



Web Runtime API (WRAPI) – Design Patterns

Approved Version 1.0 – 23 Sep 2014

Open Mobile Alliance
OMA-TS-WRAPI_Design_Patterns-V1_0-20140923-A

Use of this document is subject to all of the terms and conditions of the Use Agreement located at <http://www.openmobilealliance.org/UseAgreement.html>.

Unless this document is clearly designated as an approved specification, this document is a work in process, is not an approved Open Mobile Alliance™ specification, and is subject to revision or removal without notice.

You may use this document or any part of the document for internal or educational purposes only, provided you do not modify, edit or take out of context the information in this document in any manner. Information contained in this document may be used, at your sole risk, for any purposes. You may not use this document in any other manner without the prior written permission of the Open Mobile Alliance. The Open Mobile Alliance authorizes you to copy this document, provided that you retain all copyright and other proprietary notices contained in the original materials on any copies of the materials and that you comply strictly with these terms. This copyright permission does not constitute an endorsement of the products or services. The Open Mobile Alliance assumes no responsibility for errors or omissions in this document.

Each Open Mobile Alliance member has agreed to use reasonable endeavors to inform the Open Mobile Alliance in a timely manner of Essential IPR as it becomes aware that the Essential IPR is related to the prepared or published specification. However, the members do not have an obligation to conduct IPR searches. The declared Essential IPR is publicly available to members and non-members of the Open Mobile Alliance and may be found on the “OMA IPR Declarations” list at <http://www.openmobilealliance.org/ipr.html>. The Open Mobile Alliance has not conducted an independent IPR review of this document and the information contained herein, and makes no representations or warranties regarding third party IPR, including without limitation patents, copyrights or trade secret rights. This document may contain inventions for which you must obtain licenses from third parties before making, using or selling the inventions. Defined terms above are set forth in the schedule to the Open Mobile Alliance Application Form.

NO REPRESENTATIONS OR WARRANTIES (WHETHER EXPRESS OR IMPLIED) ARE MADE BY THE OPEN MOBILE ALLIANCE OR ANY OPEN MOBILE ALLIANCE MEMBER OR ITS AFFILIATES REGARDING ANY OF THE IPR'S REPRESENTED ON THE “OMA IPR DECLARATIONS” LIST, INCLUDING, BUT NOT LIMITED TO THE ACCURACY, COMPLETENESS, VALIDITY OR RELEVANCE OF THE INFORMATION OR WHETHER OR NOT SUCH RIGHTS ARE ESSENTIAL OR NON-ESSENTIAL.

THE OPEN MOBILE ALLIANCE IS NOT LIABLE FOR AND HEREBY DISCLAIMS ANY DIRECT, INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF DOCUMENTS AND THE INFORMATION CONTAINED IN THE DOCUMENTS.

© 2014 Open Mobile Alliance Ltd. All Rights Reserved.

Used with the permission of the Open Mobile Alliance Ltd. under the terms set forth above.

Contents

1. SCOPE	6
2. REFERENCES	7
2.1 NORMATIVE REFERENCES	7
2.2 INFORMATIVE REFERENCES	7
3. TERMINOLOGY AND CONVENTIONS	8
3.1 CONVENTIONS	8
3.2 DEFINITIONS	8
3.3 ABBREVIATIONS	8
4. INTRODUCTION	10
4.1 VERSION 1.0	10
5. SPECIFICATION TEMPLATE	11
6. USE OF WEB IDL	12
6.1 INCLUSION IN SPECIFICATIONS	12
6.1.1 General Web IDL Specifications	12
6.1.2 Web IDL Interface Specification	13
6.2 WEB IDL BINDINGS	13
6.3 USE OF WEB IDL FEATURES IN API SPECIFICATIONS	13
6.3.1 Modules	13
6.3.2 Interfaces.....	13
6.3.3 Exceptions.....	14
6.3.4 Typedefs.....	14
6.3.5 Implements Statements	15
6.3.6 Types.....	15
6.3.7 Extended Attributes	15
7. ASYNCHRONOUS METHODS	16
7.1 TYPES OF ASYNCHRONOUS METHODS	16
7.1.1 Function-Only Success Callback Methods	16
7.1.2 Interface Success Callback Methods.....	16
7.2 PENDING OPERATIONS	17
7.2.1 PendingOperation Interface	17
7.3 SUCCESS CALLBACKS	18
7.3.1 SuccessCallback Interface.....	18
7.4 ERROR CALLBACKS	19
7.4.1 errorCallback Interface	19
8. ERROR HANDLING	21
8.1 OMAAPIERROR INTERFACE	21
8.2 INTERFACE ERRORS AND EXCEPTIONS	23
8.2.1 Synchronous Interface Errors.....	23
8.2.2 Asynchronous Interface Errors	23
9. ARGUMENTS	25
10. ACCESSING APIS	26
10.1 DEFINING OMA WEB RUNTIME APIS AS PART OF THE WINDOW GLOBAL OBJECT	26
10.1.1 OmaapiObject Interface	26
10.1.2 Omaapi Interface.....	26
10.2 ACCESSING API FEATURE INFORMATION	28
10.2.1 Feature Interface	28
10.2.2 Param Interface	28
11. FULL WEB IDL	29
APPENDIX A. CHANGE HISTORY (INFORMATIVE)	31

