] interface ErrorCallback {
   void onerror(in DeviceAPIError error);
};
```

PendingOperation Interface

```
[NoInterfaceObject] interface PendingOperation {
   boolean cancel();
 };
```

References

[DEVICESTATUS]

NORMATIVE: WAC Device Status (Approved Release Version - June 2011) , see http://specs.wacapps.net/2.0/jun2011/deviceapis/devicestatus.html

[DEVICESTATUSVOC]

INFORMATIVE: WAC Device Status Vocabulary (Approved Release Version - June 2011) , see http://specs.wacapps.net/2.0/jun2011/deviceapis/vocabulary.html

[WEBINOSVOC]

NORMATIVE: Webinos Device Status Vocabulary

APIs: Device status vocabulary

Webinos Specification

June 2011

Abstract

This document describes the Webinos Vocabulary for being used by the Device Status API.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The vocabulary chosen is an extension of the WAC Vocabulary.

2 - Aspects

Aspect	**Properties**	**Supported Component Aliases**	**Description**
Battery	batteryLevel, batteryBeingCharged	_default	Describes one battery in a device. See also Delivery Context Ontology: hard:battery
Camera (**)	model (**), vendor (**), status (**), resolutionHeight (**), resolutionWidth (**), maxZoom (**), minZoom (**), currentZoom (**), hasFlash (**), flashOn (**)	_active, _default	It represents a camera of the device. See also Delivery Context Ontology: hard:Camera
CellularHardware	status	_default	It represents a device hardware that can be used to access to mobile operator telephony networks. See also Delivery Context Ontology:

Aspect	**Properties**	**Supported Component Aliases**	**Description**
			hard:CellularHardware
CellularNetwork	isInRoaming, mcc, mnc, signalStrength, operatorName, ipAddress (*), macAddress (*)	_default	It represents a Cellular Network. See also Delivery Context Ontology: net:Network
CPU (**)	model (**), currentLoad (*)	_default	It represents the CPU of the device. See also Delivery Context Ontology: hard:CPU
Device	imei, model, version, vendor	_default	It represents the device. See also Delivery Context Ontology: dcn:Device
Display	resolutionHeight, pixelAspectRatio, dpiY, resolutionWidth, dpiX, colorDepth	_active, _default	It represents a visual display on the device. See also Delivery Context Ontology: hard:Display
InputDevice (*)	type (*)	_default	It represents an input device. See also Delivery Context Ontology: hard:InputDevice
MemoryUnit	size, removable, availableSize, volatile (**)	_default	It represents a memory unit used in the device. See also Delivery Context Ontology: hard:MemoryUnit
OperatingSystem	language, version, name, vendor	_active, _default	It represents the device operating system. See also Delivery Context Ontology: soft:OperatingSystem
ParentalRating (*)	name (*), scheme (*), region (*)	_default	The parental rating which was set for this device. Properties description aligns with the OIPF R2 Vol5 DAE Spec.
WebRuntime	wacVersion, supportedImageFormats, version, name, vendor, webinosVersion (*)	_active, _default	It represents a Web runtime capable of executing widgets. See also Delivery Context Ontology:

Aspect	Properties	Supported Component Aliases	Description
			web:WebRuntime
WiFiHardware	status	_default	It represents hardware in a device that can be used to access to WiFi networks. See also Delivery Context Ontology: hard:WiFiHardware
WiFiNetwork	ssid, signalStrength, networkStatus, ipAddress (*), macAddress (*)	_active, _default	It represents a WiFi Network. See also Delivery Context Ontology: net:WiFiNetwork
WiredNetwork (*)	networkStatus (*), ipAddress (*), macAddress (*)	_active, _default	It represents a Wired Network. See also Delivery Context Ontology: net:WiredNetwork

(*) This aspect/property has been added by Webinos.
(**) This aspect/property has been taken from the BONDI vocabulary.

3 - Properties

For informations about all other properties check the WAC Vocabulary.

Model

ID

model

Associated Aspect

Camera

Description

This property indicates the model of the Camera. See also Delivery Context Ontology.

Delivery Context Ontology Associated Entity

common:model

WebIDL Type

DOMString

Vendor

ID

vendor

Associated Aspect

Camera

Description

This property indicates the vendor of the camera. See also Delivery Context Ontology.

Delivery Context Ontology Associated Entity

common:vendor

WebIDL Type

DOMString

Status

ID

status

Associated Aspect

Camera

Description

This property indicates the status of the camera. See also Delivery Context Ontology.

Delivery Context Ontology Associated Entity

hard:status

WebIDL Type

DOMString with a value chosen from the following:

(inserire tabella)

Resolution Height

ID

resolutionHeight

Associated Aspect

Camera

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Description

This property indicates the height resolution of the camera. See also Delivery Context Ontology.

Delivery Context Ontology Associated Entity

common:resolutionHeight

WebIDL Type

unsigned short

Resolution Width

ID

resolutionWidth

Associated Aspect

Camera

Description

This property indicates the width resolution of the camera. See also Delivery Context Ontology.

Delivery Context Ontology Associated Entity

common:resolutionWidth

WebIDL Type

unsigned short

Max Zoom

ID

maxZoom

Associated Aspect

Camera

Description

This property indicates the maximum zoom of the camera.

WebIDL Type

unsigned short

Min Zoom

ID

minZoom

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Associated Aspect

Camera

Description

This property indicates the minimum zoom of the camera.

WebIDL Type

unsigned short

Current Zoom

ID

currentZoom

Associated Aspect

Camera

Description

This property indicates the current zoom of the camera.

WebIDL Type

unsigned short

Has Flash

ID

hasFlash

Associated Aspect

Camera

Description

This property indicates if the camera has the flash.

WebIDL Type

boolean

Flash On

ID

flashOn

Associated Aspect

Camera

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Description

This property indicates if the flash of the camera is active.

WebIDL Type

boolean

Model

ID

model

Associated Aspect

CPU

Description

This property indicates the model of the CPU. See also Delivery Context Ontology.

Delivery Context Ontology Associated Entity

common:model

WebIDL Type

DOMString

Current load

ID

currentLoad

Associated Aspect

CPU

Description

This property indicates the current load of the CPU as a percentage. In case of multi processor CPU, it reports the average.

WebIDL Type

unsigned short

Type

ID

type

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Associated Aspect

inputDevice

Description

This property indicates the type of input devices. See also Delivery Context Ontology.

WebIDL Type

DOMString with a value chosen from the following:

Value	Description
clickWheel	See context delivery ontology hard:InputDevice_CLICK_WHEEL
fourWayScroller	See context delivery ontology hard:InputDevice_FOUR_WAY_SCROLLER
jogDial	See context delivery ontology hard:InputDevice_JOG_DIAL
mouse	See context delivery ontology hard:InputDevice_MOUSE
numericKeypad	See context delivery ontology hard:InputDevice_NUMERIC_KEYPAD
phoneKeypad	See context delivery ontology hard:InputDevice_PHONE_KEYPAD
qwertyKeyboard	See context delivery ontology hard:InputDevice_QWERTY_KEYBOARD
stylus	See context delivery ontology hard:InputDevice_STYLUS
touchScreen	See context delivery ontology hard:InputDevice_TOUCH_SCREEN
trackBall	See context delivery ontology hard:InputDevice_TRACK_BALL

Volatile

ID

volatile

Associated Aspect

MemoryUnit

Description

This property indicates if a memory unit is volatile or not. See also Delivery Context Ontology.

Delivery Context Ontology Associated Entity

hard:volatile

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

WebIDL Type

boolean

Name

ID

name

Associated Aspect

ParentalRating

Description

Rating value as denoted by scheme, eg. "PG-13".

WebIDL Type

DOMString

Scheme

ID

scheme

Associated Aspect

ParentalRating

Description

Guidance scheme URIs as defined in MPEG-7.

WebIDL Type

DOMString

Region

ID

region

Associated Aspect

ParentalRating

Description

Indicates region.

WebIDL Type

DOMString

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

Webinos Version

ID

webinosVersion

Associated Aspect

WebRuntime

Description

Indicates the version of the webinos specs supported.

WebIDL Type

DOMString

Network Status

ID

networkStatus

Associated Aspect

WiredNetwork

Description

Indicates the status of a wired network.

WebIDL Type

DOMString with a value chosen from the following:

Value	**Description**
connected	connected
unconnected	unconnected

IP Address

ID

ipAddress

Associated Aspects

CellularNetwork

WiFiNetwork

WiredNetwork

Description

IP address of the connection (null if not connected).

WebIDL Type

DOMString

Mac Address

ID

macAddress

Associated Aspects

CellularNetwork

WiFiNetwork

WiredNetwork

Description

Mac address of the connection (null if not available).

WebIDL Type

DOMString

References

[DEVICESTATUSVOC]

NORMATIVE: WAC Device Status Vocabulary (Approved Release Version - June 2011) , see http://specs.wacapps.net/2.0/jun2011/deviceapis/vocabulary.html

[BONDIDEVICESTATUSVOC]

INFORMATIVE: BONDI Device Status Vocabulary (Release 1.1) , see http://bondi.omtp.org/1.1/apis/vocabulary.htm

APIs: The device interaction module

Webinos Specification

June 2011

Abstract

This document describes the functionality that Webinos devices should implement for interacting with the end user through the device.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The interaction with the end-user is done through the WAC Device Interaction API.

2 - API

This section is NORMATIVE.

Webinos implementations MUST support WAC Device Interaction specification.

In order to use this API, access to it must be declared in the widget configuration document (i.e. config.xml). This declaration is done through the feature http://wacapps.net/api/deviceinteraction.

3 - Security

This section is NORMATIVE.

The implementation MUST NOT enable access to this API by default, but only if the declaration is present in the widget configuration document through the appropriate feature tag.

Please note that Webinos Security Framework, depending on its configuration and in the widget level of trust, MAY deny access to this API even if it is declared in the configuration document.

4 - WebIDL

This section is INFORMATIVE.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

For completeness, this specification includes a copy of the WebIDL declaration included in the WAC Device Interaction specification. The referred specification includes all the details needed to create a conformant implementation.

DeviceapisDeviceInteractionManager Interface

