

OMTP PUBLISHED



BONDI

**BONDI ARCHITECTURE & SECURITY
REQUIREMENTS**

VERSION:	Version 1.1
STATUS:	Approved Release
DATE OF LAST EDIT:	27 January 2010
OWNER	OMTP Limited

CONTENTS

1	INTRODUCTION.....	6
1.1	GENERAL	6
1.2	CONTEXT	6
1.3	RELATED DOCUMENTS	7
1.4	CONTENTS	7
1.5	CONVENTIONS.....	7
1.6	TERMINOLOGY.....	8
1.7	COMPLIANCE	8
2	ARCHITECTURE OVERVIEW.....	9
2.1	GENERAL STRUCTURE AND SCOPE	9
2.1.1	<i>Widget.....</i>	<i>9</i>
2.1.2	<i>Website.....</i>	<i>10</i>
2.1.3	<i>Web Engine.....</i>	<i>10</i>
2.1.4	<i>Browser.....</i>	<i>10</i>
2.1.5	<i>Widget User Agent.....</i>	<i>10</i>
2.1.6	<i>Web Runtime</i>	<i>11</i>
2.1.7	<i>JavaScript API Access.....</i>	<i>11</i>
2.2	KEY ARCHITECTURE AND SECURITY FEATURES	12
2.2.1	<i>Independence of Delivery Model.....</i>	<i>12</i>
2.2.2	<i>Separation Of Security Framework From Policy</i>	<i>12</i>
2.2.3	<i>Independence of JavaScript API Definition</i>	<i>12</i>
2.2.4	<i>Rich Security Framework With Configurable Access Control Policy</i> <i>13</i>	
2.2.5	<i>Defined Security Policy Management Capability.....</i>	<i>13</i>
2.2.6	<i>Access to non-BONDI APIs and plugins</i>	<i>13</i>
3	WEB APPLICATION DELIVERY	15

3.1	INTRODUCTION	15
3.2	DELIVERY MODEL AND LIFECYCLE	15
3.2.1	<i>Scope</i>	15
3.2.2	<i>Requirements Summary and Rationale</i>	15
3.2.3	<i>Widget Resource Delivery and Lifecycle Requirements</i>	16
3.2.4	<i>Website Delivery and Lifecycle requirements</i>	19
3.3	WIDGET PACKAGING	19
3.3.1	<i>Scope</i>	19
3.3.2	<i>Requirements Summary and Rationale</i>	20
3.3.3	<i>Widget Resource Packaging Requirements</i>	20
3.3.1	<i>BONDI extensions to the widget configuration document</i>	21
3.4	WIDGET SIGNING REQUIREMENTS	24
3.4.1	<i>Scope</i>	24
3.4.2	<i>Requirements Summary and Rationale</i>	24
3.4.3	<i>Signing Requirements</i>	24
4	BONDI FEATURE ACCESS	30
4.1	SCOPE	30
4.2	REQUIREMENTS SUMMARY AND RATIONALE	30
4.3	EXPRESSION OF DEPENDENCIES	32
4.3.1	<i>Static expression of dependencies</i>	32
4.3.2	<i>Programmatic expression of dependencies</i>	38
5	SECURITY FRAMEWORK FOR FEATURE ACCESS	41
5.1	INTRODUCTION	41
5.2	FRAMEWORK OVERVIEW	41
5.3	LAYER MODEL	42
5.4	LOGICAL MODEL	44
5.5	ACCESS CONTROL POLICY STRUCTURE	46

5.6	RULE PROCESSING	48
5.7	REQUIREMENTS SUMMARY AND RATIONALE	49
5.8	REQUIREMENTS.....	49
6	SECURITY POLICY MANAGEMENT.....	51
6.1	INTRODUCTION	51
6.2	GENERAL SECURITY POLICY MANAGEMENT.....	51
6.2.1	<i>Scope.....</i>	<i>51</i>
6.2.2	<i>Requirements Summary and Rationale</i>	<i>52</i>
6.2.3	<i>Signed Security Policy Document processing.....</i>	<i>52</i>
6.2.4	<i>Security Policy Provisioning Requirements.....</i>	<i>56</i>
6.2.5	<i>Remote Security Policy Management Requirements.....</i>	<i>60</i>
6.2.6	<i>User Security Settings Requirements</i>	<i>63</i>
6.3	INSTALLATION AND MANAGEMENT OF EXTENSION APIs.....	64
6.3.1	<i>Scope.....</i>	<i>64</i>
6.3.2	<i>Requirements Summary and Rationale</i>	<i>64</i>
6.3.3	<i>Requirements.....</i>	<i>66</i>
7	DEFINITION OF TERMS	67
8	ABBREVIATIONS.....	70
9	REFERENCED DOCUMENTS.....	72

The information contained in this document represents the current view held by OMTP Ltd. on the issues discussed as of the date of publication.

This document is provided “as is” with no warranties whatsoever including any warranty of merchantability, non-infringement, or fitness for any particular purpose. All liability (including liability for infringement of any property rights) relating to the use of information in this document is disclaimed. No license, express or implied, to any intellectual property rights are granted herein.

This document is distributed for informational purposes only and is subject to change without notice. Readers should not design products based solely on this document.

© 2009-10 OMTP Ltd. All rights reserved. OMTP and OMTP BONDI are registered trademarks of OMTP Ltd.

1 INTRODUCTION

This section is non-normative.

1.1 GENERAL

This document has been prepared by OMTP as part of the BONDI initiative. It contains a statement of the scope and requirements for the Architecture and Security aspects of BONDI-compliant systems. It is a companion to the other main BONDI output relating to specific interfaces.

1.2 CONTEXT

The BONDI initiative has been conducted against a backdrop of rapid and diverse technology and product development in mobile and web technology. In particular, there are three parallel trends that are shaping the landscape:

- the expanding power of web technology as a means of service delivery, building on increasing richness of interaction, functionality and presentation in the browser, and the associated expansion of supporting web services technologies;
- the creation and proliferation of desktop widget frameworks based on web technology;
- the migration of “desktop-class” browser capability to mobile devices, enabling the delivery of even greater convergence between mainstream and mobile web experiences and services.

In addition to improved browsers, a new class of “Web Runtime” frameworks is emerging on mobile devices, which support the creation of “widgets” – installable applications authored in web formats. At present there is fragmentation both in feature sets and content (metadata and packaging formats), as well as differences in provisioning and deployment models and security frameworks. Beyond this first generation of web runtimes, there is clear potential to extend the functional scope of these frameworks, in multiple directions, including blurring of the distinction between conventional websites, web applications delivered through the browser, and web runtime/widget systems. Service providers, operators and manufacturers are all now seeking to capitalise on these new systems.

The OMTP has established the BONDI initiative which aims to ensure that this can take place whilst providing both a safe and consistent experience for users of the resulting services. This is being progressed by the formulation of common requirements, technical specifications and creation of Reference Implementations. It is intended that these are used within its member organisations to promote consistency of approach within the broader industry to accelerate the development of open technical standards, and can also seed adoption within content developer and service provider communities.

The Architecture and Security aspects of the BONDI recommendations, set out in this document, are a key element of ensuring safety and consistency of technical approach.

1.3 RELATED DOCUMENTS

This document is part of a number of Public BONDI Specifications that are available at:

<http://bondi.omtp.org/1.1/>

1.4 CONTENTS

This document is organised into the following sections.

Section 2 contains an overview of the BONDI Architecture and Security work, identifying the scope and highlighting key issues.

Sections 3-6 set out the key decisions and the BONDI requirements in three broad areas:

- Application Delivery (Section 3): this covers all aspects of the deployment and provisioning of content, including requirements for discovery and packaging.
- Feature Access (Section 4): this addresses the architecture for enabling access to JavaScript APIs and other Features defined by BONDI.
- Security Framework for Feature Access (Section 5): this sets out the proposed security framework in detail.
- Security Policy Management (Section 6): this discusses issues relating to the management of the functionality described in the preceding sections.

Detailed supporting material is provided in a series of Appendices which, together, are distributed in a separate volume.

1.5 CONVENTIONS

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative sections of this document are to be interpreted as described in RFC2119 [1].

The requirements within this document are uniquely identified using the following format:

AS-####, where:

- AS is the acronym used to identify the subject of this OMTP public working draft document (ie BONDI Architecture & Security)
- #### is a 4 digit number that identifies the requirement (e.g. 0020) and which is to be unique within the document.

1.6 TERMINOLOGY

The terminology adopted within these BONDI specifications is set out in Section 7. Throughout the document, defined terms are capitalised where used.

1.7 COMPLIANCE

The BONDI compliance specification [2] defines the process by which an implementation may claim conformance with this specification.

The requirements in this document are classified according to a series of compliance classes defined in [2]; multiple compliance classes exist in order that implementations can implement subsets of these Architecture and Security requirements according to the type of product in question or field of use.

2 ARCHITECTURE OVERVIEW

This section is non-normative.

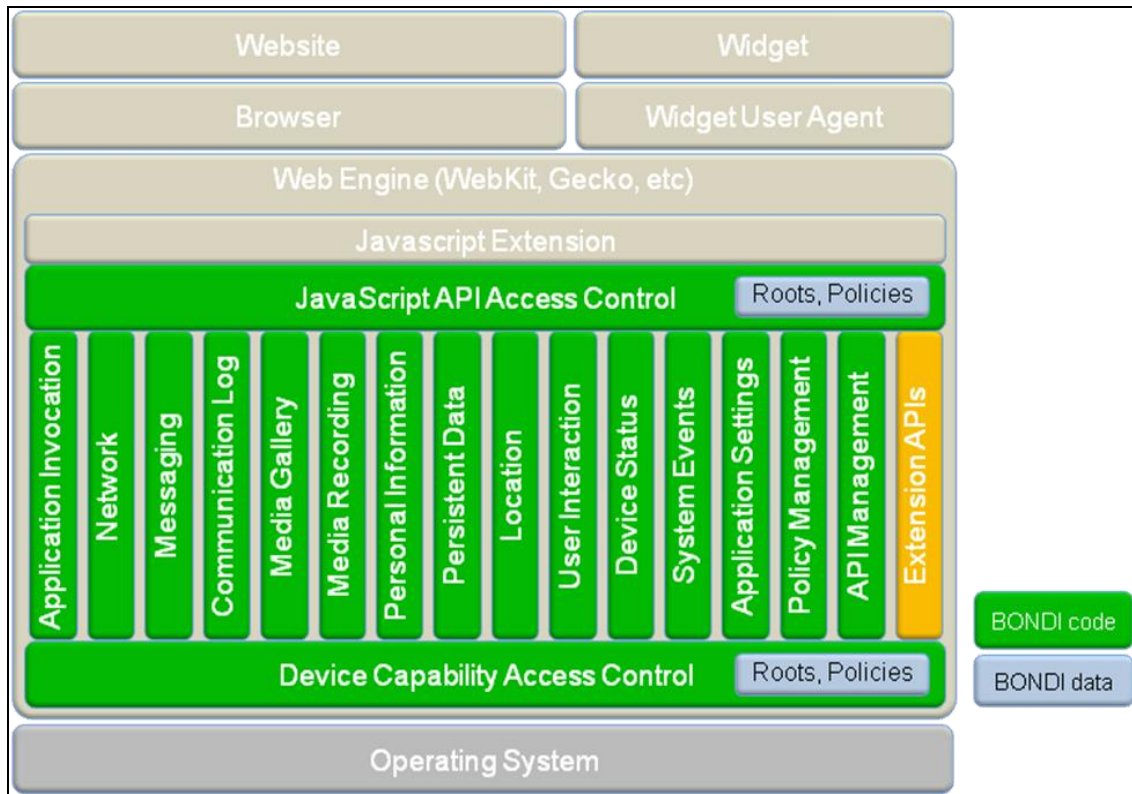


Figure 1: BONDI Architecture Overview

2.1 GENERAL STRUCTURE AND SCOPE

BONDI does not presuppose any specific architecture in the construction of a Web Runtime environment. However, it is useful to understand the typical construction of Browser and Widget User Agent to be able to place the various architecture requirements in context.

2.1.1 WIDGET

As described in Widgets 1.0: The Widget Landscape [3], a Widget is an interactive application for displaying and/or updating local data or data on the Web, packaged in a way to allow a single download and installation on a user's machine or mobile device. A Widget may run as a standalone application (meaning it can run outside of a web Browser) hosted in the Widget User Agent (see below). Prior to instantiation, a Widget exists as a Widget Resource. Widgets can be used to access and interact with any type of content.