A.1	APPROVED VERSION HISTORY	31
APPENDIX B.	STATIC CONFORMANCE REQUIREMENTS (NORMATIVE)	32
B.1	SCR FOR USER AGENT	32

Figures

Figure 1:	Web IDL Example from XMLHttpRequest	12
-----------	---	----

Tables

Table 1:	Example Web IDL Specification	13
Table 2:	Common Typedefs	15
Table 3:	Function-Only Success Callback Method Parameters	16
Table 4:	Interface Success Callback Method Parameters	16
Table 5:	PendingOperation Interface Web IDL Specification.....	17
Table 6:	PendingOperation code example	17
Table 7:	cancel() method signature	18
Table 8:	SuccessCallback Interface Web IDL Specification.....	18
Table 9:	onsuccess() method signature.....	18
Table 10:	onsuccess() method code example.....	19
Table 11:	ErrorCallback Interface Web IDL Specification.....	19
Table 12:	onerror() method signature.....	19
Table 13:	onerror() method code example.....	20
Table 14:	OmaAPIError Web IDL Specification	21
Table 15:	OmaAPIError code example	21
Table 16:	Example of Error Handling Definition in Web IDL for Synchronous Interfaces.....	23
Table 17:	Example of Error Usage Documentation for Synchronous Interfaces.....	23
Table 18:	Example of Error Handling Definition in Web IDL for Asynchronous Interfaces	24
Table 19:	Example of Error Usage Documentation for Asynchronous Interfaces	24
Table 20:	OmaapiObject Web IDL Specification	26
Table 21:	Omaapi interface Web IDL Specification.....	26
Table 22:	listAvailableFeatures() method signature	26
Table 23:	listAvailableFeatures() code example.....	27
Table 24:	listActivatedFeatures() method signature.....	27
Table 25:	listActivatedFeatures() code example.....	27
Table 26:	Feature interface Web IDL Specification	28

Table 27: Param interface Web IDL Specification.....28

1. Scope

This specification defines application programming interface (API) design patterns to be used with APIs exposed to applications executing under Web runtime environments.

2. References

2.1 Normative References

- [DOM3Core] “Document Object Model (DOM) Level 3 Core Specification”, W3C Recommendation 07 April 2004, [URL: http://www.w3.org/TR/DOM-Level-3-Core/](http://www.w3.org/TR/DOM-Level-3-Core/)
- [RFC2119] “Key words for use in RFCs to Indicate Requirement Levels”, S. Bradner, March 1997, [URL:http://www.ietf.org/rfc/rfc2119.txt](http://www.ietf.org/rfc/rfc2119.txt)
- [RFC2396] “Uniform Resource Identifiers (URI): Generic Syntax”, T. Berners-Lee et al. August 1998. [URL: http://www.ietf.org/rfc/rfc2396.txt](http://www.ietf.org/rfc/rfc2396.txt)
- [RFC2616] “Hypertext Transfer Protocol -- HTTP/1.1”, R. Fielding et. al, January 1999, [URL:http://www.ietf.org/rfc/rfc2616.txt](http://www.ietf.org/rfc/rfc2616.txt)
- [RFC4627] “The application/json Media Type for JavaScript Object Notation (JSON)”, D. Crockford, July 2006, [URL:http://www.ietf.org/rfc/rfc4627.txt](http://www.ietf.org/rfc/rfc4627.txt)
- [SCR RULES] “SCR Rules and Procedures”, Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, [URL:http://www.openmobilealliance.org/](http://www.openmobilealliance.org/)
- [W3C-URLENC] W3C HTML 2.0 Specification, form-urlencoded Media Type, [URL: http://www.w3.org/MarkUp/html-spec/html-spec_8.html#SEC8.2.1](http://www.w3.org/MarkUp/html-spec/html-spec_8.html#SEC8.2.1)
- [W3C-WebIDL] “Web IDL”, W3C, [URL: http://www.w3.org/TR/WebIDL/](http://www.w3.org/TR/WebIDL/)
- [W3C-Widgets] “Widget Packaging and XML Configuration”, W3C, [URL: http://www.w3.org/TR/widgets/](http://www.w3.org/TR/widgets/)
- [WAC2.0] “WAC 2.0”, WAC, February 2011, [URL: http://wacapps.net](http://wacapps.net)
- [WAC-API-Patterns] “WAC 2.0: Guidelines and Patterns for API Definition”, WAC, February 2011, [URL: http://wacapps.net](http://wacapps.net)
- [XMLSchema1] W3C Recommendation, XML Schema Part 1: Structures Second Edition, [URL: http://www.w3.org/TR/xmlschema-1/](http://www.w3.org/TR/xmlschema-1/)
- [XMLSchema2] W3C Recommendation, XML Schema Part 2: Datatypes Second Edition, [URL: http://www.w3.org/TR/xmlschema-2/](http://www.w3.org/TR/xmlschema-2/)

2.2 Informative References

- [OMADICT] “Dictionary for OMA Specifications”, Version 2.7, Open Mobile Alliance™, OMA-ORG-Dictionary-V2_7, [URL:http://www.openmobilealliance.org/](http://www.openmobilealliance.org/)
- [OMNA] “OMA Naming Authority”. Open Mobile Alliance™. [URL: http://www.openmobilealliance.org/OMNA.aspx](http://www.openmobilealliance.org/OMNA.aspx)

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

API Patterns	Design guidelines and requirements for definition of APIs
Browser Context	Web applications executing under a Web browser as Web runtime environment.
ECMAScript	Use definition from [OMADICT].
JavaScript	Use definition from [OMADICT].
Uniform Resource Identifier	Use definition from [OMADICT].
User Agent	Use definition from [OMADICT].
Web	The World Wide Web, a content and application framework based upon hypertext and related technologies, e.g. XML, JavaScript/ECMAScript, CSS, etc.
Web Application	An application designed using Web technologies.
Web IDL	An IDL language for Web application APIs
Web Runtime	Client software that supports the execution of Web Applications
Widget Context	Web applications installed and executing under a W3C Widget [W3C-Widgets] engine as Web runtime environment.