```
interface DeviceapisDeviceInteractionManager {
    readonly attribute DeviceInteractionManager deviceinteraction;
  };
  Deviceapis implements DeviceapisDeviceInteractionManager;
```

DeviceInteractionManager Interface

```
  interface DeviceInteractionManager {

    PendingOperation  startNotify(in  SuccessCallback  successCallback,in  ErrorCallback
errorCallback,in long duration)
                     raises (DeviceAPIError);

    void stopNotify();

    PendingOperation startVibrate(in SuccessCallback successCallback,
                                  in ErrorCallback errorCallback,
                                  in long? duration,
                      [TreatUndefinedAs=Null]in optional DOMString? pattern)
                      raises (DeviceAPIError);

    void stopVibrate();

    PendingOperation lightOn(in SuccessCallback successCallback,
                             in ErrorCallback errorCallback,
                             in long duration)
                 raises (DeviceAPIError);

    void lightOff();

    PendingOperation setWallpaper(in SuccessCallback successCallback,
                                  in ErrorCallback errorCallback,
                                  in DOMString fileName)
                                  raises (DeviceAPIError);
  };
```

DeviceAPIError Interface

```
  [NoInterfaceObject] interface DeviceAPIError {
    readonly attribute unsigned short code;
    readonly attribute DOMString message;
    const unsigned short      UNKNOWN_ERR                      = 0;
    const unsigned short      INDEX_SIZE_ERR                   = 1;
    const unsigned short      DOMSTRING_SIZE_ERR               = 2;
    const unsigned short      HIERARCHY_REQUEST_ERR            = 3;
    const unsigned short      WRONG_DOCUMENT_ERR               = 4;
    const unsigned short      INVALID_CHARACTER_ERR            = 5;
    const unsigned short      NO_DATA_ALLOWED_ERR              = 6;
    const unsigned short      NO_MODIFICATION_ALLOWED_ERR      = 7;
    const unsigned short      NOT_FOUND_ERR                  = 8;
    const unsigned short      NOT_SUPPORTED_ERR              = 9;
    const unsigned short      INUSE_ATTRIBUTE_ERR            = 10;
    const unsigned short      INVALID_STATE_ERR              = 11;
    const unsigned short      SYNTAX_ERR                     = 12;
    const unsigned short      INVALID_MODIFICATION_ERR       = 13;
    const unsigned short      NAMESPACE_ERR                  = 14;
    const unsigned short      INVALID_ACCESS_ERR             = 15;
    const unsigned short      VALIDATION_ERR                 = 16;
    const unsigned short      TYPE_MISMATCH_ERR              = 17;
    const unsigned short      SECURITY_ERR                   = 18;
    const unsigned short      NETWORK_ERR                    = 19;
    const unsigned short      ABORT_ERR                      = 20;
    const unsigned short      TIMEOUT_ERR                    = 21;
    const unsigned short      INVALID_VALUES_ERR             = 22;
  };
```

SucessCallback Interface

```
[Callback=FunctionOnly, NoInterfaceObject] interface SuccessCallback {
    void onsuccess();
  };
```

ErrorCallback Interface

```
[Callback=FunctionOnly, NoInterfaceObject] interface ErrorCallback {
   void onerror(in DeviceAPIError error);
};
```

PendingOperation Interface

```
[NoInterfaceObject] interface PendingOperation {
   boolean cancel();
 };
```

References

[DEVICEINTERACTION]

NORMATIVE: WAC Device Interaction (Approved Release Version - June 2011) , see http://specs.wacapps.net/2.0/jun2011/deviceapis/deviceinteraction.html

APIs: The device orientation module

Webinos Specification

June 2011

Abstract

This document describes the functionality that Webinos devices should implement for retrieving information about the device orientation and motion.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The access to device orientation information is done through the W3C DeviceOrientation event specification.

2 - API

This section is NORMATIVE.

Webinos implementations MUST support W3C DeviceOrientation Event Specification

In order to use this API, access to it must be declared in the widget configuration document (i.e. config.xml). This declaration is done through the feature http://www.w3.org/ns/api-perms/deviceorientation.

Access to this functionality is achieved (as specified by W3C) through two new event types available in the `Window` object: `deviceorientation` and `devicemotion`.

3 - Security

This section is NORMATIVE.

The implementation MUST NOT enable access to this API by default, but only if the declaration is present in the widget configuration document through the appropriate feature tag.

Please note that Webinos Security Framework, depending on its configuration and in the widget level of trust, MAY deny access to this API even if it is declared in the configuration document.

4 - WebIDL

This section is INFORMATIVE.

For completeness, this specification includes a copy of the WebIDL declaration included in the W3C DeviceOrientation Event specification. The referred specification includes all the details needed to create a conformant implementation.

DeviceOrientation Interface

```
interface DeviceOrientationEvent : Event {
    readonly attribute double? alpha;
    readonly attribute double? beta;
    readonly attribute double? gamma;
    readonly attribute boolean absolute;
    void initDeviceOrientationEvent(in DOMString type,
                                    in boolean bubbles,
                                    in boolean cancelable,
                                    in double? alpha,
                                    in double? beta,
                                    in double? gamma,
                                    in boolean absolute);
    };
```

Acceleration Interface