2.1.2 **WEBSITE**

A Website is a remotely hosted collection of resources, authored in web formats (including HTML, SVG, JavaScript, CSS and various media formats) and served by a web server so as to be viewable in a Browser. Websites may contain relatively static data, or be highly interactive.

Web Application is the term used generically to refer to an application delivered using web technology, whether as a Website or a Widget.

2.1.3 **WEB ENGINE**

The core of a web Browser and of a Widget User Agent is the Web Engine, responsible for handling and rendering the markup (HTML/SVG), CSS and JavaScript content. This engine is considered to include components for parsing, laying out and rendering markup, including the application of CSS. It is also considered to include the JavaScript execution engine, and the client-side JavaScript binding to the HTML or SVG DOM. The Web Engine includes, or includes hooks to, components for handling and rendering media types embedded in, or referenced by, the web content (e.g. images, audio, multimedia). Commercial deployments exist where this Web Engine is based on an open-source codebase as well as on proprietary engines.

2.1.4 **BROWSER**

The Browser is the user-facing application that provides the ability to navigate Websites. The Browser application uses the Web Engine to handle Website content. The Browser application typically provides user functionality to manage bookmarks or favourites, use the browsing history and adjust Browser and Web Engine settings. The degree to which the actual underlying functionality is delegated to the Web Engine varies between frameworks. Browsers are typically used to access and interact with Web content (HTML family markup-language based content) and related resources (images and other media types).

2.1.5 **WIDGET USER AGENT**

The Widget User Agent is the collection of all components, over and above the Web Engine, needed to support installed Widgets. The functionality available varies between existing offerings but, broadly, the Widget User Agent is expected to be responsible for installation and de-installation of Widgets, and to provide the user-facing functionality for instantiation and configuration of those Widgets. In some systems the Widget User Agent also includes an intrinsic discovery and/or subscription mechanism with associated user controls. When Widgets are instantiated, the Widget User Agent instances the Web Engine in order to render the Widget content. A variety of application models are supported in existing Widget User Agents, including the ability to execute Widgets in “full-screen” mode (possibly with or without

associated furniture, title and softkeys) as well as the ability to run “phone-top” Widgets in the style of widgets becoming widespread on the desktop.

The formats for Widget Resources, including Widget metadata, and management of the Widget lifecycle, are within BONDI scope. Mechanisms for discovery and provisioning or download of Widget Resources are outside BONDI scope at this release.

2.1.6 **WEB RUNTIME**

Web Runtime is the term used generically and collectively to refer to the components that are provided to execute Web Applications (e.g. including either a Widget User Agent, or Browser, or both).

2.1.7 **JAVASCRIPT API ACCESS**

A key feature of Web Runtimes is the provision of APIs, supplementary to the standardised client-side DOM APIs (termed JavaScript APIs in this document) that provide access to services or features of the underlying platform itself, such as the file system, camera, or geolocation functionality. In principle, these JavaScript APIs could be accessible to any content being rendered by the Web Engine, including Widgets executing under control of the Widget User Agent but also including Websites being rendered under control of the Browser.

JavaScript API access is enabled by a range of different mechanisms in existing products and BONDI is not prescriptive about the precise mechanism used. However, BONDI does define requirements governing Device Capability access by JavaScript APIs to promote interoperability and security of implementations. Separately, BONDI includes definitions of JavaScript APIs in a number of specific areas of particular relevance to mobile devices (see BONDI Interfaces [4]).

For the purposes of this document, features enabling JavaScript API access are considered to fall into two areas:

- **JavaScript extension:** the mechanisms whereby JavaScript code executing within the Web engine is bound to, and therefore able to invoke, JavaScript APIs;
- **Access Control:** the system that enforces a Security Policy, responsible for determining whether, and under what circumstances, a Web Application is allowed to use a specific JavaScript API or associated underlying Device Capability.

There is also management and configuration functionality, and associated operational and procedural requirements, in each of these areas.

2.2 KEY ARCHITECTURE AND SECURITY FEATURES

The normative architecture and security requirements are set out in sections 3-6. However, the key features of the architecture are summarised below.

2.2.1 INDEPENDENCE OF DELIVERY MODEL

As stated above, BONDI does not make specific assumptions about how a Web Runtime is structured. Furthermore, a key principle is that the general BONDI framework is able to support multiple deployment models and is not limited to any specific mechanism by which web content is delivered to the device. BONDI requirements apply equally to content in a self-contained Widget executing in a Widget User Agent as to Website content loaded from a web server executing in a Browser. In addition, other delivery models are anticipated which employ locally stored and remotely accessed content in combination.

2.2.2 SEPARATION OF SECURITY FRAMEWORK FROM POLICY

BONDI defines a general framework in which security policies can be modelled, and specific policies can be encoded. BONDI also states minimum requirements for the supporting security mechanisms. However, these requirements are independent of any particular security policy, and BONDI is not prescriptive about the specific rules, constraints or defaults that apply to specific JavaScript APIs. BONDI describes mechanisms that support the structuring of security policies into a hierarchy of separately defined and managed elements with defined combining rules. However, it is not prescriptive about who should be the management authority of any particular aspect of security policy.

Although framework and policy are clearly separated at the level of the technical framework, BONDI clearly acknowledges the potential for fragmentation in both ecosystem and user experience from the development of inconsistent policies within that framework. Except as necessary to comply with regulatory or market requirements, developers should be able to count on a consistent set of default security policies. Therefore, in a later phase, BONDI may define a recommended default policy set. The established principles and experience of the deployment of the existing OMTP Application Security Framework [5]) will be relevant in this work.

2.2.3 INDEPENDENCE OF JAVASCRIPT API DEFINITION

BONDI includes within its scope the definition of a number of specific JavaScript APIs in areas of relevance to the mobile content developer. However, the OMTP also recognises the importance of supporting JavaScript APIs defined independently from BONDI. Therefore the framework includes explicit provision for JavaScript APIs being defined independently, defines mechanisms that support extensibility by allowing Web Applications to indicate explicitly which interfaces or capabilities they depend on, and requires

that Security Policies be supported for Extension APIs. The system allows for implementations in which new JavaScript APIs may be provisioned dynamically to a device; however, the supporting mechanisms for this dynamic extension are not defined by BONDI. A corresponding mechanism is envisaged that permits the dynamic configuration of supplementary security policies to support the use of dynamically added interfaces.

2.2.4 RICH SECURITY FRAMEWORK WITH CONFIGURABLE ACCESS CONTROL POLICY

BONDI proposes a very general security framework that unifies the modelling, representation and enforcement of security policies governing access to JavaScript APIs across all delivery methods. The framework allows the expression of many different forms of security policy, including policies based on Widget Resource signatures similar to those used for other mobile application formats, blacklists and/or whitelists of Widgets, authors, or Websites, and many other combinations.

A formal model for the structure and meaning of security policies is defined, together with a compact XML-based interchange format. The model identifies the various identity types, resources, attributes and conditions that can be expressed and which must be honoured by an implementation.

2.2.5 DEFINED SECURITY POLICY MANAGEMENT CAPABILITY

The fine-grained access control functionality described above brings a configuration burden; a Security Policy configuration must be created and maintained. This in turn, could be the source of usability problems or security vulnerabilities if there is no effective way of managing the Security Policy configuration.

Although BONDI does not attempt to specify every aspect of how Security Policy is managed on a device, it does establish a minimum common baseline for Security Policy management capability. This has the dual purpose of ensuring that Web Runtimes are manageable in an operational setting using standard means (such as by existing Device Management infrastructure) and ensuring that the associated configuration data (i.e. the Security Policies, or parts of them) are interoperable between consuming devices.

2.2.6 ACCESS TO NON-BONDI APIS AND PLUGINS

At this release, BONDI explicitly does not include any requirements governing the presence of, or access control for, proprietary JavaScript APIs or plugins that might be available in a Web Runtime outside the BONDI framework. The existence of such APIs or plugins may have security implications that undermine the protection provided by the BONDI security framework. Therefore, adopters of BONDI should take this into account when formulating requirements for the product as a whole.

Note that, as a result of the extensibility inherent in the BONDI framework (see 2.2.3 above), it is possible to support proprietary APIs within the BONDI framework, mediated by BONDI Access Control, and therefore maintain compatibility with associated content.

3 WEB APPLICATION DELIVERY

3.1 INTRODUCTION

This section is non-normative.

This section addresses the BONDI requirements that relate to Web Application delivery. The requirements are organised under the headings of:

- Delivery model and lifecycle;
- Widget packaging;
- Widget signing.

3.2 DELIVERY MODEL AND LIFECYCLE

3.2.1 SCOPE

This section is non-normative.

Requirements in this section cover:

- the delivery models required to be supported;
- Widget lifecycle support requirements.

3.2.2 REQUIREMENTS SUMMARY AND RATIONALE

This section is non-normative.

At present BONDI defines a number of delivery models, and the lifecycle requirements for each, via a series of use cases. The use cases are defined in “BONDI Architecture & Security, Application Life Cycle Events Use Cases” [6]. There are no requirements at present that mandate support for any specific delivery model.

The delivery models defined are:

- Web Applications delivered as locally-installable Widget Resources. In this case, the “widget packaging” and “widget signing” requirements apply. Implementations supporting this delivery model must support a series of use cases, covering:
 - installation, update and de-installation of Widgets, including both user-initiated and remotely-initiated operations;
 - execution of Widget Resources without prior installation;
- Web Applications delivered as Websites.

3.2.3 WIDGET RESOURCE DELIVERY AND LIFECYCLE REQUIREMENTS

This section is normative.

Support for Widget Resources by the Web Runtime is required by the Widget User Agent Conformance Class. In the case that Widget Resources are supported, the following requirements apply. Note that BONDI does not require the use of specific protocols, formats or other mechanisms in order to satisfy these requirements.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0010	Support SHALL be provided for user-initiated Widget Resource installation from a web server	This requirement entails support for lifecycle use case 2.1.1 [6]. The use case describes how a user transfers a Widget Resource from a web server to his/her terminal's local storage and persistently installs it in the terminal.
AS-0020	Support SHALL be provided for remotely-initiated Widget Resource installation from a web server	This requirement entails support for lifecycle use case 2.1.2 [6]. The use case describes how a remote service initiates the transfer of a Widget Resource from a web server to a terminal and persistent Installation in the terminal.
AS-0030	Support SHALL be provided for user-initiated Widget Resource installation from local storage	This requirement entails support for lifecycle use case 2.1.3 [6]. The use case describes how a user installs a Widget Resource from a terminal's local storage and persistently stores it in the terminal.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0040	Support SHALL be provided for user-configuration of an installed Widget to adjust auto-start settings.	This requirement entails support for lifecycle use case 2.2.1 [6]. The use case describes the auto-start settings that MAY be managed for a Widget Resource. At a minimum, the user MUST be provided with an option to disable auto-start for any Widget.
AS-0050	Support SHALL be provided for remotely-initiated Widget Resource de-installation	This requirement entails support for lifecycle use case 2.3.1 [6]. The use case describes how a remote service initiates the de-installation of a Widget Resource from the terminal.
AS-0060	Support SHALL be provided for user-initiated Widget Resource de-installation	This requirement entails support for lifecycle use case 2.3.2 [6]. The use case describes how a user de-installs a Widget Resource.
AS-0070	Support SHALL be provided for auto-start of installed Widgets	This requirement entails support for lifecycle use case 2.4.6.1 [6]. The use case describes how installed Widget Resources are instantiated during the terminal boot-up procedure terminal.
AS-0080	Support for programmatic instantiation of installed Widgets SHALL be provided using the BONDI Application Launcher API and SHOULD be provided from other execution environments that support programmatic application launch	This requirement entails support for lifecycle use case 2.4.6.2 [6]. The use case describes how an installed Widget Resource is instantiated by an Application that is executed in the terminal.
AS-0090	Support SHALL be provided for user-initiated instantiation of installed Widgets	This requirement entails support for lifecycle use case 2.4.6.3 [6]. The use case describes how a user manually instantiates an installed Widget Resource.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0100	Support SHALL be provided for processing of a non-installed Widget Resource from a web server	This requirement entails support for lifecycle use case 2.4.7.1 [6]. The use case describes how a Widget Resource is loaded from a web server and is processed without prior Installation.
AS-0110	Support SHALL be provided for non-installed Widget Processing from a local storage	This requirement entails support for lifecycle use case 2.4.7.2 [6]. The use case describes how a Widget Resource is loaded from a local storage and processed without prior Installation.
AS-0115	Support SHALL be provided for user discovery of all running Widgets.	This requirement applies to all running Widgets, including those that are running in background or hidden mode.
AS-0120	Support SHALL be provided for user-initiated termination of any running Widget	This requirement entails support for lifecycle use case 2.5 [6]. The use case describes how the Processing of a Widget Resource is terminated.
AS-0130	Support SHALL be provided for automatic Widget Resource update	This requirement entails support for lifecycle use case 2.6.1.1 [6]. The use case describes how an installed Widget Resource is automatically updated.
AS-0140	Support SHALL be provided for remotely initiated Widget Resource update	This requirement entails support for lifecycle use case 2.6.1.2 [6]. The use case describes how the version update of an installed Widget Resource is initiated by a remote service.
AS-0150	Support SHALL be provided for user-initiated Widget Resource update	This requirement entails support for lifecycle use case 2.6.1.3 [6]. The use case describes how an installed Widget Resource is manually updated.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0160	The system SHALL by default inhibit automated update of widget resources over PLMN when in roaming mode	This requirement entails support for lifecycle use case 2.6.3 [6]. The use case describes how a Widget Resource is not automatically updated when the terminal is in the Roaming Mode.