3.3 Abbreviations

API	Application Programming Interface
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
OMA	Open Mobile Alliance
REST	REpresentational State Transfer
SCR	Static Conformance Requirements
SMS	Short Message Service
TS	Technical Specification
UA	User Agent
UE	User Equipment
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAC	Wholesale Applications Community

WRAPI	The OMA Web Runtime API enabler
WRT	Web runtime environment
XML	eXtensible Markup Language
XSD	XML Schema Definition

4. Introduction

This specification contains common design guidelines and requirements (“API Patterns”) intended for Web runtime APIs (referred to as simply “APIs” in this document) to be defined by the OMA. These API Patterns are a normative dependency for the specification of OMA-defined APIs exposed to applications executing under Web runtime environments.

The intent of this specification is to promote consistency in the technical approach to definition of APIs exposing OMA enabler-based services.

4.1 Version 1.0

Version 1.0 of the WRAPI Design Patterns specification addresses the following aspects:

- Use of Web IDL for API specification
- Asynchronous methods
- Error handling
- Arguments
- Accessing APIs

5. Specification Template

OMA Web runtime API specifications should use the following general outline:

- Introduction
- Interfaces: specification of all interfaces defined in the Web runtime API module, including Web IDL, description, and Javascript usage examples
- Type Definitions: specification of all type definitions defined in the Web runtime API module, including Web IDL and description
- Features: specification of
 - The feature identifier (feature URL under the OMA namespace <http://openmobilealliance.org/wrapi/>)
 - How the feature is instantiated, and the resulting object appears in the global namespace of the Web runtime
 - URIs used to declare use of the API features within the config.xml document of widget context Web applications [W3C-Widgets], including what specific aspects of the API features (e.g. interface methods) are associated with the URI
- Full WebIDL: the full Web IDL defined in the specification

6. Use of Web IDL

The W3C has defined Web IDL as “an interface definition language, *Web IDL*, that can be used to describe interfaces that are intended to be implemented in web browsers. *Web IDL* is an IDL variant with a number of features that allow the behavior of common script objects in the web platform to be specified more readily. How interfaces described with *Web IDL* correspond to constructs within *ECMAScript* and *Java* execution environments is also detailed.” [WebIDL].

6.1 Inclusion in Specifications

6.1.1 General Web IDL Specifications

Web IDL is typically included in specifications to precisely define interfaces, supplemented with text defining user agent behavior related to the interfaces and other clarifying information. To clearly identify the Web IDL from the surrounding text, it is typically included in a grayed/bordered text box and presented in a fixed-type font (e.g. Consolas), e.g. as in the XMLHttpRequest API specified by W3C:

```
[NoInterfaceObject]
interface XMLHttpRequestEventTarget : EventTarget {
    // for future use
};

[Constructor]
interface XMLHttpRequest : XMLHttpRequestEventTarget {
    // event handler attributes
    attribute Function onreadystatechange;

    // states
    const unsigned short UNSENT = 0;
    const unsigned short OPENED = 1;
    const unsigned short HEADERS_RECEIVED = 2;
    const unsigned short LOADING = 3;
    const unsigned short DONE = 4;
    readonly attribute unsigned short readyState;

    // request
    void open(DOMString method, DOMString url);
    void open(DOMString method, DOMString url, boolean async);
    void open(DOMString method, DOMString url, boolean async, DOMString? user);
    void open(DOMString method, DOMString url, boolean async, DOMString? user, DOMString? password);
    void setRequestHeader(DOMString header, DOMString value);
    void send();
    void send(Document data);
    void send([AllowAny] DOMString? data);
    void abort();

    // response
    readonly attribute unsigned short status;
    readonly attribute DOMString statusText;
    DOMString getResponseHeader(DOMString header);
    DOMString getAllResponseHeaders();
    readonly attribute DOMString responseText;
    readonly attribute Document responseXML;
};
```

Figure 1: Web IDL Example from XMLHttpRequest

The Web IDL box is typically not captioned in W3C specifications, but this is recommended for OMA specifications to allow direct access to the Web IDL in the specification through the table index. In addition, the inclusion of a header line for the box should clearly identify the contents as a Web IDL specification of an interface. The example below is formatted as a 1x1 cell table so that a caption can be added to the specification.

Web IDL Specification

```
[NoInterfaceObject]
interface XMLHttpRequestEventTarget : EventTarget {
    // for future use
};
...
```

Table 1: Example Web IDL Specification

6.1.2 Web IDL Interface Specification

Web IDL interfaces should be documented using the template shown for the Omaapi interface as an example in section 10.1.2.

6.2 Web IDL Bindings

Web IDL is designed for binding in JavaScript and Java implementations. While WRAPI APIs are primarily intended to support interface access for Web Applications executing in Web browser contexts or other Web runtime contexts (e.g. W3C widget contexts), they should be usable in Java environments as well. The specific binding to develop is an implementation choice.