```
    [Callback, NoInterfaceObject]
    interface Acceleration {
    readonly attribute double? x;
    readonly attribute double? y;
    readonly attribute double? z;
    };
```

RotationRate Interface

```
[Callback, NoInterfaceObject]
    interface RotationRate {
    readonly attribute double? alpha;
    readonly attribute double? beta;
    readonly attribute double? gamma;
    };
```

DeviceMotionEvent Interface

```
interface DeviceMotionEvent : Event {
    readonly attribute Acceleration? acceleration;
    readonly attribute Acceleration? accelerationIncludingGravity;
    readonly attribute RotationRate? rotationRate;
    readonly attribute double? interval;
    void initAccelerometerEvent(in DOMString type,
                                in boolean bubbles,
                                in boolean cancelable,
                                in Acceleration? acceleration,
                                in Acceleration? accelerationIncludingGravity,
                                in RotationRate? rotationRate,
                                     in double? interval);
    };
```

References

[DEVICEORIENTATION]

NORMATIVE: DeviceOrientation Event Specification (W3C Working Draft 28 June 2011) , see http://www.w3.org/TR/2011/WD-orientation-event-20110628/

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The file reader module

Webinos Specification

June 2011

Abstract

This document describes the functionality that Webinos devices should implement for reading files.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The access to File Reader is done through the W3C File Reader API [FILEREADER].

This module is accessed (as W3C specification mandates) through the blobal object (i.e. `Window`).

2 - API

This section is NORMATIVE.

Webinos implementations MUST support W3C File Reader specification [FILEREADER].

In order to use this API, access to it must be declared in the widget configuration document (i.e. config.xml). This declaration is done through the feature http://www.w3.org/ns/api-perms/file.read.

3 - Security

This section is NORMATIVE.

The implementation MUST NOT enable access to this API by default, but only if the declaration is present in the widget configuration document through the appropriate feature tag.

Please note that Webinos Security Framework, depending on its configuration and in the widget level of trust, MAY deny access to this API even if it is declared in the configuration document.

4 - WebIDL

This section is INFORMATIVE.

For completeness, this specification includes a copy of the WebIDL declaration included in the W3C File Reader specification [FILEREADER]. The referred specification includes all the details needed to create a conformant implementation.

Blob Interface

```
    interface Blob {
      readonly attribute unsigned long long size;
      readonly attribute DOMString type;
      //slice Blob into byte-ranged chunks
      Blob slice(in unsigned long long start,
                 in unsigned long long length,
                 optional DOMString contentType);
    };
```

File Interface

```
  interface File : Blob {
      readonly attribute DOMString name;
      readonly attribute DOMString lastModifiedDate;
};
```

File Reader Interface

```
[Constructor]
interface FileReader {
  // async read methods
  void readAsArrayBuffer(in Blob blob);
  void readAsBinaryString(in Blob blob);
  void readAsText(in Blob blob, [Optional] in DOMString encoding);
  void readAsDataURL(in Blob blob);
  void abort();
  // states
  const unsigned short EMPTY = 0;
  const unsigned short LOADING = 1;
  const unsigned short DONE = 2;
  readonly attribute unsigned short readyState;
  // File or Blob data
  readonly attribute any result;
  readonly attribute FileError error;
  // event handler attributes
  attribute Function onloadstart;
  attribute Function onprogress;
  attribute Function onload;
  attribute Function onabort;
  attribute Function onerror;
  attribute Function onloadend;
};
FileReader implements EventTarget;
```

FileReaderSync Interface

```
[Constructor]
interface FileReaderSync {
  // Synchronously return strings
  // All three methods raise FileException
  ArrayBuffer readAsArrayBuffer(in Blob blob);
  DOMString readAsBinaryString(in Blob blob);
  DOMString readAsText(in Blob blob, [Optional] in DOMString encoding);
  DOMString readAsDataURL(in Blob blob);
};
```

FileError Interface

```
 interface FileError {
   // File error codes
   // Found in DOMException
   const unsigned short NOT_FOUND_ERR = 1;
   const unsigned short SECURITY_ERR = 2;
   const unsigned short ABORT_ERR = 3;
   // Added by this specification
   const unsigned short NOT_READABLE_ERR = 4;
   const unsigned short ENCODING_ERR = 5;
   readonly attribute unsigned short code;
 };
```

FileException Exception

```
 exception FileException {
  const unsigned short NOT_FOUND_ERR = 1;
  const unsigned short SECURITY_ERR = 2;
  const unsigned short ABORT_ERR = 3;
  const unsigned short NOT_READABLE_ERR = 4;
  const unsigned short ENCODING_ERR = 5;
  unsigned short code;
};
```

WindowBlobURIMethods Interface

```
[Supplemental, NoInterfaceObject]
interface WindowBlobURIMethods {
    DOMString createObjectURL(in Blob blob);
    void revokeObjectURL(in DOMString url);
};
Window implements WindowBlobURIMethods;

WorkerUtils implements WindowBlobURIMethods;
```

References

[FILEREADER]

NORMATIVE: File API (W3C Working Draft 26 September 2010) , see http://www.w3.org/TR/2010/WD-FileAPI-20101026/

APIs: The file writer module

Webinos Specification

June 2011

Abstract

This document describes the functionality that Webinos devices should implement for writing to files.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The access to File Writer is done through the W3C File Writer API [FILEWRITER].

This module is accessed (as W3C specification mandates) through the global object (i.e. `Window`).

2 - API

This section is NORMATIVE.

Webinos implementations MUST support W3C File Writer specification [FILEWRITER].

In order to use this API, access to it must be declared in the widget configuration document (i.e. config.xml). This declaration is done through the feature http://www.w3.org/ns/api-perms/file.write.

3 - Security

This section is NORMATIVE.

The implementation MUST NOT enable access to this API by default, but only if the declaration is present in the widget configuration document through the appropriate feature tag.

Please note that Webinos Security Framework, depending on its configuration and in the widget level of trust, MAY deny access to this API even if it is declared in the configuration document.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

4 - WebIDL

This section is INFORMATIVE.

For completeness, this specification includes a copy of the WebIDL declaration included in the W3C File Writer specification [FILEWRITER]. The referred specification includes all the details needed to create a conformant implementation.

BlobBuilder Interface

```
[Constructor]
interface BlobBuilder {
    Blob getBlob (in optional DOMString contentType);
    void append (in DOMString text, in optional DOMString endings) raises
(FileException);
    void append (in Blob data);
    void append (in ArrayBuffer data);
};
```

FileSaver Interface

```
[Constructor(in Blob data)]
interface FileSaver {
    void abort () raises (FileException);
    const unsigned short INIT = 0;
    const unsigned short WRITING = 1;
    const unsigned short DONE = 2;
    readonly attribute unsigned short readyState;
    readonly attribute FileError      error;
             attribute Function       onwritestart;
             attribute Function       onprogress;
             attribute Function       onwrite;
             attribute Function       onabort;
             attribute Function       onerror;
             attribute Function       onwriteend;
};
```

File Writer Interface

```
[NoInterfaceObject]
interface FileWriter : FileSaver {
    readonly attribute unsigned long long position;
    readonly attribute unsigned long long length;
    void write (Blob data) raises (FileException);
    void seek (long long offset) raises (FileException);
    void truncate (unsigned long long size) raises (FileException);
};
```

FileWriterSync Interface

```
[NoInterfaceObject]
interface FileWriterSync {
    readonly attribute unsigned long long position;
    readonly attribute unsigned long long length;
    void write (Blob data) raises (FileException);
    void seek (long long offset) raises (FileException);
    void truncate (unsigned long long size) raises (FileException);
};
```

FileError Interface