3.2.4 WEBSITE DELIVERY AND LIFECYCLE REQUIREMENTS

This section is normative.

Support for Websites is required for the Browser Conformance Class. In the case that Websites are supported then the following requirement applies.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0170	Support SHALL be provided for processing of Websites	This requirement entails support for lifecycle use case 2.1.1 [6]. The use case describes how a Web User Agent retrieves a Web Page from a Web Page Provisioning Server and renders it.

3.3 WIDGET PACKAGING

3.3.1 SCOPE

This section is non-normative.

Requirements in this section cover formats and processing requirements relating to:

- the container (i.e. package) for Widget Resources;
- metadata for Widget Resources, either contained within a Widget Resource package or otherwise provided, that may be used by the Web Runtime during the installation or instantiation of a Widget.

The requirements identified aim to:

- ensure that Widget Resources are amenable to provisioning using standard procedures and protocols for mobile content delivery;
- support requirements for API extensibility (addressed in Section 4).

3.3.2 REQUIREMENTS SUMMARY AND RATIONALE

This section is non-normative.

BONDI has chosen to adopt the W3C Widgets 1.0: Packaging and Configuration specification [7]. The requirements in this section reference and supplement the current W3C specification to ensure BONDI’s objectives are fully met. It should be noted that it is the intention of OMTP to adopt the final published widget specifications upon completion by W3C.

The W3C specification [7] defines the format and content of an XML configuration file associated with a Widget Resource, the Widget Configuration Document. BONDI specifies certain information that can be included in a Widget Configuration Document to supplement that defined by the existing W3C specifications in the following areas:

- parameters indicating the location of remote network resources required by the Widget.

BONDI specifies the encoding of this information as a BONDI-specific attribute, in a BONDI-specific namespace, to be included in the Widget Configuration Document.

3.3.3 WIDGET RESOURCE PACKAGING REQUIREMENTS

This section is normative.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0180	<p>The Web Runtime SHALL support the processing of Widget Resources packaged according to Widgets 1.0: Packaging and Configuration [7].</p> <p>Widgets not packaged according to Widgets 1.0: Packaging and Configuration [7], with the additional constraints described in this document, SHALL not be installed.</p>	<p>The Widgets 1.0: Packaging and Configuration [7] defines packaging format and processing model for a Widget Resource.</p>

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0250	<p>The Web Runtime SHALL support the capability to display a representation to the user of the metadata contained in a Widget Configuration Document. This capability SHALL also cover the BONDI elements specified in this document.</p> <p>The Web Runtime SHALL support the capability to retrieve this information from the Widget Configuration Document of a Widget Resource and display it to the user before the Widget Resource is installed.</p>	<p>Presenting the user with a representation of the information contained in a widget's configuration document MAY help them to make a more informed decision about whether or not they want to download and install the Widget Resource.</p>
AS-0260	<p>If a Widget Resource is being provisioned using a provisioning mechanism that supports download of metadata separately from the Widget Resource, the Web Runtime SHOULD support the capability to retrieve this information from the provisioning metadata and display it to the user for confirmation before performing the download of the Widget Resource.</p>	<p>Mobile provisioning protocols such as OMA DL [9] support this capability.</p>

3.3.1 BONDI EXTENSIONS TO THE WIDGET CONFIGURATION DOCUMENT

This section is normative.

BONDI identifies a set of additional parameters relating to the operation of a Widget that are not currently representable by the elements in the Widget Configuration Document defined in Widgets 1.0: Packaging and Configuration [7].

BONDI has therefore defined additional elements and attributes to allow this information to be included by the Widget author as part of the Widget Configuration Document. It should be noted that it is the intention of OMTP to

deprecate these BONDI-defined elements at such time that same information becomes representable in a Widget Configuration Document as specified by a Full Recommendation by W3C.

The BONDI namespace

The BONDI namespace is:

`http://bondi.omtp.org/ns/widgets`

BONDI-specified elements and/or attributes included in a Widget Configuration Document **MUST** be assigned to the BONDI namespace.

Network access by Widgets

Network access by a Widget to a resource on the network, includes both programmatic access (for example using an API such as XMLHttpRequest) and inclusion of document elements that refer to network resources (such as iframe, script or img).

Network access is disallowed by default. Such access only becomes possible if explicitly requested by the Widget, and if access is granted by the Web Runtime based on the configured Security Policy.

Access is requested to network resources via the BONDI-defined `<network-access>` element and associated BONDI-defined `uri` attribute.

A `<widget>` element **MAY** contain multiple `<network-access>` elements, each requesting access to a set of network resources.

The following attributes are defined for `<network-access>`:

- Attribute: `uri`
 - Use: Required
 - Type: URI
 - Constraints: the specified URI **SHALL** be a valid URI including a host component and **SHALL** omit fragment and user info components.
- Attribute: `subdomains`
 - Use: Optional
 - Type: Boolean
 - Default value: false

Each `<network-access>` element defines a set of *target* IRIs. The set of target IRIs defined is the set of all given IRIs *i* satisfying all of the following constraints:

- the scheme component of *i* equals the scheme component of the `uri` attribute;
- in the case that that the `subdomains` attribute is absent or has value `false`, the host component of *i* *matches* the host component of the `uri` attribute;
- in the case that that the `subdomains` attribute is present and has value `true`, the host component of *i* *matches* the host component of the `uri` attribute, or *matches* some subdomain of, the host component of the `uri` attribute;
- in the case that both *i* and the `uri` attribute have a port component, they are equal;
- in the case that exactly one of *i* and the `uri` attribute has a port component, it is equal to the default port for the scheme component of *i*;
- the concatenation of path and query components of *i* is equal to, or begins with, the concatenation of the path and query components of the `uri` attribute.

In the above, the matching of host components is defined as:

- each host component is first converted to an ASCII string using the ToASCII algorithm of [RFC3490];
- for all schemes other than `http` and `https`, the resulting strings are identical;
- for `http` and `https`, the resulting strings are case-insensitively equal.

The set of *target IRIs* for the Widget is the union of the sets of target IRIs defined by each of the `<network-access>` elements.

The Web Runtime SHALL deny each network access attempt at runtime to a resource whose IRI does not belong to the set of target IRIs.

A Web Runtime SHALL verify that each network access attempt at runtime is permitted by evaluating an access control Query, with a `device-cap` resource attribute corresponding to the IRI of the resource to which access is being requested, against the configured Security Policy in accordance with the

evaluation rules specified in Appendix B. The evaluation SHALL be performed at a time such that the result of the evaluation is determined. If the evaluated result of the Query is Deny, the network access SHALL be denied.

Denial of a programmatic network access attempt SHALL generate a non-fatal run-time exception. Denial of a network access attempt by a referring document element SHALL result in a non-fatal failure to load the referred resource, processed by the Web Runtime in the same manner as if the resource had been unavailable.

3.4 WIDGET SIGNING REQUIREMENTS

3.4.1 SCOPE

This section is non-normative.

This section covers requirements related to signing a Widget Resource.

3.4.2 REQUIREMENTS SUMMARY AND RATIONALE

This section is non-normative.

The W3C Widgets 1.0 Digital Signature specification [8] defines a signature format and processing model for generating and verifying a digital signature over a Widget Resource.

The generated signature value allows the Web Runtime to verify the integrity and authenticity of every file in the Widget Resource, excluding any other signature files that may be present.

The requirements in this section reference and supplement the W3C specifications to ensure BONDI's objectives are fully met.

3.4.3 SIGNING REQUIREMENTS

This section is normative.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0270	The Web Runtime SHALL support the processing of Widget Resources that have been signed according to the Widgets 1.0: Digital Signatures specification [8], with the additional constraints and processing described in this document.	The Widgets 1.0: Digital Signatures specification [8] defines a signature format and a processing model for generating and verifying a digital signature over a Widget Resource.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0280	The Web Runtime SHALL only treat a Widget Resource as having a valid author signature (and present associated author signature Subject Attributes in the security model) when it has been able to successfully verify the author signature following the processing rules defined in Widgets 1.0: Digital Signatures specification [8], with the additional constraints and processing described in this document.	Widget Resources MUST be signed according to the Widgets 1.0: Digital Signatures specification [8].
AS-0285	The Web Runtime SHALL only treat a Widget Resource as having a valid distributor signature (and present associated distributor signature Subject Attributes in the security model) when it has been able to successfully verify a distributor signature following the processing rules defined in Widgets 1.0: Digital Signatures specification [8], with the additional constraints and processing described in this document.	Widget Resources MUST be signed according to the Widgets 1.0: Digital Signatures specification [8].
AS-0286	If a Widget Resource has multiple distributor signatures that are successfully verified, the Web Runtime SHALL use the first distributor signature that is successfully verified (based on the order of processing of distributor signatures defined in Widgets 1.0: Digital Signatures specification [8] (and present the associated distributor signature Subject Attributes in the security model).	In the case of there being multiple verifiable distributor signatures, the lowest numbered verifiable signature is used and all other distributor signatures are discarded.

REQ. ID	FUNCTIONALITY	DESCRIPTION
<p>AS-0290</p>	<p>If a Widget Resource is being provisioned using a provisioning mechanism that supports prior download of metadata (the “metadata provisioning step”) separately from download of the Widget Resource (the “Widget Resource provisioning step”), the Web Runtime SHOULD support the capability to retrieve this information provisioned metadata and complete the signature validation processing of a digital signature document, as defined in Widgets 1.0: Digital Signatures specification [8] before performing the Widget Resource provisioning step.</p> <p>If the Web Runtime performs signature validation processing on signature metadata based on that obtained in the metadata provisioning step, then it MUST NOT treat the widget as signed until the Widget Resource provisioning step is completed and the following verification checks have been successfully completed:</p> <ul style="list-style-type: none"> - the actual signature metadata contained in the Widget Resource is compared and verified to be identical to that obtained in the metadata provisioning step; - reference validation processing of the digital signature document against the actual assets within the Widget Resource. 	<p>This functionality is necessary to allow the Web Runtime to check that it has access to the referenced root certificate before downloading the Widget Resource.</p> <p>Note that in the case that the referenced root certificate is not present; the Widget Resource MAY still be downloaded depending on the security policy of the consuming device.</p> <p>The Widgets 1.0: Digital Signatures specification [8] generates a signature over the Widget Resource that is included in one or more xml files. These files could be extracted from the Widget Resource and provided to the consuming device separately from the Widget Resource. A signature file contains all of the necessary information to verify the signature value over the <signedInfo> element of the signature element and confirm that the necessary root certificate is available on the device.</p> <p>No protocol for delivering a signature file has been defined by W3C. An option would be to use OMA DL [9].</p>

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0300	The Web Runtime SHALL support processing of certificates that conform to the Wireless Application Protocol WAP Certificate and CRL Profiles Specification [10].	For details of a mobile specific profile of X.509 certificates see the Wireless Application Protocol WAP Certificate and CRL Profiles Specification [10]
AS-0310	<p>In addition to supporting the use of the key usage extension and extended key usage extension, as defined in Widgets 1.0: Digital Signatures [8], a Web Runtime SHOULD support the capability to associate a usage restriction to a root certificate so that either:</p> <ul style="list-style-type: none"> - any end-entity certificate issued under that root is only valid for signing Widget Resources; or - any end-entity certificate issued under that root is only valid for another purpose which excludes signing Widget Resources, e.g. TLS or signing other documents that are not part of a Widget Resource. <p>In the case that the Web Runtime uses a root certificate store shared with other applications, the Web Runtime MUST respect any similar usage restrictions imposed by that root certificate store.</p>	This functionality is necessary to enable the issuing of certificates that can only be used to sign Widget Resources.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0320	The Web Runtime SHOULD support the processing and use of CRLs contained in the signature files defined in [8].	The Widgets 1.0: Digital Signatures [8] is built on the XML Signature Syntax and Processing (Second Edition) [11] and therefore includes the capability to include CRLs in the signature element. This capability allows up-to-date revocation information to be included with the signature enabling more efficient CRL distribution.
AS-0330	The Web Runtime SHOULD support the retrieval and processing of CRLs using the CRL distribution contained within a certificate.	This capability is not currently specified in Widgets 1.0: Digital Signatures [8]. CRL retrieval support is desirable to provide an alternative in the case that an embedded CRL is not present and an OCSP retrieval point is not provided.
AS-0340	The Web Runtime SHOULD support the retrieval and processing of CRLs using the CRL distribution contained within a certificate.	CRL retrieval support is desirable to provide an alternative in the case that an embedded CRL is not present and an OCSP retrieval point is not provided.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0350	<p>The Widget User Agent MUST support the mandatory OCSP client functions and behaviour defined in the Open Mobile Alliance: Online Certificate Status Protocol Mobile Profile specification [12].</p> <p>The Web Runtime SHALL attempt to retrieve a valid OCSP response from the location or locations indicated in the certificate's Authority Information Access extension.</p>	<p>This requirement is to establish a default means of obtaining revocation information.</p>

4 BONDI FEATURE ACCESS

4.1 SCOPE

This section is non-normative.