6.3 Use of Web IDL Features in API Specifications

6.3.1 Modules

[WebIDL] defines a *module* as a definition that “serves as a container for grouping together related definitions.”

Web IDL modules are primarily used for organization of APIs into functionally related units, which are optionally identified through a namespace. W3C does not use modules for their API definitions. All W3C specs place interfaces and exceptions at the top level scope (i.e., not in a module). WAC uses modules only to organize their APIs.

Note that the status of modules is currently under discussion in W3C, and they may be removed from the Web IDL specification.

It is recommended that OMA Web runtime APIs be defined without use of modules.

6.3.2 Interfaces

[WebIDL] defines an interface as “a specification of a set of interface members, which are the *constants*, *attributes* and *operations* given by the InterfaceMembers part of the Interface. Objects implementing the interface will have members that correspond to each of the interface’s members.”

Appropriate use of interface inheritance is a key design criteria for APIs. Interfaces which build upon other interfaces are common in API definitions. This is commonly referred to as “extending” another interface, and is a best practice when applied carefully. The key criteria is that the inherited interface is public and stably defined, i.e. it is defined in an open standards specification that is stable, and is unlikely to be affected by late changes in the specification. At what point an interface is “stable” is often a judgement call; in W3C for example interfaces can be stable for years, although the related specifications have not reached the Candidate Recommendation stage, and the API may be supported by only a few of the major browsers. In such cases, whether to inherit the interface, or to define an equivalent set of interface functions is a key design decision.

It is recommended that OMA Web runtime APIs use inheritance only if the inherited interface is determined to be stably defined.

6.3.2.1 Constants

[WebIDL] defines a constant as “a declaration that matches the Const non-terminal, and is used to bind a constant value to a name. Constants can appear on interfaces and exceptions.”

It is recommended that OMA Web runtime APIs use constant declarations for all constant values provided by the API.

See Table 1 for an example of constant definition.

6.3.2.2 Attributes

[WebIDL] defines an attribute as “an interface member that matches an AttributeOrOperation non-terminal with an Attribute non-terminal in its derivation, and is used to declare that objects implementing the interface will have an attribute with the given identifier whose value can be retrieved and (in some cases) changed.”

No specific recommendations have been identified related to the use of attributes.

See Table 1 for an example of attribute definition.

6.3.2.3 Operations

[WebIDL] defines an operation as “an interface member that matches an AttributeOrOperation non-terminal with no Attribute non-terminal in its derivation. There are two kinds of operation: regular operations, which are those used to declare that objects implementing the interface will have a method with the given identifier, and special operations, which are used to declare special behavior on objects implementing the interface, such as object indexing and stringification.”

Note that the approach to supporting “operation overloading” (i.e. multiple operation definitions based upon the type of parameters passed) is currently under discussion in W3C. Until this discussion is resolved, it is recommended that OMA Web runtime API interfaces do not use operation overloading.

See Table 1 for an example of operation definition.

6.3.2.4 Special Operations

[WebIDL] defines a special operation as “a declaration of a certain kind of special behavior on objects implementing the interface on which the special operation declarations appear. Special operations are declared by using one or more special keywords in an operation declaration.”

Note that support for “unnamed getters/setters” may be removed from Web IDL. It is recommended that OMA Web runtime API interfaces do not define unnamed getters/setters.

See [WebIDL] for an example of special operation definition.

6.3.2.5 Overloading

As noted in 6.3.2.3, it is recommended that OMA Web runtime API interfaces do not use operation overloading.

6.3.3 Exceptions

Error and exception handling are key aspects requiring consistent approaches in OMA Web runtime APIs. See section 8 for a description of error and exception definition in Web IDL.

6.3.4 Typedefs

[WebIDL] defines a typedef as “a definition that matches the Typedef non-terminal, and is used to declare a new name for a type”. Typedefs are useful to allow the use of a short identifier instead of a long scoped name or sequence type.

The following typedefs MUST be supported by all OMA Web runtime API implementations.

Web IDL Specification

```
typedef sequence<DOMString> StringArray;  
typedef sequence<octet>      ByteArray;  
typedef sequence<short>     ShortArray;  
typedef sequence<short>     UnsignedShortArray;  
typedef sequence<long>      LongArray;  
typedef sequence<float>     FloatArray;  
typedef unsigned long long  Date;  
typedef sequence<Feature>   FeatureArray;
```

Table 2: Common Typedefs

6.3.5 Implements Statements

Web IDL's "implements" keyword is used to specify the object which instantiates the given interface as an object.

WAC APIs are defined as implemented by the *Window* object. W3C APIs are sometimes defined as implemented by the *Navigator* or *Window* objects. The reason for this variance is unclear; thus the recommended choice for OMA Web runtime APIs is TBD.

6.3.6 Types

6.3.7 Extended Attributes

7. Asynchronous Methods

As described by [WAC-API-Patterns], “ECMAScript methods can be either synchronous or asynchronous. Methods using the synchronous mode of operation do not return control to the caller until the operation is complete. Asynchronous methods return immediately and notify the caller at some point in the future of the results via callback method(s). The callback methods are specified as input parameters to the asynchronous method. Methods that may take a long time to be executed or that may be subject to security prompt are defined as asynchronous methods...”.