```
interface FileError {
    const unsigned short NOT_FOUND_ERR = 1;
    const unsigned short SECURITY_ERR = 2;
    const unsigned short ABORT_ERR = 3;
    const unsigned short NOT_READABLE_ERR = 4;
    const unsigned short ENCODING_ERR = 5;
    const unsigned short NO_MODIFICATION_ALLOWED_ERR = 6;
    const unsigned short INVALID_STATE_ERR = 7;
    const unsigned short SYNTAX_ERR = 8;
    readonly attribute unsigned short code;
};
```

FileException Exception

```
exception FileException {
    const unsigned short NOT_FOUND_ERR = 1;
    const unsigned short SECURITY_ERR = 2;
    const unsigned short ABORT_ERR = 3;
    const unsigned short NOT_READABLE_ERR = 4;
    const unsigned short ENCODING_ERR = 5;
    const unsigned short NO_MODIFICATION_ALLOWED_ERR = 6;
    const unsigned short INVALID_STATE_ERR = 7;
    const unsigned short SYNTAX_ERR = 8;
    unsigned short code;
};
```

References

[FILEWRITER]

NORMATIVE: File API: Writer (W3C Working Draft 19 April 2011) , see http://www.w3.org/TR/2011/WD-file-writer-api-20110419/

APIs: The file system module

Webinos Specification

June 2011

Abstract

This document describes the functionality that Webinos devices should implement for accessing file system directories.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The access to File System is done through the W3C File Directories and System API [FILEDIRSYS].

This module is accessed (as W3C specification mandates) through the global object (i.e. `Window`).

2 - API

This section is NORMATIVE.

Webinos implementations MUST support W3C File Directories and System API specification [FILEDIRSYS].

In order to use this API, access to it must be declared in the widget configuration document (i.e. config.xml). This declaration is done through the feature http://www.w3.org/ns/api-perms/file.system.

3 - Security

This section is NORMATIVE.

The implementation MUST NOT enable access to this API by default, but only if the declaration is present in the widget configuration document through the appropriate feature tag.

Please note that Webinos Security Framework, depending on its configuration and in the widget level of trust, MAY deny access to this API even if it is declared in the configuration document.

4 - WebIDL

This section is INFORMATIVE.

For completeness, this specification includes a copy of the WebIDL declaration included in the W3C File Directories and System API specification [FILEDIRSYS]. The referred specification includes all the details needed to create a conformant implementation.

Metadata Interface

```
[NoInterfaceObject]
interface Metadata {
    readonly attribute Date modificationTime;
};
```

Flags Interface

```
[NoInterfaceObject]
interface Flags {
    attribute boolean create;
    attribute boolean exclusive;
};
```

FileSystem Interface

```
[NoInterfaceObject]
interface FileSystem {
    readonly attribute DOMString      name;
    readonly attribute DirectoryEntry root;
};
```

Entry Interface

```
[NoInterfaceObject]
interface Entry {
    readonly attribute boolean    isFile;
    readonly attribute boolean    isDirectory;
    void        getMetadata (MetadataCallback successCallback, optional ErrorCallback
errorCallback);
    readonly attribute DOMString  name;
    readonly attribute DOMString  fullPath;
    readonly attribute FileSystem filesystem;
    void        moveTo (DirectoryEntry parent, optional DOMString newName, optional
EntryCallback successCallback, optional ErrorCallback errorCallback);
    void        copyTo (DirectoryEntry parent, optional DOMString newName, optional
EntryCallback successCallback, optional ErrorCallback errorCallback);
    DOMString toURL (optional DOMString mimeType);
    void             remove  (VoidCallback  successCallback,  optional  ErrorCallback
errorCallback);
    void           getParent  (EntryCallback  successCallback,  optional  ErrorCallback
errorCallback);
};
```

DirectoryEntry Interface

```
[NoInterfaceObject]

interface DirectoryEntry : Entry {

    DirectoryReader createReader ();

    void             getFile (DOMString path, optional Flags options, optional
EntryCallback successCallback, optional ErrorCallback errorCallback);

    void                getDirectory (DOMString path, optional Flags options,
optional     EntryCallback     successCallback,     optional     ErrorCallback
errorCallback);

    void             removeRecursively (VoidCallback successCallback, optional
ErrorCallback errorCallback);

};
```

DirectoryReader Interface

```
[NoInterfaceObject]
interface DirectoryReader {
    void   readEntries   (EntriesCallback   successCallback,   optional   ErrorCallback
errorCallback);
};
```

FileEntry Interface

```
[NoInterfaceObject]
interface FileEntry : Entry {
    void  createWriter  (FileWriterCallback  successCallback,  optional  ErrorCallback
errorCallback);
    void file (FileCallback successCallback, optional ErrorCallback errorCallback);
};
```

FileSystemCallback Interface

```
[NoInterfaceObject, Callback=FunctionOnly]
interface FileSystemCallback {
    void handleEvent (FileSystem filesystem);
};
```

EntryCallback Interface

```
[NoInterfaceObject, Callback=FunctionOnly]
interface EntryCallback {
    void handleEvent (Entry entry);
};
```

EntriesCallback Interface

```
[NoInterfaceObject, Callback=FunctionOnly]
interface EntriesCallback {
    void handleEvent (Entry[] entries);
};
```

MetadataCallback Interface

```
[NoInterfaceObject, Callback=FunctionOnly]
interface MetadataCallback {
    void handleEvent (Metadata metadata);
};
```

FileWriterCallback Interface

```
[NoInterfaceObject, Callback=FunctionOnly]
interface FileWriterCallback {
    void handleEvent (FileWriter fileWriter);
};
```

FileCallback Interface

```
[NoInterfaceObject, Callback=FunctionOnly]
interface FileCallback {
    void handleEvent (File file);
};
```

VoidCallback Interface

```
[NoInterfaceObject, Callback=FunctionOnly]
interface VoidCallback {
    void handleEvent ();
};
```

ErrorCallback Interface

```
[NoInterfaceObject, Callback=FunctionOnly]
interface ErrorCallback {
    void handleEvent (FileError err);
};
```

FileSystemSync Interface

```
[NoInterfaceObject]
interface FileSystemSync {
    readonly attribute DOMString          name;
    readonly attribute DirectoryEntrySync root;
};
```

EntrySync Interface

```
[NoInterfaceObject]
interface EntrySync {
    readonly attribute boolean        isFile;
    readonly attribute boolean        isDirectory;
    Metadata           getMetadata () raises (FileException);
    readonly attribute DOMString      name;
    readonly attribute DOMString      fullPath;
    readonly attribute FileSystemSync filesystem;
    EntrySync           moveTo (DirectoryEntrySync parent, optional DOMString newName)
raises (FileException);
    EntrySync           copyTo (DirectoryEntrySync parent, optional DOMString newName)
raises (FileException);
    DOMString          toURL (optional DOMString mimeType);
    void               remove () raises (FileException);
    DirectoryEntrySync getParent ();
};
```

DirectoryEntrySync Interface

```
[NoInterfaceObject]
interface DirectoryEntrySync : EntrySync {
    DirectoryReaderSync createReader () raises (FileException);
    FileEntrySync           getFile  (DOMString path,  optional  Flags  options)  raises
(FileException);
    DirectoryEntrySync   getDirectory (DOMString path, optional Flags options) raises
(FileException);
    void                removeRecursively () raises (FileException);
};
```

DirectoryReaderSync Interface

```
[NoInterfaceObject]
interface DirectoryReaderSync {
    EntrySync[] readEntries () raises (FileException);
};
```

FileEntrySync Interface

```
[NoInterfaceObject]
interface FileEntrySync : EntrySync {
    FileWriterSync createWriter () raises (FileException);
    File           file () raises (FileException);
};
```

FileError Interface