This section defines the architecture requirements relating to support for BONDI Features, including JavaScript APIs (ie APIs in addition to the standardised client-side DOM APIs supported in the browser environment). BONDI is not prescriptive about the actual mechanism used to implement Features. The requirements in this section relate to the organisation and identification of Features and how web applications gain access to them.

Security aspects of Feature access are defined in Section 5.

4.2 REQUIREMENTS SUMMARY AND RATIONALE

This section is non-normative.

The dependencies of BONDI Web Applications are indicated in terms of one or more *Features*.

A Feature corresponds to specific functionality provided by a Web Runtime, made available by a defined set of Web Runtime behaviours and JavaScript interfaces.

A Feature is identified uniquely by IRI, and is the unit of expression of dependencies by BONDI Web Applications.

BONDI defines a Feature, or a number of Features, for each of its defined interfaces. Any independent entity that also wishes to define and make available a JavaScript API in the BONDI framework must define the corresponding Feature(s). An author of a JavaScript API and associated Features may also define a FeatureSet, also identified by IRI, to be a collection of those Features. A key assumption underlying the framework for Feature access is that BONDI cannot, and should not, attempt to hold back the definition of Features independently; there are too many ongoing initiatives, both public and private, for that to be possible or desirable. Instead, the BONDI framework recognises multiple, functionally overlapping Features are going to exist, and aims to create cohesion around their implementation and deployment, and the application of security policies to their use.

The framework and specific requirements embody the following principles.

Feature addition: the set of JavaScript APIs is not fixed, and is expected to be extended over time, as new features become supportable (and perhaps also as features become obsolete). An example of this might be the creation of a SIM access API where none existed initially.

Feature evolution: Any individual JavaScript API can itself evolve over time, to reflect new device capabilities, or new use cases, or to correct errors or shortcomings in earlier JavaScript API versions. Each individual JavaScript API definition can be associated with a version number.

Decentralised Feature definition: The definition of Feature is not limited to any central body such as a standards organisation; instead, Features can be defined and published (and implemented, but not necessarily portably) by any independent entity. Features might be defined by terminal manufacturers, network operators, special interest bodies, individual service providers or publishers. This requires that a system of identifiers exists for Features which allows for decentralised management. A common way to do this is to use IRIs as Feature identifiers.

The framework is also designed to enable dynamic provisioning of Features, defined as follows.

Dynamic Feature provisioning: the ability to enable a Feature on a device when the device is in the field, via some specific extension mechanism. This does not include general extension mechanisms such as re-flashing the entire device. The specific extension mechanism might be a mechanism supported by the underlying operating system (e.g. ActiveX), or a mechanism that is internal to the Web Runtime (such as browser extensions or plug-ins).

However, it is not mandatory that devices include a mechanism for dynamic Feature provisioning, and no standardised (ie interoperable) mechanism is defined for Feature provisioning.

The implications of these principles are given below.

Features, including both interfaces and implementations of those interfaces, are identified by IRI. Usual IRI namespacing conventions can therefore be used to manage the namespace of JavaScript API identifiers. Any centrally defined JavaScript APIs can be defined within an IRI belonging to the defining organisation.

Any Web Application (whether a Website or Widget) must explicitly declare the Features it intends to use. This declaration can take the form of a static declaration in the Widget Configuration Document for a Widget Resource, or can be a programmatic declaration by using a BONDI-defined API. This declaration allows for:

- a check that the Web Runtime supports the Feature in question;
- dynamic provisioning of the Feature, in implementations that support this;
- an access control check that the Web Application in question is permitted to use that Feature. This delivers flexibility and usability

benefits to the BONDI Security Framework as compared with simply enforcing access control at the point that JavaScript APIs are called, or JavaScript APIs attempt to perform security-relevant platform operations.

- the ability of a Web Runtime to resolve, based on local circumstances or other specified parameters, which particular implementation or configuration of a Feature to load or bind to. This also mirrors the approach increasingly followed by the toolkits and web API frameworks, wherein APIs are programmatically loaded rather than being explicitly and directly referenced by a `<script>` tag.
- In the case of statically declared dependencies, the ability for a Widget User Agent to determine prior to installation whether or not a Widget is capable of running on the target Web Runtime.

Feature dependencies are expressed as an IRI. It is the responsibility of the author of the Feature in question to define the Feature and any associated API, and to advertise the corresponding IRI. It is also possible, using standard IRI techniques, to create a family of IRIs so that the Feature definition is “parameterised”. It is up to the author of the Feature definition to specify the valid usage and meaning of the associated IRIs in this case.

BONDI itself defines two kinds of Feature:

- **BONDI JavaScript APIs.** These Features correspond directly to BONDI-defined APIs. IRIs for BONDI-defined JavaScript APIs SHALL be defined from <http://bondi.omtp.org/api/>. The IRI definition, including any subfeatures is defined for each BONDI API in [4].
- **BONDI processing assertions.** These features consist of metadata, assertions and other directives that relate to the processing of Web Applications. IRIs for BONDI processing assertions SHALL be defined from <http://bondi.omtp.org/meta/>. The IRI definition and associated parameters are defined in Appendix A of this document.

For Widgets, BONDI provides a means for the Feature dependencies to be indicated statically, using the `<feature>` element in the Widget Configuration Document.

For both Widgets and Websites, BONDI defines a JavaScript API that sites can use to indicate dependencies programmatically.

4.3 EXPRESSION OF DEPENDENCIES

This section is normative.

4.3.1 **DEFINITIONS**

BONDI defines the following terms.

A **Device Capability** is a specific resource, or functionality of a device, that can be accessed, manipulated or exploited by a Web Application. Device Capabilities are defined and identified in a portable way, without a dependency on any specific JavaScript API, or on any underlying software platform or platform-specific API.

A **JavaScript API** is a program interface for Web Applications defined using an Interface Definition Language (IDL). JavaScript APIs are usually provided as a means for a Web Application to gain access to Device Capabilities. However, the definition of the API itself concerns the interfaces, methods, properties and other attributes that make up the API; the definition of the API is not necessarily associated with any specific Device Capabilities and, by itself, access to an API does not imply access to any underlying Device Capabilities.

A **Feature** is a set of JavaScript APIs and/or device behaviours that provide access to specified Device Capabilities. A Feature is identified uniquely by IRI, and is the unit of expression of dependencies by BONDI Web Applications.

A **Feature Set** is a set of Features, defined in terms of the IRIs of its constituent Features. A Feature Set is identified uniquely by IRI.

4.3.2 **GENERAL**

The dependencies of a Web Application are expressed by identifying the Features on which the Web Application depends.

Each Feature dependency of a Web Application **MUST** be expressed explicitly, either statically (see 4.3.2 below) or programmatically (see 4.3.3 below).

A Web Runtime **SHALL** ensure that no Feature is made available to a Web Application unless an explicit request has been made for that Feature, and access to that Feature is permitted by the Security Policy.

Where a Feature is associated with a JavaScript API, the global properties belonging to that API (including any instances of its interfaces, or public constructors for its interfaces) only become accessible to an instance of a Web Application after it has expressed a dependency on the corresponding Feature, and access to the Feature has been permitted by the Security Policy.

Requirements

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0351	The Web Runtime MUST only enable a Web Application to use a JavaScript API if a dependency has been explicitly expressed on the corresponding Feature and access to the Feature has been granted.	The Web Runtime MUST ensure that only the explicitly declared dependencies are accessible to a running Web Application.
AS-0352	It MUST be possible for Feature dependencies of a Web Application to be resolved to already-available Features.	The simplest case is a web Runtime that contains a fixed collection of Features / JavaScript APIs, embedded in the Web Runtime at the time of manufacture, and the Web Runtime makes these JavaScript APIs available to Widgets on request.
AS-0353	It MAY be possible for Feature dependencies of a Web Application to be resolved to dynamically provisioned Features.	If supported by the Web Runtime, it would also be possible for the Web Runtime to satisfy the dependency by provisioning an implementation of the requested Feature on demand, at the time the specific Feature dependency is first expressed.
AS-0354	Resolution of each Feature request SHALL include an access control Query that determines whether or not the Web Application is authorised to access to the Feature in question. Failure to resolve and satisfy any dependency SHALL result in the requested Feature being unavailable to the Web Application.	Any access to a requested Feature is conditional, depending on the access being granted by the applicable Security Policy. The definition of the static and programmatic means for expressing dependencies includes the definition of the timing of the access control Query and the specific error conditions resulting from a failure condition.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0355	Failure to express a Feature dependency SHALL result in a non-fatal run-time error each time an operation is attempted that calls the JavaScript API associated with (or otherwise depends on) that Feature.	<p>The Web Runtime MUST ensure that only the explicitly declared dependencies are accessible to a running Web Application.</p> <p>Any attempt to access a JavaScript API that is not available (eg by accessing a global property belonging to the API) will fail in the same manner as if the API is absent (for example by throwing a <code>ReferenceError</code> when an attempt is made to access one of its global properties).</p> <p>The Web Runtime MUST ensure that only the requested Feature(s) are available when using any associated JavaScript API. Any attempt to access Device Capabilities, or other functionality of the API that is not covered by the requested Feature(s) SHALL result in a <code>SecurityError</code> being thrown with code <code>PERMISSION_DENIED_ERROR</code></p>

STATIC EXPRESSION OF DEPENDENCIES

This section is normative.

A Feature dependency of a Widget MAY be expressed statically by including a `<feature>` element within the `<widget>` element in the Widget Configuration Document.

The processing of each `<feature>` element is defined by BONDI based on the values of the following attributes:

- name
 - Contains the IRI of the requested Feature (W3C-defined attribute)
 - Use: Required

- Type: IRI String
- `required`
 - Indicates whether the dependency in question is mandatory (`required = true`) or optional (`required = false`) (W3C-defined attribute).
 - Use: Optional (default is `true`)
 - Type: Boolean

Requirements

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0360	The Web Runtime MUST support static expression of Feature dependencies by a Widget Resource using the <code><feature></code> element defined in Widgets 1.0: Packaging and Configuration [7].	This information will be used to determine whether the requested Feature is available on the target Web Runtime (and if it is not available, MAY provide information allowing the Feature to be fetched by the Web Runtime).
AS-0370	The Web Runtime MUST support processing of the <code>required</code> attribute of the <code><feature></code> element by which a Widget Resource declares whether a dependency is mandatory or optional. If the <code>required</code> attribute is omitted, the dependency MUST be treated as optional.	This information MAY be used by the Web Runtime to determine which security policies are relevant to the installation of the Widget Resource. For example, if an optional Feature is not accessible to the Widget, either because the Feature is not available in the Web Runtime or the Security Policy denies access, the Widget Resource could still be installed.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0390	The Web Runtime SHALL resolve all dependencies of Features referenced statically at install time, or at instantiation time for Widget Resources that are instantiated without prior installation.	Each Feature resolves to one or more JavaScript APIs. The Web Runtime MUST determine at installation time whether or not the requested JavaScript APIs are accessible.
AS-0430	As part of the resolution of each statically referenced Feature, the Web Runtime SHALL evaluate an access control Query that determines whether or not the Widget is authorised to access the requested Feature. If the evaluated Decision of that Query is Deny, and the dependency is marked as <i>required</i> , the Web Runtime SHALL abort installation and it SHALL not be possible to instantiate that Widget.	Any access to a requested Feature is conditional, depending on the access being granted by the applicable Security Policy.
AS-0440	If permission is not granted for that Widget to access that Feature, and the dependency is marked as optional, the installation SHALL proceed. Access to any associated JavaScript API at runtime SHALL fail if the dependency remains unsatisfied at the time the Widget is instantiated.	Dependencies that are not <i>required</i> will not prevent installation and instantiation of a Web Application. If a dependency was available, but access to that Feature was denied by Security Policy at the time of installation, it is nonetheless possible that the Feature is accessible at the time of instantiation as a result of some intervening change to Security Policy. In this case, any attempt to access the Feature at run time MAY succeed.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0460	The Web Runtime MUST support static expression of dependencies identified by a FeatureSet using the <feature> element of the Widget Configuration Document.	A <feature> declaration that references a FeatureSet is equivalent to a series of <feature> declarations for each of the constituent Features. If the declaration has a required attribute value of true, then all of the constituent Feature dependencies MUST be satisfied in order for the FeatureSet dependency to be satisfied.