The key consideration for choice of operation mode is whether request processing can result in delayed response, or require user input. Generally it is preferred that application threads not be blocked API requests that must complete before the application can continue. For example, user prompts can be described as “modal” (blocking) or “non-modal” (non-blocking). In some cases, modal prompts may be preferred, and Web technologies support this need explicitly (e.g. the HTML5 showModalDialog API). However in most cases APIs that result in user prompts, e.g. for data input or confirmation of API permissions to grant to the application, should be designed in asynchronous mode, to avoid modal prompts.

Other types of interfaces that require asynchronous mode operation include interfaces that establish an event listener, e.g. as in the WRAPI Push API. Event listeners are expected to receive one or more events at a future time, and thus require the asynchronous interface mode.

7.1 Types of Asynchronous Methods

Two types of asynchronous methods are defined by [WAC-API-Patterns], “function-only success callback” and “interface success callback”. The type of method chosen for the API depends upon whether the API request will result in a single success or error callback (“function-only”), or whether in addition specific event interfaces are defined as well. For example, an interface can include specific “on event” handler definitions for events related to the interface’s purpose, e.g. a messaging interface that provides not only message sending success and error callbacks, but an event interface for message delivery or read receipt.

7.1.1 Function-Only Success Callback Methods

Parameter (in order of occurrence)	Optionality	Description
Success callback	Mandatory	Function to handle success callback
Error callback	Optional	Function to handle error callback
Additional parameters	Optional	Method-specific parameters

Table 3: Function-Only Success Callback Method Parameters

7.1.2 Interface Success Callback Methods

Parameter (in order of occurrence)	Optionality	Description
Success callback	Mandatory	Interface defining functions to handle specific callback events
Error callback	Optional	Function to handle error callback
Additional parameters	Optional	Method-specific parameters

Table 4: Interface Success Callback Method Parameters

7.2 Pending Operations

7.2.1 PendingOperation Interface

To promote consistency in the handling of asynchronous operations, the PendingOperation interface is defined for use by OMA Web runtime APIs.

OMAWeb runtime APIs MUST return a PendingOperation object upon the successful invocation of asynchronous operations that are cancellable. If a callback was executed prior to completion of the asynchronous operation method, OMAWeb runtime APIs MUST return null.

The result of an asynchronous method call is always a PendingOperation object, or null. Pending Operation objects provide a means to cancel the asynchronous operation, if possible. The return of null indicates that a callback was executed prior to completion of the asynchronous method, with the effect that the operation is completed (no longer pending).

Web IDL Specification

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

Table 5: PendingOperation Interface Web IDL Specification

Code example

Code example

```
var pendingOp = null;

// Define the success callback
function someOpSuccess() {
    alert("The someOp action was successful ");
    pendingOp = null;
}

// Define the error callback
function someOpError(error) {
    alert("The someOp action could not be completed: " + error.message);
    pendingOp = null;
}

// To be executed if, for instance, the user presses a cancel button in the user interface
function cancel() {
    if (pendingOp != null) {
        if (pendingOp.cancel()) {
            alert("The someOp action has been canceled");
        } else {
            alert("The someOp action cannot be canceled");
        }
    } else {
        alert("The someOp action cannot be canceled");
    }
}

pendingOp = omaapi.someApi.someOP(someOpSuccess, someOpError);
```

Table 6: PendingOperation code example

Methods

cancel

Call to cancel the underlying asynchronous operation.

Signature

Method Signature

```
boolean cancel();
```

Table 7: cancel() method signature

This call is always successful, i.e. the pending operation is either canceled or one of the callbacks is called.

Return value

true if the cancellation is successful. No callbacks will be called by the cancelled pending operation.

false if the cancellation did not succeed either because the pending operation finished already or because the cancellation cannot succeed due to technical limitations in the underlying implementation. Consequently, the pending operation completes, and depending on the success or failure the appropriate callbacks will be called after this method returns.

7.3 Success Callbacks

7.3.1 SuccessCallback Interface

To promote consistency in the handling of success callbacks, the SuccessCallback interface is defined for use by OMA Web runtime APIs.

For success callbacks that do not provide any callback parameters, OMAWeb runtime APIs MUST define the success callback using the SuccessCallback interface type.

For success callbacks that do provide callback parameters, OMAWeb runtime APIs MUST define corresponding success callback interfaces within the OMAWeb runtime API module.

Web IDL Specification

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

Table 8: SuccessCallback Interface Web IDL Specification**Methods****onsuccess**

Method invoked when the asynchronous call completes successfully.

Signature**Method Signature**

```
void onsuccess();
```

Table 9: onsuccess() method signature**Code example**

Code example

```
// Define the success callback
function someOpSuccess() {
    alert("The someOp action was successful ");
}

// Define the error callback
function someOpError(error) {
    alert("The someOp action could not be completed: " + error.message);
}

omaapi.someApi.someOP(someOpSuccess, someOpError);
```

Table 10: onsuccess() method code example

7.4 Error Callbacks

7.4.1 ErrorCallback Interface

To promote consistency in the handling of error callbacks, the ErrorCallback interface is defined for use by OMA Web runtime APIs.

For error callbacks that provide only an error as callback parameter, OMAWeb runtime APIs MUST define the error callback using the ErrorCallback interface type.

For error callbacks that do provide additional callback parameters, OMAWeb runtime APIs MUST define corresponding error callback interfaces within the OMAWeb runtime API module.

Web IDL Specification

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

Table 11: ErrorCallback Interface Web IDL Specification

Methods

onerror

Method invoked when the asynchronous call results in an error.