```
interface FileError {
    const unsigned short NOT_FOUND_ERR = 1;
    const unsigned short SECURITY_ERR = 2;
    const unsigned short ABORT_ERR = 3;
    const unsigned short NOT_READABLE_ERR = 4;
    const unsigned short ENCODING_ERR = 5;
    const unsigned short NO_MODIFICATION_ALLOWED_ERR = 6;
    const unsigned short INVALID_STATE_ERR = 7;
    const unsigned short SYNTAX_ERR = 8;
    const unsigned short INVALID_MODIFICATION_ERR = 9;
    const unsigned short QUOTA_EXCEEDED_ERR = 10;
    const unsigned short TYPE_MISMATCH_ERR = 11;
    const unsigned short PATH_EXISTS_ERR = 12;
    attribute unsigned short code;
};
```

FileException Exception

```
exception FileException {
    const unsigned short NOT_FOUND_ERR = 1;
    const unsigned short SECURITY_ERR = 2;
    const unsigned short ABORT_ERR = 3;
    const unsigned short NOT_READABLE_ERR = 4;
    const unsigned short ENCODING_ERR = 5;
    const unsigned short NO_MODIFICATION_ALLOWED_ERR = 6;
    const unsigned short INVALID_STATE_ERR = 7;
    const unsigned short SYNTAX_ERR = 8;
    const unsigned short INVALID_MODIFICATION_ERR = 9;
    const unsigned short QUOTA_EXCEEDED_ERR = 10;
    const unsigned short TYPE_MISMATCH_ERR = 11;
    const unsigned short PATH_EXISTS_ERR = 12;
    unsigned short code;
};
```

References

[FILEDIRSYS]

NORMATIVE: File API: Directories and System (W3C Working Draft 19 April 2011) , see http://www.w3.org/TR/2011/WD-file-system-api-20110419/

APIs: The gallery module

Webinos API Specifications

1 Jul 2011

Authors

- W3C Editor's Draft 04 November 2010
- WIDL version for webinos created by Christian Fuhrhop <christian.fuhrhop@fokus.fraunhofer.de>

Abstract

W3C based Gallery API interface.

Summary of Methods

Interface	Method
Gallery	PendingOp find(DOMString [] fields, GalleryFindCB successCB, GalleryErrorCB errorCB, GalleryFindOptions options) PendingOp getGalleries(GalleryInfoCB successCB, GalleryErrorCB errorCB)
MediaObject	
GalleryInfo	
GalleryFindOptions	
GalleryFindCB	void onSuccess(MediaObject [] mediaObjectObjs)
GalleryInfoCB	void onSuccess(GalleryInfo [] galleryInfoObjs)
GalleryErrorCB	void onError(GalleryError error)
GalleryError	

1. Introduction

This specification provides a wrapper that mandates the use of the W3C Gallery API (Editor's draft 4 November 2010) that provides access to media gallery located on the device.

The Gallery API defines a high-level interface for accessing media gallery located on the device. A media gallery is a collection of media objects such as video, audio and image. *

2. Interfaces

2.1. Gallery

The Gallery interface exposes an interface to access media gallery located on the device.

```
 [NoInterfaceObject] interface Gallery {
    const unsigned short AUDIO_TYPE = 0;
    const unsigned short VIDEO_TYPE = 1;
    const unsigned short IMAGE_TYPE = 2;
    const unsigned short SORT_BY_FILENAME = 3;
    const unsigned short SORT_BY_FILEDATE = 4;
    const unsigned short SORT_BY_MEDIATYPE = 5;
    const unsigned short SORT_BY_TITLE = 6;
    const unsigned short SORT_BY_AUTHOR = 7;
    const unsigned short SORT_BY_ALBUM = 8;
    const unsigned short SORT_BY_DATE = 9;
    const unsigned short SORT_BY_ASCENDING = 10;
    const unsigned short SORT_BY_DESCENDING = 11;
    readonly attribute unsigned long length;

    caller PendingOp find (in DOMString[] fields, in GalleryFindCB successCB, in
optional GalleryErrorCB errorCB, in optional GalleryFindOptions options);

   caller PendingOp getGalleries (in GalleryInfoCB successCB, in optional
GalleryErrorCB errorCB);
};
```

Code example

```
  // append images with a title matching 'foobar' from galleries
 // not older than 3 months to the document.body

 var gallery = navigator.service.gallery;

 function getGalleriesSuccess(galleryInfoObjs) {
     var galleries = [];
     for (var i in galleryInfoObjs) {
         if ((new Date().getTime())-galleryInfoObjs[i].createDate  100*60*60*24*3) {
             galleries.push(galleryInfoObjs[i]);
         }
     }
     appendMedia(galleries);
 }

 function appendMedia(galleries) {
     function findSuccess(mediaObjs) {
         var container = document.createElement("div");
         for (var i in mediaObjs) {
             var img = document.createElement("img");
```

```
                var title = document.createElement("div");
                title.innerHTML = "Title: " + mediaObjs[i].title;
                // create blob URI using window.createObjectURL():
                // http://dev.w3.org/2006/webapi/FileAPI/#creating-revoking
                img.src = createObjectURL(mediaObjs[i]);
                container.appendChild(img);
                container.appendChild(title);
            }
            document.body.appendChild(container);
        }

        function findError() {
            console.log('whoops, something went wrong!');
        }

        gallery.find(['title', 'uri'], findSuccess, findError,
                {filter: 'foobar', galleries: galleries, mediaType: gallery.IMAGE_TYPE
});
 }

 gallery.getGalleries(getGalleriesSuccess);
```

Constants

`unsigned short AUDIO_TYPE`

Constant used to identify audio type of media.

`unsigned short VIDEO_TYPE`

Constant used to identify video type of media.

`unsigned short IMAGE_TYPE`

Constant used to identify image type of media.

`unsigned short SORT_BY_FILENAME`

Constant used to identify sort by filename.

`unsigned short SORT_BY_FILEDATE`

Constant used to identify sort by file date.

`unsigned short SORT_BY_MEDIATYPE`

Constant used to identify sort by media type.

`unsigned short SORT_BY_TITLE`

Constant used to identify sort by title.

`unsigned short SORT_BY_AUTHOR`

Constant used to identify sort by author.

`unsigned short SORT_BY_ALBUM`

Constant used to identify sort by album.

```
unsigned short SORT_BY_DATE
```

Constant used to identify sort by date

```
unsigned short SORT_BY_ASCENDING
```

Constant used to identify ascending sort order.

```
unsigned short SORT_BY_DESCENDING
```

Constant used to identify ascending sort order.

Attributes

`readonly unsigned long length`

the number of media objects in the gallery.

No exceptions.

This attribute is readonly.

Methods

`find`

Find media objects in the gallerys according to the find process detailed below.

Signature

```
caller PendingOp find(in

               DOMString

             []   fields,   in   GalleryFindCB   successCB,   in   optional
GalleryErrorCB errorCB, in optional GalleryFindOptions options);
```

This method takes two, three or four arguments. When called, it immediately returns a PendingOp object, as defined in [CORE-DEVICE], and then asynchronously starts a find process defined as follows:

1. If there are any tasks from the PendingOp task source in one of the task queues (i.e. an existing find() operation is still pending a response), and the current method was invoked with a non-null errorCB argument, dispatch an error event with a PENDING_OPERATION_ERROR code value.

2. Search for media object in the galleries

3. If the attempt was successful, dispatch a success event. If the attempt fails, and the method was invoked with a non-null errorCB argument, this method must dispatch an error event with the code attribute set according to the type of failure that has occurred.

No exceptions.

Parameters

- **fields**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** array
    - **Description:** The search qualifier.
- **successCB**
    - **Optional:** No.
    - **Nullable**: No
    - **Type:** GalleryFindCB
    - **Description:** Function to call when the asynchronous operation completes
- **errorCB**
    - **Optional:** Yes.
    - **Nullable**: No
    - **Type:** GalleryErrorCB
    - **Description:** Function to call when the asynchronous operation fails.
- **options**
    - **Optional:** Yes.
    - **Nullable**: No
    - **Type:** GalleryFindOptions
    - **Description:** The options to apply to the output of this method.

Return value

PendingOperation to cancel the asynchronous call

getGalleries

Retrieve all galleries from available sources(e.g. device local memory, external memory and even Fliker, Facebook, etc.) according to the retrieve process detailed below.

Signature