4.3.3 PROGRAMMATIC EXPRESSION OF DEPENDENCIES

This section is normative.

A Feature dependency of a Web Application MAY be expressed programmatically by calling the BONDI-defined requestFeature() API (Web IDL below)

```
[NoInterfaceObject]
interface bondi {
    PendingOperation requestFeature(
        in RequestFeatureSuccessCallback successCallback,
        in ErrorCallback errorCallback,
        in DOMString name)
    raises(DeviceAPIError, SecurityError);
};
```

This function requests a named Feature asynchronously and returns a pending operation object. If it succeeds it calls the successCallback. If it fails it calls the errorCallback passing in a DeviceAPIError which provides an error message and error code indicating the nature of the error.

If a JavaScript API is associated with the requested Feature, then all global properties of that API are made accessible prior to the successCallback being invoked. The definition of the API MAY optionally designate one of its interfaces as a "root" interface, such that a singleton instance of that interface provides access to the functionality of the API. If such a root interface is defined, and instance of that interface is passed as an argument to the successCallback if the Feature request is successful.

Parameters

- successCallback: the success callback function

- errorCallback: the error callback function
- name: the feature name IRI

Return value

PendingOperation enabling the requester to cancel this request.

Exceptions

The errorCallback will receive one of the following errors:

- DeviceAPIError.NOT_FOUND_ERROR if the requested feature could not be found
- DeviceAPIError.INVALID_ARGUMENT_ERROR if a malformed argument has been supplied or a required argument has been omitted.
- SecurityError.PERMISSION_DENIED_ERROR if the requested feature is not permitted to load/bind or that access to a required device capability has been denied.
- DeviceAPIError.UNKNOWN_ERROR if some other error occurred .

The requestFeature() API is itself a Feature, identified by:

`http://bondi.omtp.org/api/bondi.requestfeature`

Websites have access to this API automatically and this is the sole means of expression of Feature dependencies by Websites.

Access to this API for a Widget is conditional on the Widget making a static declaration via a <feature> element expressing a dependency on the above Feature.

Requirements

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0471	The Web Runtime MUST support programmatic resolution of Feature dependencies for Websites via the requestFeature API.	

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0472	<p>The Web Runtime MUST support programmatic resolution of Feature dependencies for Widgets via the requestFeature API. The Web Runtime MUST ensure that access to the requestFeature API for a Widget is enabled only if the Widget has statically expressed a dependency on the associated Feature.</p>	<p>A Widget MUST indicate a dependency statically on http://bondi.omtp.org/api/bon-di.requestfeature in order to have access to the requestFeature() API. As with any other Feature request, an access control Query MUST be evaluated to determine whether or not the Widget is authorised to use that Feature.</p>
AS-0480	<p>Programmatic resolution of each Feature request SHALL include an access control Query that determines whether or not the Website is authorised to access to the Feature in question. Failure to resolve and satisfy any dependency requested programmatically SHALL generate a non-fatal run-time exception.</p>	<p>Any access to a requested Feature is conditional, depending on the access being granted by the applicable Security Policy. The definition of the requestFeature() API includes the definition of the specific Errors thrown in each failure condition.</p>
AS-0491	<p>The Web Runtime MUST support programmatic expression of dependencies identified by a FeatureSet. A request for a FeatureSet succeeds if and only if requests for each of the constituent Features succeeds.</p>	<p>If a request for any one of the constituent Features fails, then the request for the FeatureSet also fails. In this case the Web Runtime MUST ensure that none of the constituent Features is made available to the Web Application.</p>

5 SECURITY FRAMEWORK FOR FEATURE ACCESS

5.1 INTRODUCTION

This section is non-normative.

BONDI specifies the security framework responsible for controlling access to JavaScript APIs in order to mitigate the risks created by allowing Web Applications access to Device Capabilities. The definition of the framework is organised into two parts, as follows.

- **Security Policy framework.** This is the general conceptual framework within which BONDI policies are formulated, plus a detailed formal model of a Web Runtime Security Policy. This defines the entities involved in any access control decision (ie the subject(s), resources etc) and the form of the access control rules that are applied in order to make those decisions.
- **Security Policy management.** This covers a set of procedures and supporting technical enablers relating to the management of Web Runtime security policies.

The formal model underlying the general framework, and textual representation of associated policies, are defined in detail in Appendix B of this document. This section provides an overview of the Security Policy framework and defines the associated requirements. Security Policy management requirements are addressed in Section 6.

5.2 FRAMEWORK OVERVIEW

This section is non-normative.

The BONDI security model is based on a clear separation between the definition of the general framework and the creation of specific Security Policies. This allows the configured Security Policy of a Web Runtime to be adapted to the needs of different stakeholders at any time during the lifecycle of a product. This includes, but is not limited to, the embedding and configuration of Web Runtimes at manufacture as well as post-sales installation of Web Runtimes.

As a general high-level objective, it should be noted that the security framework is intended to protect the device and its user from Web Applications that pose a security risk from unintentional but undetected implementation flaws as well as those that are malicious.

The intention is that the framework, at the most general level, allows a very wide range of policies to be represented (although manageability and

interoperability concerns require there to be a certain level of similarity and compatibility in policy structure). However, the framework must permit fine-grained security policies to be represented as well as policies based on broad groupings of APIs and assignment of web applications to a small number of trust domains. For example, a fine-grained security policy is necessary to grant or deny access to individual APIs for individual web applications.

The framework is based on a very general model that governs access by *subjects* to *resources* based on a hierarchy of *policies* and *policy sets*, where each policy consists of a number of *rules*. Subjects and resources are characterised by a number of defined *subject attributes* and *resource attributes*. A range of attributes is defined so that policies can be expressed controlling access based on a Widget Resource signer's identity, or an individual Widget Resource identity, or the Widget Resource signature's root certificate, or a Website's URL. Therefore, the generality of the BONDI framework derives from the range of attributes that are supported, as well as the flexible structure of policies and policy sets in the security model.

The BONDI model is defined using concepts, terminology and semantics from the eXtensible Access Control Markup Language (XACML) [13] framework. BONDI policies are capable of representation in a compact XML format (and other formats, including a compact binary representation if necessary).

It is intended that BONDI policies are also eventually capable of representation in XACML, using a specific dictionary of attributes and a subset of XACML elements; however this is not currently possible without defining a number of extensions to XACML. It is hoped that this becomes possible with future revisions of the XACML standard.

5.3 LAYER MODEL

This section is non-normative.

A Web Application uses a BONDI-specified mechanism to gain access to (or *bind to*) a *JavaScript API* by stating one or more *Features* exposed by the JavaScript API. The implementation of this API, in turn, makes use of one or more *Device Capabilities*, defined in terms of facilities that might be provided by APIs at the device, platform or OS level, but without reference to any particular API. Where a JavaScript API is defined within the BONDI model, each function definition specifies which Feature it implements, and which Device Capabilities it uses.

The mechanisms provided in BONDI to bind to APIs, and to control access (both to the Features and the Device Capabilities exposed by them) are generic, and are not themselves dependent on any particular set of JavaScript APIs. The BONDI model envisages that these controls apply, irrespective of whether the APIs in use are defined by BONDI or any independent entity. This

“layering” of JavaScript APIs and mediating access control is illustrated in the figure below.

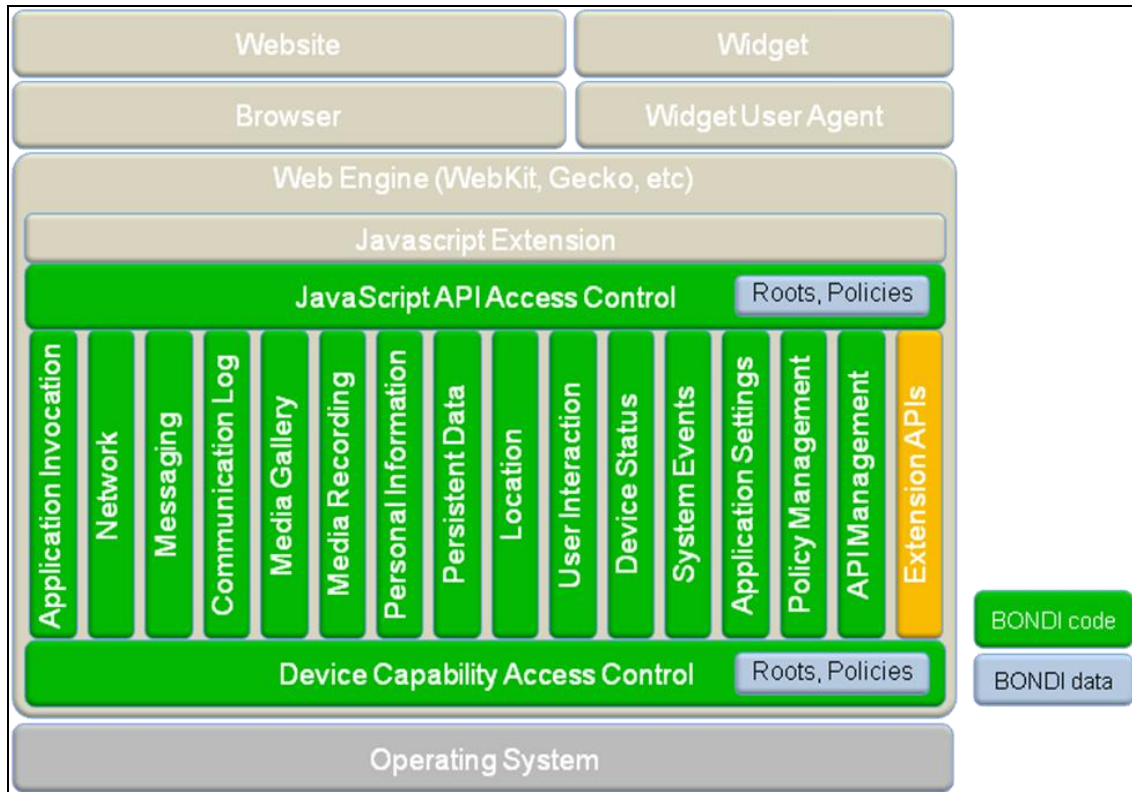


Figure 2: JavaScript API Access Control

Figure 2 shows a *JavaScript API Access Control layer* which controls access to all JavaScript APIs exposed by the Web Runtime. Each Feature is identified uniquely by URI, and this security layer mediates access to Features on the basis of that ID.

Figure 2 also shows a *Device Capability Access Control layer* which controls access to the underlying capabilities of the device when used from JavaScript APIs. These Device Capabilities themselves are identified so that it is possible to write security policies that control access to specific capabilities independently of the JavaScript APIs used to access them.

This explicit separation of Features and their dependent Device Capabilities addresses a number of significant requirements, as follows.

Extensibility is an intrinsic part of the BONDI model. BONDI expects that APIs will be defined and implemented independently, and the nature of those APIs will not necessarily be known to the author of a Security Policy. Therefore, if a Security Policy author wishes to deny access to a specific Device Capability, then there must be a way of doing this that is independent of the JavaScript API being used.

The Web Runtime must only grant access to Features that are advertised as being dependencies of the Web Application. As stated in Section 4, mechanisms are provided both for Websites and for Widgets to indicate their dependencies on Features (either programmatically or statically). The access control system must ensure that only those requested Features are accessible to the Web Application. This requires that the access control system is able to control access based on the ID of a Feature.