Signature

Method Signature

```
void onerror(in OmaAPIError error);
```

Table 12: onerror() method signature

Parameters

error

The error that is raised.

Code example

Code example

```
// Define the success callback
function someOpSuccess() {
    alert("The someOp action was successful ");
}

// Define the error callback
function someOpError(error) {
    alert("The someOp action could not be completed: " + error.message);
}

omaapi.someApi.someOP(someOpSuccess, someOpError);
```

Table 13: onerror() method code example

8. Error Handling

8.1 OmaAPIError Interface

To promote consistency in common error codes, the OmaAPIError interface provides a pre-defined set of error codes for use by OMA Web runtime APIs. The OmaAPIError interface is based upon the DOMError interface defined in [DOM3Core].

OMAWeb runtime APIs MUST use the OmaAPIError interface for these error codes.

OMA Web runtime APIs MAY define additional error codes through the use of the [Supplemental] keyword without the need to specify a new Error interface.

Web IDL Specification

```
[NoInterfaceObject] interface OmaAPIError {
  readonly attribute short code;
  readonly attribute DOMString message;
  const short UNKNOWN_ERR = 0;
  const short INDEX_SIZE_ERR = 1;
  const short DOMSTRING_SIZE_ERR = 2;
  const short HIERARCHY_REQUEST_ERR = 3;
  const short WRONG_DOCUMENT_ERR = 4;
  const short INVALID_CHARACTER_ERR = 5;
  const short NO_DATA_ALLOWED_ERR = 6;
  const short NO_MODIFICATION_ALLOWED_ERR = 7;
  const short NOT_FOUND_ERR = 8;
  const short NOT_SUPPORTED_ERR = 9;
  const short INUSE_ATTRIBUTE_ERR = 10;
  const short INVALID_STATE_ERR = 11;
  const short SYNTAX_ERR = 12;
  const short INVALID_MODIFICATION_ERR = 13;
  const short NAMESPACE_ERR = 14;
  const short INVALID_ACCESS_ERR = 15;
  const short VALIDATION_ERR = 16;
  const short TYPE_MISMATCH_ERR = 17;
  const short SECURITY_ERR = 18;
  const short NETWORK_ERR = 19;
  const short ABORT_ERR = 20;
  const short TIMEOUT_ERR = 21;
  const short INVALID_VALUES_ERR = 22;
};
```

Table 14: OmaAPIError Web IDL Specification

This interface will be used by the APIs in order to throw errors synchronously or return them in the error callbacks of asynchronous methods. The error codes defined in this interface are as defined by the DOM Level 3 specification, with some additions. APIs may extend the list of error codes with module-specific ones through the use of the [Supplemental] Web IDL keyword. API specific error numbers MUST start from 100 in order to avoid collision with future DOM Error codes.

Code example

Code example

```
omaapi.someapi.somemethod(
  function(dir) { // do something },
  function(e) {
    // handling error defined in Omaapi
    if (e.code == e.NOT_SUPPORTED_ERR) alert("not supported");
    // handling error defined in appropriate module (e.g. filesystem)
    if (e.code == e.IO_ERR) alert("i/o error");
  }, 'images', 'r'
);
```

Table 15: OmaAPIError code example

Constants

short UNKNOWN_ERR

Applicable for generic errors not related to the other codes.

short INDEX_SIZE_ERR

short DOMSTRING_SIZE_ERR

short HIERARCHY_REQUEST_ERR

short WRONG_DOCUMENT_ERR

short INVALID_CHARACTER_ERR

short NO_DATA_ALLOWED_ERR

short NO_MODIFICATION_ALLOWED_ERR

short NOT_FOUND_ERR

short NOT_SUPPORTED_ERR

Applicable if the implementation does not support the requested object type or operation.

short INUSE_ATTRIBUTE_ERR

short INVALID_STATE_ERR

short SYNTAX_ERR

short INVALID_MODIFICATION_ERR

short NAMESPACE_ERR

short INVALID_ACCESS_ERR

short VALIDATION_ERR

short TYPE_MISMATCH_ERR

Applicable if the object type is incompatible with the expected type of the parameter associated to the object.

short SECURITY_ERR

Applicable if an attempt is made to perform an operation or access data in a way that presents a security risk or a violation of the user agent security policy.

short NETWORK_ERR

Applicable if a network error occurs in synchronous requests.

short ABORT_ERR

short TIMEOUT_ERR

Applicable if the operation timed out and was not completed.

short INVALID_VALUES_ERR

Applicable if the content of an object does not contain valid values.

Attributes

readonly short code

16-bit error code. This attribute is readonly.

readonly DOMString message

Error message. Describes the details of the error encountered. This attribute is readonly.

Note: This attribute is not intended to be used directly in user interfaces as it is mainly intended to be useful for developers rather than end-users, and may not express the error in terms users will understand.

8.2 Interface Errors and Exceptions

The approach to notifying the application that an error has occurred depends upon the type of interface being accessed.

8.2.1 Synchronous Interface Errors

When an asynchronous method is executed, all error conditions are indicated by throwing an instance of `OmaAPIError` synchronously. These error conditions are documented in Web IDL through the `raises(OmaAPIError)` clause. The error codes that may be returned in the exception (`OmaAPIError`) instance are documented in the Exceptions section of the method documentation.