```
caller PendingOp getGalleries(in GalleryInfoCB successCB, in optional
GalleryErrorCB errorCB);
```

This method takes one or two arguments. When called, it immediately returns a PendingOp object, as defined in [CORE-DEVICE], and then asynchronously starts a retrieve process defined as follows:

1. If there are any tasks from the PendingOp task source in one of the task queues (i.e. an existing find() operation is still pending a response), and the current method was invoked with a non-null errorCB argument, dispatch an error event with a PENDING_OPERATION_ERROR code value.

2. retrieve for all galleries

3. If the attempt was successful, dispatch a success event. If the attempt fails, and the method was invoked with a non-null errorCB argument, this method must dispatch an error event with the code attribute set according to the type of failure that has occurred.Find media objects in the gallerys according to the find process detailed below.

No exceptions.

Parameters

- **successCB**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** GalleryInfoCB
  - **Description:** Function to call when the asynchronous operation completes
- **errorCB**
  - **Optional:** Yes.
  - **Nullable**: No
  - **Type:** GalleryErrorCB
  - **Description:** Function to call when the asynchronous operation fails.

Return value

PendingOperation to cancel the asynchronous call

2.2. MediaObject

The Gallery interface exposes an interface to access media gallery located on the device.

```
[NoInterfaceObject] interface MediaObject : File {
    readonly attribute unsigned long   id;
    readonly attribute GalleryInfo     gallery;
    readonly attribute DOMString?       title;
    readonly attribute DOMString?       language;
    readonly attribute DOMString?       locator;
    readonly attribute DOMString?       contributor;
    readonly attribute DOMString?       Creator;
    readonly attribute Date?            CreateDate;
    readonly attribute DOMString?       location;
    readonly attribute DOMString?       description;
    readonly attribute DOMString?       keyword;
    readonly attribute DOMString?       genre;
    readonly attribute unsigned long?   rating;
    readonly attribute DOMString?       relation;
    readonly attribute DOMString?       collection;
    readonly attribute DOMString?       copyright;
    readonly attribute DOMString?       policy;
    readonly attribute DOMString?       publisher;
    readonly attribute DOMString?       targetAudience;
    readonly attribute DOMString?       fragment;
    readonly attribute DOMString?       namedFragment;
    readonly attribute unsigned long?   frameSize;
    readonly attribute DOMString?       compression;
    readonly attribute unsigned long?   duration;
    readonly attribute DOMString?       format;
    readonly attribute unsigned long?   samplingRate;
    readonly attribute unsigned long?   framerate;
    readonly attribute unsigned long?   averageBitRate;
    readonly attribute unsigned short?  numTracks;
};
```

Attributes

`readonly unsigned long id`

Unique id of media object. This id is a unique numeric identifiers of the object. This id is persistent while the gallery is opened.

No exceptions.

This attribute is readonly.

`readonly GalleryInfo gallery`

gallery information associated to the media object.

No exceptions.

This attribute is readonly.

`readonly DOMString? title`

The title of the media object.

No exceptions.

This attribute is readonly.

`readonly DOMString? language`

The language used in the media object.

No exceptions.

This attribute is readonly.

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

`readonly DOMString? locator`

The logical address at which the media object can be accessed.

No exceptions.

This attribute is readonly.

`readonly DOMString? contributor`

The contributor related with the media object. e.g., actor, cameraman, director, singer, author, artist, etc.

No exceptions.

This attribute is readonly.

`readonly DOMString? Creator`

The author of the media object.

No exceptions.

This attribute is readonly.

`readonly Date? CreateDate`

The date and time the media object was originally created.

No exceptions.

This attribute is readonly.

`readonly DOMString? location`

The description where the media object has been created, developed, recorded, or otherwise authored.

No exceptions.

This attribute is readonly.

`readonly DOMString? description`

A free-form text describing the content of the media object.

No exceptions.

This attribute is readonly.

`readonly DOMString? keyword`

A concept, descriptive phrase or keyword that specifies the topic of the media object.

No exceptions.

This attribute is readonly.

`readonly DOMString? genre`

The category of the content of the media object.

No exceptions.

This attribute is readonly.

`readonly unsigned long? rating`

The rating value related with the media object.

No exceptions.

This attribute is readonly.

`readonly DOMString? relation`

The description related with resource that the current media object is related with.

No exceptions.

This attribute is readonly.

`readonly DOMString? collection`

The name of the collection from which the media object originates or to which it belongs.

No exceptions.

This attribute is readonly.

`readonly DOMString? copyright`

The copyright statement. Identification of the copyrights holder.

No exceptions.

This attribute is readonly.

`readonly DOMString? policy`

A policy statement (typically human-readable) associated with the media object.

No exceptions.

This attribute is readonly.

`readonly DOMString? publisher`

The publisher of a media object.

No exceptions.

This attribute is readonly.

`readonly DOMString? targetAudience`

The classification information related with media object including the issuer of the classification (e.g., a parental guidance issuing agency, or a targeted geographical region).

No exceptions.

This attribute is readonly.

`readonly DOMString? fragment`

A fragment identifier. A fragment is a portion of the resource

No exceptions.

This attribute is readonly.

`readonly DOMString? namedFragment`

A named fragment identifier.

No exceptions.

This attribute is readonly.

`readonly unsigned long? frameSize`

The frame size of the media object (e.g., width and height of 720 and 480 units, respectively).

No exceptions.

This attribute is readonly.

`readonly DOMString? compression`

The compression type used.

No exceptions.

This attribute is readonly.

`readonly unsigned long? duration`

The actual duration of the media object.

No exceptions.

This attribute is readonly.

`readonly DOMString? format`

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

The MIME type of the media object (e.g., wrapper or bucket media types).

No exceptions.

This attribute is readonly.

`readonly unsigned long? samplingRate`

The audio sampling rate.

No exceptions.

This attribute is readonly.

`readonly unsigned long? framerate`

The video frame rate.

No exceptions.

This attribute is readonly.

`readonly unsigned long? averageBitRate`

The average bit rate.

No exceptions.

This attribute is readonly.

`readonly unsigned short? numTracks`

The number of tracks of a resource.

No exceptions.

This attribute is readonly.

2.3. GalleryInfo

The GalleryInfo exposes an interface to capture generic metadata information of a gallery.

```
[NoInterfaceObject] interface GalleryInfo {
    readonly attribute DOMString   title;
    readonly attribute Date        createdDate;
    readonly attribute DOMString   location;
    readonly attribute DOMString[] description;
    readonly attribute DOMString[] supportedMediaObjectType;
};
```

Attributes

`readonly DOMString title`

The title of the gallery.

No exceptions.

This attribute is readonly.

`readonly Date createdDate`

The date and time the gallery was originally created.

No exceptions.

This attribute is readonly.

`readonly DOMString location`

The location the gallery is located on.

No exceptions.

This attribute is readonly.

`readonly DOMString [] description`

The description of the gallery.

No exceptions.

This attribute is readonly.

`readonly DOMString [] supportedMediaObjectType`

A list of media object type supported by this gallery.

No exceptions.

This attribute is readonly.

2.4. GalleryFindOptions

The GalleryFindOptions exposes an interface to describe the options that can be applied to media object searching and displaying.