There will not be a direct 1-1 correspondence between JavaScript APIs and Device Capabilities. Although there will be simple JavaScript APIs that provide access only to a single Device Capability, it must be expected that there are also more complex APIs that expose multiple Device Capabilities; examples might include a camera API that provides the ability to geotag a photo with the current location, or a messaging API that provides the ability to access documents stored locally and attach them to outgoing messages. Therefore, enabling or disabling access to a specific Device Capability will not directly correspond to enabling or disabling access to a single JavaScript API.

Implementations of JavaScript APIs need not be as highly trusted as the Web Runtime. Authors of Security Policies may require the ability to control access to specific JavaScript APIs, or families of APIs, based on the identity of the API (and not just the Device Capabilities it exposes), according to the trustworthiness of the author of the API.

It must be possible to represent security Policies portably. This implies that all identifiers used in a Security Policy (both for Features and for Device Capabilities) must be portably defined, and not (for example) based on any platform-specific API names. This requires that the identifiers for Device Capabilities are defined in a platform-independent way; BONDI defines a set of these capabilities in Appendix A.

5.4 LOGICAL MODEL

This section is non-normative.

The BONDI access control system, from a logical perspective, mediates any attempt by an executing Web Application to access Device Capabilities using JavaScript APIs. The access control system, implementing a specific access control policy, has the sole effect of making and enforcing an access control decision in relation to each attempted access. (In order to make that decision the access control system may request interactive confirmation from the user, but this is invisible to the requesting Web Application.)

The access control system itself consists of a number of logically distinct elements. Specific BONDI requirements and interfaces are specified in terms of these separate functional components. This logical breakdown and associated terminology is adopted from XACML and illustrated below.

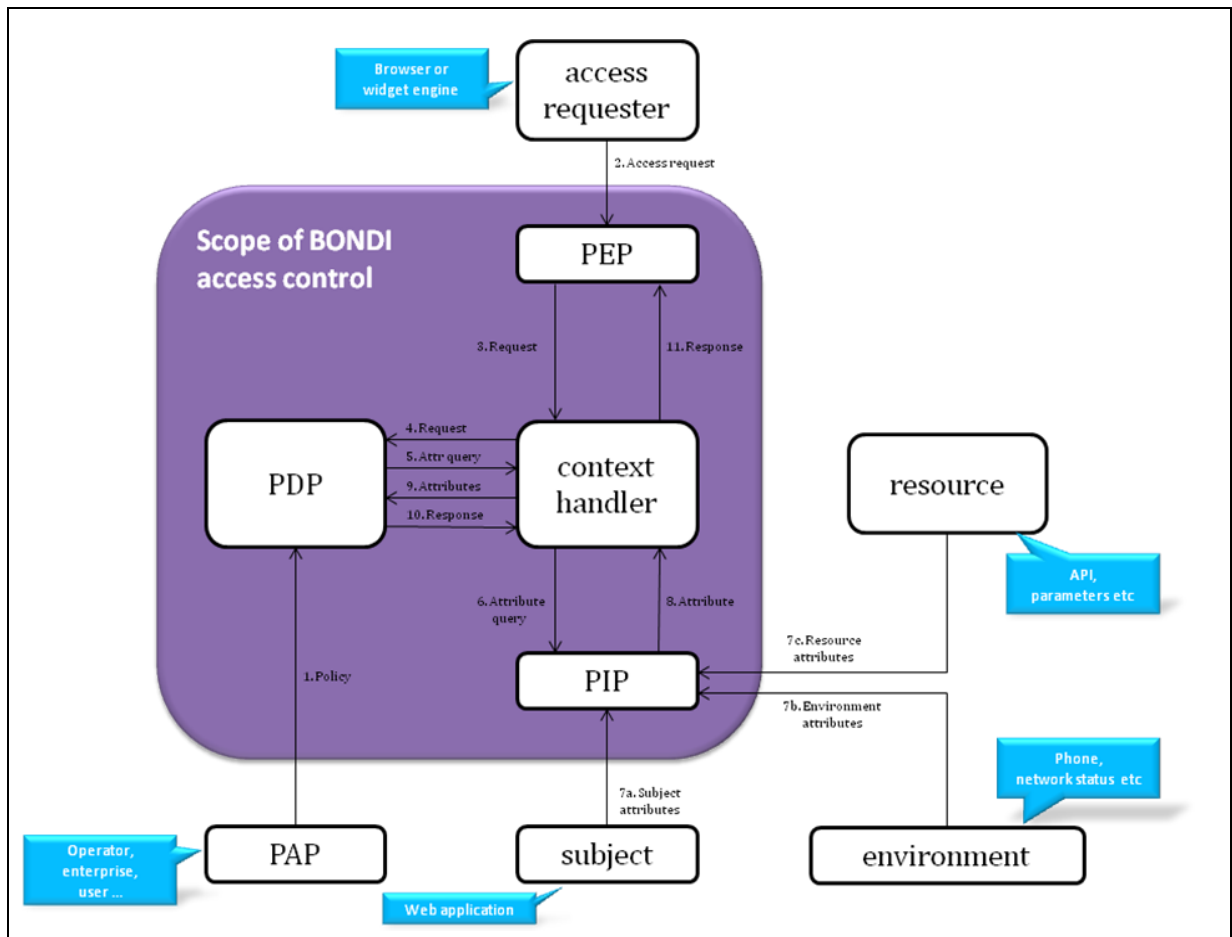


Figure 3: The BOND security mode, derived from XACML Specification Schema [13]

The specified functional components are as follows:

- **Policy Decision Point (PDP):** the module that evaluates whether or not a device API can be accessed by a web application, based on the current policy.
- **Policy Enforcement Point (PEP):** the module that allows or prevents access to device APIs.
- **Policy Information Point (PIP):** the module that gathers information to be used by the PDP to evaluate an access control request. In particular it collects the subject attributes (i.e. how the web application was identified and its associated security attributes), the resource attributes (i.e. which device API is being requested and using which parameters) and the environment (i.e. status of the device).
- **Policy Administration Point (PAP):** the authority that defines the policy. It could be a network operator, a terminal manufacturer, a web

runtime developer, an enterprise or a user at run-time. Policies can be provided by the PAP in different ways, for instance using a preloaded file or data structure, or a remote management mechanism (e.g. OMA DM). In the present phase, BONDI does not define any specific external interface requirements for the PAP, although this may be in scope for later phases. The security policy itself is not defined by BONDI, although a default policy may be recommended by BONDI at a future stage.

The functionality required in each of these components is specified in terms of the following entities:

- **Subject:** the Web Application that requires access to JavaScript APIs. Examples of subjects are Websites and Widgets.
- **Subject Attribute:** Each subject is associated with a set of attributes. Subject attributes include specific attributes that represent the identity of the Web Application attempting access to a resource. Valid identity attributes include the Widget identifier URI for Widgets and the URL for Websites; other identifiers may be supported. Subject attributes also include the credentials used to verify the authenticity and integrity of the subject, e.g. a TLS or code signing digital certificate. Other credentials may be supported.
- **Resource:** the resources that subjects may request access to. The device features or services (e.g. the location module or PIM database) are the actual resources that are being protected, but from the point of view of the security framework these resources are associated with the API Features and Device Capabilities used to access them.
- **Resource Attribute:** each resource is associated with a set of attributes. Resource attributes include an identifier. Other attributes may be associated with a resource, and these can include specific parameters that are specified as part of the request when attempting access.
- **Environment:** the collection of device status and context attributes that may be relevant to the circumstances of a resource access attempt, but are not directly associated with either the Subject or Resource. For example, Environment attributes can include terminal charging, network connection status, whether roaming.

5.5 ACCESS CONTROL POLICY STRUCTURE

This section is non-normative.

The policy in effect in any given context is logically expressed as a collection of specific access control **rules**. The rules are organised into groups, termed

policies, and these in turn are organised into groups termed **policy sets**. This structuring serves a number of purposes:

- the rules that apply to a specific Web Application, or group of Web Applications, can be grouped together, which simplifies writing and maintaining the policy;
- it helps organise the rules into groups that can be independently created and maintained, sometimes under different authority (and subject to differing degrees of control);
- it provides a way of ensuring that the correct precedence is applied when processing rules. This makes some rules easier to write because their applicability is more narrowly scoped by their enclosing policy. More significantly, it ensures that security requirements determined by one authority are not wrongly overridden by rules provided by a subordinate authority.

Simplistically, each rule is specified by defining a **condition**, which is a set of statements which must be satisfied in order for that particular rule to apply, and an **effect** which represents the rule’s outcome – ie whether that rule indicates that the access request should be permitted or not.

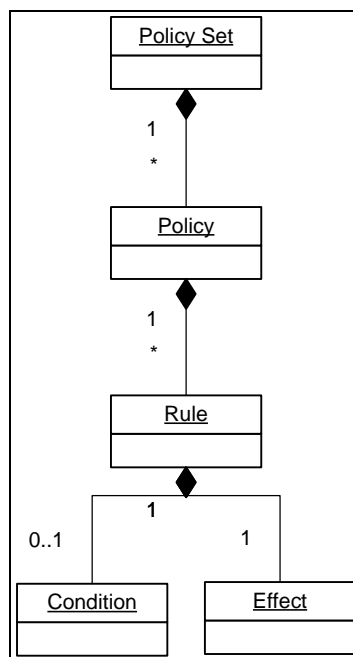


Figure 4: Policy language model, derived from XACML Specification Schema [13]

The specific details of how rules are constructed, and how they are organised into policies and policy sets, are defined in Appendix B.

5.6 RULE PROCESSING

This section is non-normative.

The following steps illustrate the logical processing required when a Policy Set is used to determine the outcome of a particular access attempt:

- the identity of the calling Web Application (Subject) is known and is used to determine the subject attributes for that Web Application.
- the application in question attempts an action (attempts to invoke a JavaScript API, say). This identifies the Resource and all associated Resource Attributes including `api-feature` and, where applicable, `device-cap` Resource Attribute if the action entails use of a Device Capability. Any parameters used by any such Device Capability use, where designated as being security-relevant, are also captured within a `param` Resource Attribute.
- the Environment attributes are also captured.
- the set of Subject, Resource and Environment Attribute values so captured is embodied in a Query which is evaluated in the context of the top-level policy-set associated with the configured Security Policy;
- based on the Subject, Resource, and Environment Attribute values presented in the Query, the Conditions associated with each Rule and Target can be evaluated, and the applicable Rules, Policies and Policy-sets can be determined;
- the Effect of each applicable Rule is determined;
- based on the evaluated Effect of each applicable Rule, the relevant combining rules are used to determine the Policies, Policy-sets and Rules with precedence, which in turn determines the overall Effect of evaluating the Query.
- if the Effect requires the user to be prompted, this is done, and the results of the user's decision are recorded where appropriate.

From a logical perspective, an access control Query can be evaluated at any time between the point that the necessary attribute values become available and when the attempted operation is performed. In the case of a JavaScript API call for which arguments supplied to the call are designated Resource Attributes, the relevant attributes are only known once the API call has been made by the calling Web Application. However, in other cases the information may be known earlier. For example, if all Feature requirements are stated in a Widget Configuration Document, and none of the API access operations have

conditions depending on API call parameters, then all access control Queries may be evaluated fully at Widget Resource installation time.

5.7 REQUIREMENTS SUMMARY AND RATIONALE

This section is non-normative.

The framework defines an abstract model for a security policy, plus a concrete representation of that for interchange and configuration purposes. The BONDI requirement is simply that a Web Runtime is capable of configuration with a Security Policy, and its internal representation is complete and faithful to the defined model, supporting all required elements and attributes. A Web Runtime must also use this configured Security Policy as the sole basis on which access control decisions are made, and then reliably enforced, whenever a Web Application attempts to make use of a JavaScript API.

Requirements relating to configurability of a Web Runtime, interactively by the user, by import of policy documents and by device management, are covered in Section 6.

5.8 REQUIREMENTS

This section is normative.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0500	A Web Runtime SHALL use a configured Security Policy as the sole basis on which access control decisions are made when a Web Application attempts to make use of a Feature on which it has previously expressed a dependency.	

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0505	A Web Runtime SHALL verify that each use of each Feature is permitted by evaluating the Feature request against the configured Security Policy in accordance with the evaluation rules specified in Appendix B. The evaluation SHALL be performed at a time such that the result of the evaluation is determined.	When storing the configured policy, and making access control decisions based on that configured policy, the Web Runtime SHALL fully implement the model specified in Appendix B. Where the outcome of the access control decision depends on Environment or Resource attributes that are undetermined until instantiation or invocation time, the associated parts of the policy evaluation MUST be deferred until that time.