Web IDL Specification

```
Void getUrl(in DOMString url)
  raises(OmaAPIError);
```

Table 16: Example of Error Handling Definition in Web IDL for Synchronous Interfaces

The applicable error codes should be described in the specification text (outside the Web IDL) of the OMA Web runtime API, e.g.:

Exceptions

- `OmaAPIError`:
 - with error code `NOT_SUPPORTED_ERR` if the URI scheme of the URL parameter is not supported in the device.
 - With error code `NOT_FOUND_ERR` if the attempt to connect to the resource addressed by the URL resulted in an HTTP 404 NOT FOUND response.

Table 17: Example of Error Usage Documentation for Synchronous Interfaces

8.2.2 Asynchronous Interface Errors

When an asynchronous method is executed, the implementation checks that the arguments are of the valid type or can be coerced to the appropriate type. If any of the arguments passed has an unexpected type compared to the the declared type of that argument, the method will raise an `OmaAPIError` synchronously with code `TYPE_MISMATCH_ERR`. Any other error will be returned in the `ErrorCallback` that was passed as input argument to the asynchronous function. The documentation of the method will specify which codes and under which conditions the error will be returned in the `ErrorCallback`. If no error callback is passed, it is null or undefined, no additional notification is needed (i.e. the method fails silently).

Type conversions between the supplied arguments and the declared IDL argument type are performed in accordance with the ECMAScript Web IDL binding [WebIDL].

The following is a Web IDL excerpt declaring an asynchronous method.

Web IDL Specification

```

Void getUrl(in SuccessCallback successCallback,
            in ErrorCallback errorCallback,
            in DOMString url)
  raises(OmaAPIError);

```

Table 18: Example of Error Handling Definition in Web IDL for Asynchronous Interfaces

The raises (OmaAPIError) part is intended to specify that although the method is asynchronous, an OmaAPIError may be thrown if the arguments are not of the valid type (TYPE_MISMATCH_ERR). That is documented in the method exception list. The errors returned in the errorCallback are also specified in the method description.

getUrl

Attempt to retrieve the content at the URL using HTTP GET.

If any of the input parameters are not compatible with the expected type for that parameter, an OmaAPIError with code TYPE_MISMATCH_ERR MUST be synchronously thrown.

If the URI scheme of the URL parameter is not supported in the device, an OmaAPIError with code NOT_SUPPORTED_ERR MUST be synchronously thrown.

If the URL could not be accessed, the errorCallback is invoked. The following error codes may be returned in the error callback depending on the error conditions:

- NOT_FOUND_ERR: if the attempt to connect to the resource addressed by the URL resulted in an HTTP 404 NOT FOUND response.

Table 19: Example of Error Usage Documentation for Asynchronous Interfaces

9. Arguments

Except as defined below, User Agents MUST process method arguments as defined in [WebIDL].

Web IDL provides a mechanism to document that an argument is optional. In order to so, the keyword *optional* must be included just before the argument type.

All mandatory arguments MUST precede any optional arguments in OMA Web runtime API specifications. There cannot be a mandatory argument after an optional argument.

If an optional argument is not the appropriate type or cannot be coerced to it, the User Agent MUST throw an error with a `TYPE_MISMATCH_ERR` code.

10. Accessing APIs

10.1 Defining OMA Web Runtime APIs as part of the window global object

OMA Web runtime APIs must be defined as part of the window global object in order to be accessible to Web applications. The definition of the OmaapiObject and Omaapi interface below illustrate how these definitions should be included in OMA Web runtime API specifications.

10.1.1 OmaapiObject Interface

The OmaapiObject interface defines the omaapi object, through which interfaces common to OMA Web runtime APIs are defined as part of the window global object. The OmaapiObject interface MUST be supported by all OMA Web runtime API implementations.

Web IDL Specification

```
[NoInterfaceObject] interface OmaapiObject {
    readonly attribute Omaapi omaapi;
};
Window implements OmaapiObject;
```

Table 20: OmaapiObject Web IDL Specification

10.1.2 Omaapi Interface

The Omaapi interface defined below is the OMA Web runtime API root interface that is part of the ECMAScript global object as a property, as specified by the OmaapiObject interface. The Omaapi interface MUST be supported by all OMA Web runtime API implementations.

The Omaapi interface will be always available within the Window object in the ECMAScript hierarchy.

Web IDL Specification

```
[NoInterfaceObject] interface Omaapi {
    FeatureArray listAvailableFeatures();
    FeatureArray listActivatedFeatures();
};
```

Table 21: Omaapi interface Web IDL Specification

Methods

listAvailableFeatures

Returns a list with all the OMA Web runtime API features available in the Web Runtime (WRT).

Signature

Method Signature

```
FeatureArray listAvailableFeatures();
```

Table 22: listAvailableFeatures() method signature

Provides a list with all the features that are available in the WRT.

For the Browser Context, all implemented features MUST be included.

For the Widget Context:

- Features not requested in the widget configuration document (config.xml), including the attributes required and parameters in the feature instance MUST be null.
- For those features requested in the widget configuration document, the attributes required and parameters MUST contain the values provided in the widget configuration document.

Return value

Array with all the available features or null in case an error occurs.

Code example**Code example**

```
var features = omaapi.listAvailableFeatures();
for (var i=0; i<features.length; i++) {
    alert("The Feature " + features[i].uri + " is supported");
}
```

Table 23: listAvailableFeatures() code example**listActivatedFeatures**

Returns a list with all the OMA Web runtime API features that has been activated for the application in the WRT.