```
[NoInterfaceObject] interface GalleryFindOptions {
    attribute DOMString?      filter;
    attribute short?          mediaType;
    attribute GalleryInfo[]?  gallery;
    attribute short?          order;
    attribute short?          firstSortOption;
    attribute short?          secondSortOption;
    attribute Date?           startDate;
    attribute Date?           endDate;
};
```

Attributes

`DOMString? filter`

A DOMString-based search filter with which to search. It's working based on the metadata of media object.

No exceptions.

`short? mediaType`

Specify the scope of media type for finding the media object

No exceptions.

`GalleryInfo [] gallery`

Specify the scope of gallery for finding the media object

No exceptions.

`short? order`

Specify wheither media objects are ordered in ascending or descending order. Default is an ascending order.

No exceptions.

`short? firstSortOption`

Primary criteria to order the media object of the gallery.

No exceptions.

`short? secondSortOption`

Second criteria to order the media object of the gallery.

No exceptions.

`Date? startDate`

Start date for performing the search. Media object with date previous to that date will not be returned.

No exceptions.

`Date? endDate`

End date for performing the search. Media object with date later to that date will not be returned.

No exceptions.

2.5. GalleryFindCB

find specific success callback.

```
[Callback=FunctionOnly, NoInterfaceObject] interface GalleryFindCB : PendingOp {
        void onSuccess (in MediaObject[] mediaObjectObjs);
};
```

Methods

`onSuccess`

Method invoked when the asynchronous call completes successfully

Signature
```
void onSuccess(in

              MediaObject

            [] mediaObjectObjs);
```

No exceptions.

Parameters

- **mediaObjectObjs**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** array
  - **Description:** The Media Object resulting from the given Gallery find() method.

Return value
void

2.6. GalleryInfoCB

getGalleries specific success callback.

```
[Callback=FunctionOnly, NoInterfaceObject] interface GalleryInfoCB : PendingOp {
    void onSuccess (in GalleryInfo[] galleryInfoObjs);
};
```

Methods

onSuccess

Method invoked when the asynchronous call completes successfully

Signature
```
void onSuccess(in

              GalleryInfo

            [] galleryInfoObjs);
```

No exceptions.

Parameters

- **galleryInfoObjs**
  - **Optional:** No.
  - **Nullable**: No

- **Type:** array
- **Description:** The GalleryInfo Objects resulting from the given Gallery getGalleries() method.

Return value

void

2.7. GalleryErrorCB

Gallery API specific error callback.

```
[Callback=FunctionOnly, NoInterfaceObject] interface GalleryErrorCB : PendingOp {
    void onError (in GalleryError error);
};
```

Methods

onError

Method invoked when the asynchronous call completes unsuccessfully

Signature

```
void onError(in GalleryError error);
```

No exceptions.

Parameters

- **error**
  - **Optional:** No.
  - **Nullable**: No
  - **Type:** GalleryError
  - **Description:** The Gallery API related error object.

Return value

void

2.8. GalleryError

The GalleryError interface encapsulates all errors in the Gallery API.

```
[NoInterfaceObject] interface GalleryError {
    const unsigned short UNKNOWN_ERROR = 0;
    const unsigned short INVALID_ARGUMENT_ERROR = 1;
    const unsigned short TIMEOUT_ERROR = 2;
    const unsigned short PENDING_OPERATION_ERROR = 3;
    const unsigned short IO_ERROR = 4;
    const unsigned short NOT_SUPPORTED_ERROR = 5;
    const unsigned short PERMISSION_DENIED_ERROR = 20;
    readonly attribute unsigned short code;
};
```

Constants

`unsigned short UNKNOWN_ERROR`

An unknown error occurred.

`unsigned short INVALID_ARGUMENT_ERROR`

An invalid parameter was provided when the requested method was invoked.

`unsigned short TIMEOUT_ERROR`

The requested method timed out before it could be completed.

`unsigned short PENDING_OPERATION_ERROR`

If the user agent is currently waiting for a callback on a current find() operation, as defined in this specification.

`unsigned short IO_ERROR`

An error occurred in communication with the underlying implementation that meant the requested method could not complete.

`unsigned short NOT_SUPPORTED_ERROR`

The requested method is not supported by the current implementation.

`unsigned short PERMISSION_DENIED_ERROR`

Attributes

`readonly unsigned short code`

An error code assigned by an implementation when an error has occurred in Gallery API processing.

No exceptions.

This attribute is readonly.

3. Features

4. Full WebIDL

```
module gallery {

 [NoInterfaceObject] interface Gallery {
    const unsigned short AUDIO_TYPE = 0;
    const unsigned short VIDEO_TYPE = 1;
    const unsigned short IMAGE_TYPE = 2;
    const unsigned short SORT_BY_FILENAME = 3;
    const unsigned short SORT_BY_FILEDATE = 4;
    const unsigned short SORT_BY_MEDIATYPE = 5;
    const unsigned short SORT_BY_TITLE = 6;
    const unsigned short SORT_BY_AUTHOR = 7;
    const unsigned short SORT_BY_ALBUM = 8;
    const unsigned short SORT_BY_DATE = 9;
    const unsigned short SORT_BY_ASCENDING = 10;
    const unsigned short SORT_BY_DESCENDING = 11;
    readonly attribute unsigned long length;

    caller PendingOp find (in DOMString[] fields, in GalleryFindCB successCB, in optional GalleryErrorCB errorCB, in optional GalleryFindOptions options);

   caller PendingOp getGalleries (in GalleryInfoCB successCB, in optional GalleryErrorCB errorCB);
};

[NoInterfaceObject] interface MediaObject : File {
    readonly attribute unsigned long    id;
    readonly attribute GalleryInfo      gallery;
    readonly attribute DOMString?       title;
    readonly attribute DOMString?       language;
    readonly attribute DOMString?       locator;
    readonly attribute DOMString?       contributor;
    readonly attribute DOMString?       Creator;
    readonly attribute Date?            CreateDate;
    readonly attribute DOMString?       location;
    readonly attribute DOMString?       description;
    readonly attribute DOMString?       keyword;
    readonly attribute DOMString?       genre;
    readonly attribute unsigned long?   rating;
    readonly attribute DOMString?       relation;
    readonly attribute DOMString?       collection;
    readonly attribute DOMString?       copyright;
    readonly attribute DOMString?       policy;
    readonly attribute DOMString?       publisher;
    readonly attribute DOMString?       targetAudience;
    readonly attribute DOMString?       fragment;
    readonly attribute DOMString?       namedFragment;
    readonly attribute unsigned long?   frameSize;
    readonly attribute DOMString?       compression;
    readonly attribute unsigned long?   duration;
    readonly attribute DOMString?       format;
    readonly attribute unsigned long?   samplingRate;
    readonly attribute unsigned long?   framerate;
    readonly attribute unsigned long?   averageBitRate;
    readonly attribute unsigned short?  numTracks;
};

[NoInterfaceObject] interface GalleryInfo {
    readonly attribute DOMString   title;
    readonly attribute Date        createdDate;
    readonly attribute DOMString   location;
    readonly attribute DOMString[] description;
    readonly attribute DOMString[] supportedMediaObjectType;
};

[NoInterfaceObject] interface GalleryFindOptions {
    attribute DOMString?      filter;
    attribute short?          mediaType;
    attribute GalleryInfo[]?  gallery;
    attribute short?          order;
    attribute short?          firstSortOption;
    attribute short?          secondSortOption;
    attribute Date?           startDate;
    attribute Date?           endDate;
};

[Callback=FunctionOnly, NoInterfaceObject] interface GalleryFindCB : PendingOp {
       void onSuccess (in MediaObject[] mediaObjectObjs);
};