6 SECURITY POLICY MANAGEMENT

6.1 INTRODUCTION

This section is non-normative.

The BONDI security model is based on a clear separation of the specification of the security framework, the definition of Security Policies and the management and provisioning of these Security Policies. This allows adaptation of a Web Runtime's Security Policies to the needs of the different stakeholders at any point of time during the lifecycle of a Web Runtime. This includes, but is not restricted to, the manufacturing process of a terminal that integrates a Web Runtime as well as to remote Security Policy management post-sales.

A Web Runtime's Security Policies are by nature security-critical system components that limit the misuse of sensitive Web Runtime functions by Web Applications. Therefore, it is necessary to protect the security model from compromise by unauthorised Security Policies but also to maintain sufficient flexibility to those stakeholders that are authorised to establish Security Policies.

Requirements in this section relate to Security Policy provisioning and management aspects. Requirements are defined in two areas:

- general Security Policy management;
- policy governing the installation and management of Extension APIs.

6.2 GENERAL SECURITY POLICY MANAGEMENT

6.2.1 SCOPE

This section is non-normative.

At this release, BONDI specifies a number of mechanisms that must be supported for configuration of Web Runtime Security Policies, and the interchange format for those policies.

BONDI is not prescriptive about which entity should be the authority for definition of Web Runtime Security Policies. However, it is believed that in many practical situations it is not appropriate for there to be a single authority or point of configuration for a device's entire policy; hence there is a need to examine mechanisms that enable security policy configuration to be partitioned into multiple areas, each with a different management authority. However, mechanisms to enable multiple authorities, and to delegate configuration rights for confined policy elements from one authority to a subordinate authority, are beyond the scope of this BONDI version.

6.2.2 **REQUIREMENTS SUMMARY AND RATIONALE**

This section is non-normative.

The requirements here are based on the assumption that a device has a locally stored Security Policy, which is made up of one or more separately configurable fragments, each in BONDI XML security policy format. No structure is imposed by BONDI at this release on the partitioning of complete Security Policies into fragments.

The requirements are organised under the following headings.

Signed Security Policy Document processing. In order to assure the integrity, authenticity and interoperability of Security Policy updates, a format is defined for a Signed Security Policy, allowing one or more Security Policy documents to be signed and disseminated by an authorised entity. The signature format is based on the XML Digital Signature standard [11].

Security Policy provisioning. The obvious consequence of a rich and configurable policy is the need for a means of establishing and maintaining that configuration. A series of requirements relating to the delivery of policy documents to a device, and establishing those documents as active security policies. In order to maximise the practical effectiveness of Security Policy provisioning, BONDI also includes requirements to support an interoperable Security Policy representation.

Remote Security Policy management. Remote configuration is a practical operational necessity for both enterprises and operators. It is particularly important that security policy can be effectively maintained, so BONDI defines a series of requirements relating to the remote configuration of policies, or policy fragments, using standardised device management mechanisms. Support for the OMA DM standard is required, and standardised definitions of management objects will be provided in a later BONDI release.

User security settings. The Device Capabilities that are exposed by BONDI interfaces cover a wide range of functionality, with a similarly wide range of security implications, including device integrity, service availability, compromise of privacy, and abuse of chargeable services. In many areas the user has a legitimate right to configure policy settings, even if default settings are initially set by the device manufacturer or operator. To ensure consistency of user experience, and to ensure that policies are able to reflect intended behaviour faithfully, BONDI includes specific requirements relating to user policy configuration.

6.2.3 **SIGNED SECURITY POLICY DOCUMENT PROCESSING**

This section is normative.

The following requirements apply to the processing of a Signed Security Policy Document, whether used as part of a provisioning or remote management procedure.

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0510	<p>A Web Runtime MUST only accept Signed Security Policies from authorised security policy provisioning authorities.</p> <p>A Signed Security Policy SHALL be identified as coming from an authorised security policy provisioning authority if the end-entity certificate associated to the signature is either contained in the set of authorised policy certificates or chains to a certificate in the set of authorised policy certificates.</p> <p>Certificates in the set of authorised policy certificates MUST only be used by the Web Runtime for the validation of Signed Security Policies.</p>	<p>A conforming Web Runtime is expected to support a configurable set of signer (end-entity) certificates authorised to sign Security Policy Documents, and a configurable set of root certificates to which signer certificates MUST chain, as alternative means of establishing the authority of a security policy provisioning authority.</p>

REQ. ID	REQUIREMENT	DESCRIPTION
<p>AS-0520</p>	<p>The Web Runtime SHALL validate a Signed Security Policy Document by validating the signature in accordance with the validity rules for a Signed Security Policy Document as defined in this specification.</p> <p>To process the signature element of a Signed Policy Document:</p> <ul style="list-style-type: none"> ▪ The Web Runtime MUST support the required Signature Algorithms specified in [8]; ▪ The Web Runtime MUST support the required Digest Algorithms specified in [8]; ▪ The Web Runtime MUST support the required Canonicalization Algorithms specified in [8]; ▪ The Web Runtime MUST support the mandatory certificate format specified in [8]. <p>If signature validation fails for any reason the Signed Policy Document MUST NOT be installed by the Web Runtime.</p>	<p>The defined Signed Security Policy Document format MUST be supported to ensure the integrity and authenticity of imported and provisioned Security Policies.</p> <p>A minimum set of support requirements for algorithms and certificate format is specified to promote interoperability of Signed Security Policy Documents.</p>

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0530	The Web Runtime SHALL support at least one security policy provisioning authority.	At this release, BONDI only requires conforming Web Runtimes to support a single provisioning authority for the entire BONDI Security Policy, but does not preclude implementations that support multiple provisioning authorities.

REQ. ID	REQUIREMENT	DESCRIPTION
<p>AS-0540</p>	<p>The <code><signature></code> element MUST be validated according to Core Validation, as defined in XML Signature Syntax and Processing (Second Edition) [11], with the additional rules defined below:</p> <ul style="list-style-type: none"> ▪ The signature element MUST contain one or more valid XML <code><Reference></code> elements; ▪ The URL attribute of each XML Reference element MUST contain a reference to a policy element that is a sibling of the <code><Signature></code> element; ▪ The XML Reference element MUST NOT have any <code><Transform></code> elements; ▪ The Web Runtime MUST treat the Signed Security Policy Document as invalid if it contains a <code>policy</code> or <code>policy-set</code> element for which there is no XML <code><Reference></code> element. 	<p>These requirements relate to the well-formed nature of a <code><signature></code> element in a Signed Security Policy Document.</p>

6.2.4 SECURITY POLICY PROVISIONING REQUIREMENTS

This section is normative.

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0550	The Web Runtime SHALL provide a means to import a Security Policy.	
AS-0560	The Web Runtime SHALL support the BONDI Signed Security Policy Document format as an import format for Security Policies.	At this release, BONDI requires that Signed Security Policy Documents are supported, among any other policy document formats that the Web Runtime might support. At a future release BONDI might also require support for (unsigned) Security Policy Documents, to be used in conjunction with some other import mechanism that independently provides assurance of integrity and authenticity.
AS-0565	The Web Runtime SHALL support at least the UTF-8 encoding for Security Policy Documents and Signed Security Policy Documents.	This is to guarantee at least one encoding in which Policy documents are known to be interoperable.
AS-0570	The Web Runtime SHALL validate an imported Signed Security Policy Document.	Validation of the imported Security Policy SHALL be in conformance to the general Signed Security Policy Document processing requirements.
AS-0580	The Web Runtime SHALL support total update of an imported security policy. The Web Runtime SHALL replace the Security Policy that was previously imported with the new Security Policy.	At this release, BONDI requires support only for complete replacement of a Web Runtime Security Policy using a single Signed Security Policy Document.

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0581	A Security Policy document containing a root <code>policy</code> or <code>policy-set</code> with no optional <code>id</code> attribute SHALL be treated as a total update.	
AS-0582	A Signed Security Policy document SHALL be treated as a total update if and only if its <code><signed-policy></code> element contains a single child <code><policy></code> or <code><policy-set></code> with no <code>id</code> attribute.	
AS-0585	<p>The Web Runtime MAY support partial update of an imported security policy.</p> <p>If partial update is supported, the Web Runtime SHALL update the previous Security Policy with the replacement <code>policy</code> or <code>policy-set</code>.</p> <p>If the Web Runtime is unable to process a partial update, the update SHALL be ignored.</p>	<p>An implementation MAY also support the update of individual parts (or <i>fragments</i>) of a Security Policy and MAY use the optional <code>id</code> attribute of a <code>policy</code> or <code>policy-set</code> to associate the update document with the corresponding fragment of the previously imported Security Policy.</p>

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0586	<p>A Signed Security Policy document SHALL be treated as a partial update if and only if its <code><signed-policy></code> element contains one or more child <code><policy></code> or <code><policy-set></code> elements, each having an <code>id</code> attribute.</p> <p>Each such child <code><policy></code> and <code><policy-set></code> SHALL be processed as separate partial update, using the <code>id</code> attribute to associate that update with the corresponding fragment of the previously imported Security Policy.</p>	
AS-0587	<p>A Signed Security Policy document SHALL be treated as invalid and rejected if it is not a valid total update according to AS-0581 or a valid partial update according to AS-0586.</p>	
AS-0590	<p>The Web Runtime SHALL support retrieval of a Signed Security Policy Document from a remote server, e.g. from a web server via the Internet or from a Smartcard Web Server (SCWS) on a UICC, or an OMA DM [14] server through the OMA DM standard (using an embedded OMA DM client if present).</p>	

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0600	The Web Runtime SHALL support a configurable remote location from which to retrieve a Signed Security Policy Document, as a URL.	
AS-0610	The Web Runtime SHALL support retrieval of a Signed Security Policy Document from local storage, e.g. device memory (filesystem or DM settings repository), removable memory card, SIM/UICC.	
AS-0620	The Web Runtime SHALL support a configurable local storage location from which it will load its Security Policy upon start-up.	
AS-0630	If the Web Runtime separately stores Security Policy internally, it SHALL validate that policy against the configured local policy each time it starts.	
AS-0640	The Web Runtime MAY support user selection of a locally stored file that contains a Signed Security Policy Document to be imported.	Note that the ability of the user to select a policy file might be configurable, ie it might not be allowed by the security policy provisioning authority.

6.2.5 REMOTE SECURITY POLICY MANAGEMENT REQUIREMENTS

This section is normative.

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0650	The Web Runtime SHALL be manageable via OMA DM v1.2 [14].	
AS-0660	The management objects' changes in the DM Tree SHALL take effect as soon as practical after they were initiated via OMA DM to the Web Runtime and were updated in the DM Tree.	
AS-0670	The Web Runtime SHALL monitor changes to its settings in the DM Tree and enact any changes.	
AS-0680	The Web Runtime SHALL expose a configuration parameter in the DM Tree that enables the initiation of a Security Policy update.	
AS-0690	The Web Runtime SHALL support addition of a Security Policy via remote management.	
AS-0700	The Web Runtime SHALL support the BONDI Signed Security Policy Document format as an format for delivery of Security Policies via remote management.	
AS-0710	The Web Runtime SHALL validate an Signed Security Policy Document delivered via remote management. Validation of the delivered Security Policy MAY be performed by the OMA DM client before delivery to the Web Runtime.	Validation of the delivered Security Policy SHALL be in conformance to the general Signed Security Policy Document processing requirements.