Signature**Method Signature**

```
FeatureArray listActivatedFeatures();
```

Table 24: listActivatedFeatures() method signature

Provides a list with all the features that are activated in the WRT.

For the Browser Context, all implemented features MUST be included.

For the Widget Context:

- This method MUST provide a list with all the features that have been activated for the widget in the WRT.
- For every feature, the required and params attributes MUST contain the values provided in the widget configuration document.

Return value

Array with all the activated features or null in case an error occurs.

Code example**Code example**

```
var features = Omaapi.listActivatedFeatures();
for (var i=0; i<features.length; i++) {
    alert("The Feature " + features[i].uri + " has been activated");
}
```

Table 25: listActivatedFeatures() code example

10.2 Accessing API Feature Information

10.2.1 Feature Interface

The Feature interface defined below enables access to information about API features. The Feature interface **MUST** be supported by all OMA Web runtime API implementations.

Web IDL Specification

```
[NoInterfaceObject] interface Feature {
  attribute DOMString uri;
  attribute boolean required;
  attribute ParamArray params;
};
```

Table 26: Feature interface Web IDL Specification

Attributes

DOMString uri

This is the unique identifier of the API feature.

boolean required

This attribute is set to true if the feature has been tagged as required in the widget configuration document (config.xml).

ParamArray params

This attribute contains an array with all the parameters included for that feature in the widget configuration document (config.xml).

10.2.2 Param Interface

The Param interface represents a parameter linked to a feature. The Param interface **MUST** be supported by all OMA Web runtime API implementations.

Web IDL Specification

```
[NoInterfaceObject] interface Param {
  attribute DOMString name;
  attribute DOMString value;
};
```

Table 27: Param interface Web IDL Specification

Attributes

DOMString name

Name of the parameter.

DOMString value

Value of the parameter.

11.Full Web IDL

Web IDL Specification

```

typedef sequence<DOMString> StringArray;
typedef sequence<octet>      ByteArray;
typedef sequence<short>     ShortArray;
typedef sequence<short>     UnsignedShortArray;
typedef sequence<long>      LongArray;
typedef sequence<float>     FloatArray;
typedef unsigned long long  Date;
typedef sequence<Feature>   FeatureArray;

[NoInterfaceObject] interface OmaapiObject {
    readonly attribute Omaapi omaapi;
};
Window implements OmaapiObject;

[NoInterfaceObject] interface Omaapi {
    FeatureArray listAvailableFeatures();
    FeatureArray listActivatedFeatures();
};

[NoInterfaceObject] interface OmaAPIError {
    readonly attribute short code;
    readonly attribute DOMString message;
    const short UNKNOWN_ERR = 0;
    const short INDEX_SIZE_ERR = 1;
    const short DOMSTRING_SIZE_ERR = 2;
    const short HIERARCHY_REQUEST_ERR = 3;
    const short WRONG_DOCUMENT_ERR = 4;
    const short INVALID_CHARACTER_ERR = 5;
    const short NO_DATA_ALLOWED_ERR = 6;
    const short NO_MODIFICATION_ALLOWED_ERR = 7;
    const short NOT_FOUND_ERR = 8;
    const short NOT_SUPPORTED_ERR = 9;
    const short INUSE_ATTRIBUTE_ERR = 10;
    const short INVALID_STATE_ERR = 11;
    const short SYNTAX_ERR = 12;
    const short INVALID_MODIFICATION_ERR = 13;
    const short NAMESPACE_ERR = 14;
    const short INVALID_ACCESS_ERR = 15;
    const short VALIDATION_ERR = 16;
    const short TYPE_MISMATCH_ERR = 17;
    const short SECURITY_ERR = 18;
    const short NETWORK_ERR = 19;
    const short ABORT_ERR = 20;
    const short TIMEOUT_ERR = 21;
    const short INVALID_VALUES_ERR = 22;
};

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

[Callback=FunctionOnly, NoInterfaceObject] interface ErrorCallback {
    void onerror(in OmaAPIError error);
};

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

[NoInterfaceObject] interface Feature {
    attribute DOMString uri;
    attribute boolean required;
    attribute ParamArray params;
};

```

```
[NoInterfaceObject] interface Param {  
  attribute DOMString name;  
  attribute DOMString value;  
};
```

Appendix A. Change History

(Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-WRAPI_Design_Patterns-V1_0-20140923-A	23 Sep 2014	Status changed to Approved by TP TP Ref # OMA-TP-2014-0216-INP_WRAPI_V1_0_ERP_for_final_Approval

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

B.1 SCR for User Agent

Item	Function	Reference	Requirement
WRAPI-PATTERN-C-001-M	Typedefs	6.3.4	
WRAPI-PATTERN-C-002-M	PendingOperation Interface	7.2.1	
WRAPI-PATTERN-C-003-M	SuccessCallback interface	7.3.1	
WRAPI-PATTERN-C-004-M	ErrorCallback interface	7.4.1	
WRAPI-PATTERN-C-005-M	OmaAPIError interface	8.1	
WRAPI-PATTERN-C-006-M	Arguments	9	
WRAPI-PATTERN-C-007-M	OmaapiObject interface	10.1.1	
WRAPI-PATTERN-C-008-M	Omaapi interface	10.1.2	
WRAPI-PATTERN-C-009-M	Feature interface	10.2.1	
WRAPI-PATTERN-C-010-M	Param interface	10.2.2	