[Callback=FunctionOnly, NoInterfaceObject] interface GalleryInfoCB : PendingOp {
    void onSuccess (in GalleryInfo[] galleryInfoObjs);
};

[Callback=FunctionOnly, NoInterfaceObject] interface GalleryErrorCB : PendingOp {
    void onError (in GalleryError error);
};

[NoInterfaceObject] interface GalleryError {
    const unsigned short UNKNOWN_ERROR = 0;
    const unsigned short INVALID_ARGUMENT_ERROR = 1;
    const unsigned short TIMEOUT_ERROR = 2;
    const unsigned short PENDING_OPERATION_ERROR = 3;
    const unsigned short IO_ERROR = 4;
    const unsigned short NOT_SUPPORTED_ERROR = 5;
    const unsigned short PERMISSION_DENIED_ERROR = 20;
    readonly attribute unsigned short code;
};

};
```

APIs: The geolocation module

Webinos Specification

June 2011

Abstract

This document describes the functionality that Webinos devices should implement for retrieving geographical information.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The access to Geolocation is done through the W3C Geolocation API [GEOLOCATION].

This module is loaded (as W3C specification mandates) in the Navigator interface with the name geolocation (i.e. navigator.geolocation).

2 - API

This section is NORMATIVE.

Webinos implementations MUST support W3C Geolocation specification [GEOLOCATION].

In order to use this API, access to it must be declared in the widget configuration document (i.e. config.xml). This declaration is done through the feature http://www.w3.org/ns/api-perms/geolocation.

3 - Security

This section is NORMATIVE.

The implementation MUST NOT enable access to this API by default, but only if the declaration is present in the widget configuration document through the appropriate feature tag.

Please note that Webinos Security Framework, depending on its configuration and in the widget level of trust, MAY deny access to this API even if it is declared in the configuration document.

4 - WebIDL

This section is INFORMATIVE.

For completeness, this specification includes a copy of the WebIDL declaration included in the W3C Geolocation specification [GEOLOCATION]. The referred specification includes all the details needed to create a conformant implementation.

Geolocation Instatiation

```
 [NoInterfaceObject]
 interface NavigatorGeolocation {
   readonly attribute Geolocation geolocation;
 };

 Navigator implements NavigatorGeolocation;
```

Geolocation Interface

```
[NoInterfaceObject]
 interface Geolocation {
   void getCurrentPosition(in PositionCallback successCallback,
                           in optional PositionErrorCallback errorCallback,
                           in optional PositionOptions options);

   long watchPosition(in PositionCallback successCallback,
                      in optional PositionErrorCallback errorCallback,
                      in optional PositionOptions options);

   void clearWatch(in long watchId);
 };

 [Callback=FunctionOnly, NoInterfaceObject]
 interface PositionCallback {
   void handleEvent(in Position position);
 };

 [Callback=FunctionOnly, NoInterfaceObject]
 interface PositionErrorCallback {
   void handleEvent(in PositionError error);
 };
```

Position Options

```
  [Callback, NoInterfaceObject]
  interface PositionOptions {
    attribute boolean enableHighAccuracy;
    attribute long timeout;
    attribute long maximumAge;
  };
```

Position Interface

```
 interface Position {
    readonly attribute Coordinates coords;
    readonly attribute DOMTimeStamp timestamp;
  };
```

Coordinates Interface

```
interface Coordinates {
    readonly attribute double latitude;
    readonly attribute double longitude;
    readonly attribute double? altitude;
    readonly attribute double accuracy;
    readonly attribute double? altitudeAccuracy;
    readonly attribute double? heading;
    readonly attribute double? speed;
  };
```

PositionError Interface

```
interface PositionError {
    const unsigned short PERMISSION_DENIED = 1;
    const unsigned short POSITION_UNAVAILABLE = 2;
    const unsigned short TIMEOUT = 3;
    readonly attribute unsigned short code;
    readonly attribute DOMString message;
  };
```

References

[GEOLOCATION]

NORMATIVE: Geolocation API Specification (W3C Candidate Recommendation 07 September 2010) , see http://www.w3.org/TR/2010/CR-geolocation-API-20100907/

This work is partially funded by webinos, an EU-funded project under the EU FP7 ICT Programme, No 257103.

APIs: The media capture module

Webinos Specification

June 2011

Abstract

This document describes the functionality that Webinos devices should implement for capturing media.

1 - Introduction

This section is INFORMATIVE.

Webinos is fully committed to the use of Open Standards whenever available. The access to capture media capability is done through the W3C Media Capture API [MEDIACAPTURE].

2 - API

This section is NORMATIVE.

Webinos implementations MUST support W3C Media Capture API specification [MEDIACAPTURE].

In order to use this API, access to it must be declared in the widget configuration document (i.e. config.xml). This declaration is done through the feature http://www.w3.org/ns/api-perms/mediacapture.

3 - Security

This section is NORMATIVE.

The implementation MUST NOT enable access to this API by default, but only if the declaration is present in the widget configuration document through the appropriate feature tag.

Please note that Webinos Security Framework, depending on its configuration and in the widget level of trust, MAY deny access to this API even if it is declared in the configuration document.

4 - WebIDL

This section is INFORMATIVE.

For completeness, this specification includes a copy of the WebIDL declaration included in the W3C Media Capture API specification [MEDIACAPTURE]. The referred specification includes all the details needed to create a conformant implementation.

DeviceCapture Interface

```
[NoInterfaceObject]
interface DeviceCapture {
    readonly attribute Capture capture;
};
Device implements DeviceCapture;
```

Capture Interface

```
[Supplemental, NoInterfaceObject]
interface Capture {
    readonly attribute MediaFileData[] supportedImageFormats;
    readonly attribute MediaFileData[] supportedVideoFormats;
    readonly attribute MediaFileData[] supportedAudioFormats;
    PendingOperation captureImage (in CaptureCB successCB, in optional CaptureErrorCB
errorCB, in optional CaptureImageOptions options);
    PendingOperation captureVideo (in CaptureCB successCB, in optional CaptureErrorCB
errorCB, in optional CaptureVideoOptions options);
    PendingOperation captureAudio (in CaptureCB successCB, in optional CaptureErrorCB
errorCB, in optional CaptureAudioOptions options);
};
```

CaptureCB Interface

```
[Callback=FunctionOnly, NoInterfaceObject]
interface CaptureCB {
    void onSuccess (in FileList capturedMedia);
};
```

CaptureErrorCB Interface

```
[Callback=FunctionOnly, NoInterfaceObject]
interface CaptureErrorCB {
    void onError (in CaptureError error);
};
```

CaptureError Interface

```
[NoInterfaceObject]
interface CaptureError {
    const unsigned short CAPTURE_INTERNAL_ERR = 0;
    const unsigned short CAPTURE_APPLICATION_BUSY = 1;
    const unsigned short CAPTURE_INVALID_ARGUMENT = 2;
    const unsigned short CAPTURE_NO_MEDIA_FILES = 3;
    readonly attribute unsigned short code;
};
```

CaptureImageOptions Interface

```
[NoInterfaceObject]
interface CaptureImageOptions {
    attribute unsigned long limit;
};
```

CaptureVideoOptions Interface

```
[NoInterfaceObject]
interface CaptureVideoOptions {
    attribute unsigned long limit;
    attribute float duration;
};
```

CaptureAudioOptions Interface

```
[NoInterfaceObject]
interface CaptureAudioOptions {
    attribute unsigned long limit;
};
```

PendingOperation Interface

```
[NoInterfaceObject]
interface PendingOperation {
    void cancel ();
};
```

References

[MEDIACAPTURE]

NORMATIVE: Media Capture API (W3C Working Draft 28 September 2008) , see http://www.w3.org/TR/2010/WD-media-capture-api-20100928/