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0720	<p>The Web Runtime SHALL support total update of a Security Policy previously delivered via remote management. The Web Runtime SHALL replace the security policy that was previously delivered with the new Security Policy.</p> <p>If the Web Runtime is unable to process a total update, the update SHALL be ignored.</p>	<p>At this release, BONDI requires support only for complete replacement of a Web Runtime Security Policy using a single Signed Security Policy Document.</p>
AS-0725	<p>The Web Runtime MAY support partial update of a Security Policy previously delivered via remote management.</p> <p>If partial update is supported, the Web Runtime SHALL update the previous Security Policy with the replacement policy or policy-set.</p> <p>If the Web Runtime is unable to process a partial update, the update SHALL be ignored.</p>	<p>An implementation MAY also support the update of individual parts (or <i>fragments</i>) of a Security Policy and MAY use the optional <code>id</code> attribute of a <code>policy</code> or <code>policy-set</code> to associate the update document with the corresponding fragment of the previously delivered Security Policy and/or corresponding node of a DM tree.</p>
AS-0730	<p>The Web Runtime SHALL support deletion of a Security Policy previously delivered via remote management.</p>	
AS-0740	<p>The Web Runtime SHALL support retrieval of a Signed Security Policy Document from a remote server, e.g. from a web server via the Internet or from a Smartcard Web Server (SCWS) on a UICC.</p>	

6.2.6 USER SECURITY SETTINGS REQUIREMENTS

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0750	The Web Runtime SHALL support persistent storage of user security settings.	A Web Runtime MUST store user preferences persistently where needed. User preferences MAY arise where there is explicit provision for user configuration, or where a user requests that preferences are remembered as part of a security prompt.
AS-0760	The Web Runtime SHALL by default store user security settings per Web Application.	Settings stored on a per-web application basis will correspond to creating a <code>policy-set</code> whose target corresponds to that web application, and storing the user setting as one or more rules within that <code>policy-set</code> . As a user experience optimisation, a Web Runtime MAY also offer the option of storing user settings on a broader basis with user confirmation, e.g. by applying the settings to all web applications by the same author, or by adding the web application to the target of a pre-existing <code>policy-set</code> .
AS-0770	The Web Runtime SHALL allow the user to view all stored user security settings	It MUST be possible for a user to discover all user-specified changes from the Security Policy established by the security policy provisioning authority.

REQ. ID	REQUIREMENT	DESCRIPTION
AS-0780	The Web Runtime SHALL provide the user the ability to revert a stored user setting	The user MUST have the ability to revert individual changes at the level of granularity that they were originally set by the user. An implementation MAY also provide an enhanced user experience by providing additional flexibility in the management of configured user settings.
AS-0790	The Web Runtime SHALL provide the user the ability to revert all stored user settings as a single operation.	It MUST be possible for a user to revert all user-specified changes in a way that reinstates the Security Policy established by the security policy provisioning authority.

6.3 INSTALLATION AND MANAGEMENT OF EXTENSION APIS

6.3.1 SCOPE

This section is non-normative.

The BONDI architecture for Device API access defined in Section 4 supports the idea of extensibility. This means, on devices that support such a feature, that new JavaScript APIs (*Extension APIs*) could be added and made available to Web Applications after the device has been launched.

This section defines the security requirements governing the installation and management of such Extension APIs.

6.3.2 REQUIREMENTS SUMMARY AND RATIONALE

This section is non-normative.

The method by which Extension APIs are provisioned is not currently in scope and support for such extensibility is not a requirement at this stage. However, for implementations that support dynamic extensibility, BONDI specifies certain minimum requirements so that the protection afforded by the BONDI security framework is preserved when Extension APIs are added.

The requirements in Sections 5 and 6 establish controls that govern the use of all APIs, including Extension APIs. In particular:

- All Features provided by Extension APIs must be explicitly declared as dependencies of a Web Application, and access to such Features must be granted by a Security Policy before those Features are made accessible to the requesting Application.
- When determining whether or not a given Web Application should be granted access to the Features of a dynamically deployed Extension API, the Security Policy has access to specific attributes of the signer of the Extension API implementation itself. This allows the Security Policy to restrict access to only those Extension APIs that are trusted at a level commensurate with the Web Applications that make use of them.
- All Device Capabilities exposed via those Features must themselves be accessible according to the Security Policy. The Web Runtime and Extension API implementation, together, must ensure that those Device Capabilities are subjected to access control checks.
- Updates to the Security Policy itself are controlled, including any Security Policy updates that pertain to use of an Extension API.

The remaining requirement, specified in this section, is to ensure that the use of an Extension API by a Web Runtime does not compromise the integrity or assurance of the Web Runtime in a way that risks its security mechanisms being circumvented.

The specific assurance requirements will vary from one Web Runtime implementation to another, and will depend largely on the nature of the security protection afforded by the underlying platform and operating system, and the implementation technology of the Extension API. Depending on these and other circumstances, the required assurance will be provided by an appropriate combination of technical, procedural and management measures. The measures range from requiring no assurance of the Extension API implementation (in the case that it executes in a fully managed execution environment such as a Java Virtual Machine), to requiring the same level of trust (assured with independent test, code review, and digital signatures) as is required for the Web Runtime implementation itself.

Therefore, rather than specify a specific set of assurance measures, BONDI requires that a conforming implementation, if dynamic extensibility is supported, must document the assurance requirements attached to Extension API implementations, must implement the technical measures that attach to the Web Runtime, and must document the assurance measures, technical and non-technical, that attach to components outside the Web Runtime itself.

6.3.3 **REQUIREMENTS**

This section is normative.

REQ. ID	FUNCTIONALITY	DESCRIPTION
AS-0800	If a Web Runtime supports the dynamic installation of Extension APIs, it SHALL define and implement measures to ensure that the introduction of those APIs does not compromise the integrity of the Web Runtime.	A combination of technical, procedural and management measures MAY be needed, depending on the nature of the Web Runtime and underlying platform, to ensure the integrity of the security enforcing mechanisms of the Web Runtime.
AS-0810	A Web Runtime that supports dynamic installation of Extension APIs SHALL define and enforce security mechanisms that limit the use of Extension APIs to appropriately authorised implementations.	The measures that attach to the Web Runtime itself MUST be defined, and implemented, but will in general depend on the Web Runtime and underlying platform.
AS-0820	A Web Runtime that supports dynamic installation of Extension APIs SHALL define any additional assurance requirements for Extension API implementations to ensure that its security enforcing mechanisms are not compromised by installation or use of such APIs.	The Web Runtime's own measures MAY, in general, be supplemented by assurance requirements on the Extension API implementations themselves. Again, the specific measures will depend on the Web Runtime and underlying platform.

7 DEFINITION OF TERMS

This section is normative.

TERM	DESCRIPTION
BROWSER	The user-facing application that provides the ability to navigate Websites. The Browser application uses the Web Engine to handle Website content. The Browser application typically provides user functionality to manage bookmarks or favourites, use the browsing history and adjust other settings.
DEVICE CAPABILITY	A Device Capability is a specific resource, or functionality of a device, that can be accessed, manipulated or exploited by a Web Application. Device Capabilities are defined and identified in a portable way, without a dependency on any specific JavaScript API, or on any underlying software platform or platform-specific API.
FEATURE	A Feature is is a set of JavaScript APIs and/or device behaviours that provide access to specified Device Capabilities. A Feature is identified uniquely by IRI, and is the unit of expression of dependencies by BONDI Web Applications.
FEATURESET	A set of Features, defined in terms of the IRIs of its constituent Features. A FeatureSet is identified uniquely by IRI.
JAVASCRIPT API	A JavaScript API is a program interface for Web Applications defined using an Interface Definition Language (IDL). JavaScript APIs are usually provided as a means for a Web Application to gain access to Device Capabilities. However, the definition of the API itself concerns the interfaces, methods, properties and other attributes that make up the API; the definition of the API is not necessarily associated with any specific Device Capabilities and, by itself, access to an API does not imply access to any underlying Device Capabilities.

TERM	DESCRIPTION
SECURITY POLICY	A collection of access control constraints, abstractly representable as a policy-set, as defined in the security policy model in this specification, that describes the circumstances under which Web Applications are permitted to access Features and underlying Device Capabilities.
SECURITY POLICY DOCUMENT	An XML document containing a BONDI <code><policy></code> or <code><policy-set></code> element as its root element. A Security Policy Document defines a Security Policy, or part of a Security Policy.
SIGNED SECURITY POLICY DOCUMENT	A Signed Policy Document is a XML file containing one or more Policy Documents and a detached XML Signature, as profiled in this specification, covering those Policy Documents.
WEB APPLICATION	The term used generically to refer to an application delivered using web technology, whether as a Website or a Widget.
WEB RUNTIME	The term used generically and collectively to refer to the components that are provided to execute Web Applications (i.e. including either a Widget User Agent, or Browser, or both).
WEB ENGINE	The component of an HTML Browser and of a Widget User Agent responsible for handling and rendering the HTML, CSS and JavaScript content. The Web Engine includes components for parsing, laying out and rendering HTML content, including the application of CSS. It also includes the JavaScript execution engine, and the client-side JavaScript binding to the HTML DOM.
WEBSITE	A remotely hosted collection of resources, authored in web formats (including HTML, JavaScript, CSS and various media formats) and served by a web server so as to be viewable in a Browser.
WIDGET	An interactive application for displaying and/or updating local data or data on the Web, packaged in a way to allow a single download and installation on a user's machine or mobile device.

TERM	DESCRIPTION
WIDGET CONFIGURATION DOCUMENT	A Widget Configuration Document is an XML document, belonging to a Widget Resource, that contains certain configuration information and other metadata relating to the Widget whose implementation is in the Widget Resource. A Widget Configuration Document has a <widget> element at its root that is in the W3C widget namespace.
WIDGET RESOURCE	A Widget Resource is a package, conforming to the W3c Widgets 1.0 Packaging and Configuration specification, containing the various files (including start files, configuration documents, icons, thumbnail, arbitrary files and digital signatures) that constitute the implementation of a Widget.
WIDGET USER AGENT	The collection of all components, over and above the Web Engine, needed to support installed Widgets. The functionality available varies between existing offerings but, broadly, the Widget User Agent is expected to be responsible for installation and de-installation of Widgets, and to provide the user-facing functionality for instantiation and configuration of those Widgets.

8 ABBREVIATIONS

ABBREVIATION	DESCRIPTION
API	Application Programming Interface
CRL	Certificate Revocation List
CSS	Cascading Style Sheets
DL	(OMA) Download
DM	(OMA) Device Management
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IRI	Internationalized Resource Identifier (see RFC 3987)
OCSP	Online Certificate Status Protocol
OMA	Open Mobile Alliance
OMTP	Open Mobile Terminal Platform
SCWS	Smartcard Web Server
TLS	Transport Layer Security
UI	User Interface
UICC	Universal Integrated Circuit Card
URI	Uniform Resource Identifier (see RFC 3986)
URL	Uniform Resource Locator (see RFC 3986)
URN	Uniform Resource Name (see RFC 2141)
W3C	World Wide Web Consortium
WR	Web Runtime
XACML	eXtensible Access Control Markup Language

ABBREVIATION	DESCRIPTION
XML	eXtensible Markup Language

9 REFERENCED DOCUMENTS

No.	DOCUMENT	AUTHOR	DATE
1	RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels	S Bradner	March 1997
2	BONDI v1.0 compliance specification http://bondi.omtp.org/1.1/compliance/BONDI_Compliance_Process_v1.1.pdf	OMTP	October 2009
3	Widgets 1.0: The Widget Landscape http://www.w3.org/TR/widgets-land/	W3C	April 2008
4	BONDI Interfaces v1.0 http://bondi.omtp.org/1.1/apis/BONDI_Interface_Requirements_v1.0.pdf	OMTP	May 2009
5	OMTP Application Security Framework v2.2 http://www.omtp.org/Publications/Display.aspx?Id=c4ee46b6-36ae-46ae-95e2-cfb164b758b5	OMTP	June 2008
6	BONDI Architecture & Security, Application Lifecycle Events Use Cases http://bondi.omtp.org/1.1/security/BONDI_Architecture_and_Security_Application_Lifecycle_v1.0.pdf	OMTP	May 2009
7	Widgets 1.0: Packaging and Configuration http://www.w3.org/TR/2009/CR-widgets-20091201/	W3C	July 2009

No.	DOCUMENT	AUTHOR	DATE
8	Widgets 1.0: Digital Signatures http://www.w3.org/TR/2009/CR-widgets-digsig-20090625/	W3C	June 2009
9	OMA Download V1.0 http://www.openmobilealliance.org/technical/release_program/download_v1_0.aspx	OMA	June 2004
10	Wireless Application Protocol WAP Certificate and CRL Profiles Specification http://www.openmobilealliance.org/technical/affiliates/wap/wap-211-wapcert-20010522-a.pdf	OMA	May 2001
11	XML Signature Syntax and Processing (Second Edition) http://www.w3.org/TR/xmlsig-core/	W3C	June 2008
12	Open Mobile Alliance: Online Certificate Status Protocol Mobile Profile specification http://www.openmobilealliance.org/Technical/release_program/docs/OCSP/V1_0-20070403-A/OMA-WAP-OCSP_MP-V1_0-20070403-A.pdf	OMA	April 2007
13	OASIS eXtensible Access Control Markup Language (XACML) Version 2.0	OASIS	February 2005

No.	DOCUMENT	AUTHOR	DATE
14	OMA Device Management V1.2 http://www.openmobilealliance.org/Technical/release_program/dm_v1_2.aspx	OMA	June 08

----- END OF DOCUMENT -----