

IEEE Standard Protocol for Stream Management in Media Client Devices

IEEE Computer Society

Sponsored by the
Storage Systems Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 2200™-2012

9 August 2012

IEEE Standard Protocol for Stream Management in Media Client Devices

Sponsor

Storage Systems Committee
of the
IEEE Computer Society

Approved 8 June 2012

IEEE-SA Standards Board

Abstract: Interfaces for intelligently distributing and replicating content over heterogeneous networks to portable and intermediate devices with local storage are defined.

Keywords: content caching, content queue, delayed delivery, device discovery download, IEEE 2200, intermediate devices, mobile content delivery network policy, seamless play

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2012 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 9 August 2012. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

3GPP™, LTE™, and UMTS™ are registered trademarks of ETSI.

Bluetooth is a registered trademark owned by Bluetooth SIG, Inc.

DLNA is a registered trademark of the Digital Living Network Alliance. Digital Living Network Alliance is a service mark of the Digital Living Network Alliance.

UPnP is a trademark of UPnP Forum™.

W3C is a registered trademark of the World Wide Web Consortium (registered in numerous countries). Marks of W3C are registered and held by its host institutions: Massachusetts Institute of Technology (MIT), European Research Consortium for Information and Mathematics (ERCIM), and Keio University, Japan.

PDF: ISBN 978-0-7381-7289-7 STD97264
Print: ISBN 978-0-7381-7395-5 STDPD97264

IEEE prohibits discrimination, harassment, and bullying. For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>. No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Notice and Disclaimer of Liability Concerning the Use of IEEE Documents: IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

Use of an IEEE Standard is wholly voluntary. IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon any IEEE Standard document.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained in its standards is free from patent infringement. IEEE Standards documents are supplied "AS IS."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

Translations: The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official Statements: A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on Standards: Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important to ensure that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. Any person who would like to participate in evaluating comments or revisions to an IEEE standard is welcome to join the relevant IEEE working group at <http://standards.ieee.org/develop/wg/>.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

Photocopies: Authorization to photocopy portions of any individual standard for internal or personal use is granted by The Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Notice to users

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at <http://standards.ieee.org/index.html> or contact the IEEE at the address listed previously. For more information about the IEEE Standards Association or the IEEE standards development process, visit IEEE-SA Website at <http://standards.ieee.org/index.html>.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/findstds/errata/index.html>. Users are encouraged to check this URL for errata periodically.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at <http://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this IEEE standard was completed, the High Quality Mobile Experience Working Group had the following membership:

David Koren, *Chair*
Minsoo Lee, *Vice Chair*
Yehuda Hahn, *Technical Editor*

Robert Blanchard
Mohammed Dadas
Don Eklund
William Fisher
Yehuda Hahn
Matt Hammond
Bob Hitching

Juhani Huttunen
Eero Jyske
Minsoo Lee
Charles Lo
Ram Madduluri
David McKeague
Joe Meza

Dan Newman
Kiyokazo Nozawa
Yoni Shternhell
Mitch Singer
Ariel Sobelman
Tom Tofigh
Jarek Wilkiewicz

Sponsoring Committee

John L. (Jack) Cole, *Chair*, Storage Systems Standards Committee

The following members of the entity balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

LG Electronics
Marvell Semiconductor, Inc.
Nokia

Qualcomm Incorporated
SanDisk Corporation

Sony Corporation
Synopsys, Inc.
Thought Technology Ltd.

When the IEEE-SA Standards Board approved this standard on 8 June 2012, it had the following membership:

Richard H. Hulett, *Chair*
John Kulick, *Vice Chair*
Robert M. Grow, *Past Chair*

Satish Aggarwal
Masayuki Ariyoshi
Peter Balma
William Bartley
Ted Burse
Clint Chaplin
Wael Diab
Jean-Philippe Faure

Alexander Gelman
Paul Houzé
Jim Hughes
Young Kyun Kim
Joseph L. Koepfinger*
David J. Law
Thomas Lee
Hung Ling

Oleg Logvinov
Ted Olsen
Gary Robinson
Jon Walter Rosdahl
Mike Seavey
Yatin Trivedi
Phil Winston
Yu Yuan

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Richard DeBlasio, *DOE Representative*
Michael Janezic, *NIST Representative*

Michelle Turner
IEEE Standards Program Manager, Document Development

Michael D. Kipness
IEEE Standards Program Manager, Technical Program Development

Matthew J. Ceglia
IEEE Client Services Manager, Professional Services

Introduction

This introduction is not part of IEEE Std 2200-2012, IEEE Standard Protocol for Stream Management in Media Client Devices.

This document describes a protocol and reference architecture that enables the deferred delivery of content to media client devices. The goal of this protocol is to improve user experience in the delivery of mobile content and reduce streaming congestion within wireless networks where deferred delivery of content is feasible. Content cached on devices is governed by content and network policies, which allow a content owner, network operator, and user to optimize the retention and consumption of the content.

Contents

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
2. Normative references.....	1
3. Definitions, acronyms, and abbreviations	4
3.1 Definitions	4
3.2 Acronyms and abbreviations	6
4. Functional requirements	8
4.1 General requirements.....	8
4.2 Requirements for queue management and networking.....	9
4.3 Requirements for cache usage	10
4.4 Requirements for Asset lifecycle.....	10
4.5 Security requirements	11
4.6 Superuser role requirements	11
4.7 Playlists.....	12
5. Reference architecture	12
5.1 General	12
5.2 Block diagram.....	12
6. Protocols and transports	13
6.1 Interface groups.....	13
6.2 Endpoints.....	13
6.3 Concurrency in objects	14
6.4 Available application programming interfaces	15
7. Queue objects	17
7.1 QueueRequest.....	17
7.2 QueueRequest states	19
7.3 RequestId.....	21
7.4 Sample flow	21
7.5 Application programming interface.....	22
7.6 Request manager interface.....	31
7.7 QueueRequest interface	53
8. Policy objects	65
8.1 Overview	65
8.2 XML structure	68
8.3 Rights checking	78
9. Cache objects.....	81
9.1 Overview	81
9.2 Application programming interface.....	83
9.3 Virtual storage device.....	89
9.4 ContentObject interface.....	112
9.5 Function groups	130

10. D Interfaces	142
10.1 Introduction	142
10.2 HTTP	143
10.3 Dynamic adaptive streaming over HTTP (DASH)	145
10.4 Generic multicast delivery	149
10.5 Local file	149
10.6 Playlists	149
11. Device capabilities discovery	153
11.1 Introduction	153
11.2 Device discovery via UPnP	153
11.3 Device capability exchange	156
11.4 Device description	158
11.5 XML Structure of device description	158
12. HTTP commands	164
12.1 Introduction	164
12.2 HTTP verbs and URL structure	165
12.3 Functions	165
12.4 Properties	167
12.5 Return values and error codes	168
12.6 Return value format	169
Annex A (informative) Use cases and requirements	170
Annex B (normative) Content profile	181
Annex C (informative) Right checking	183
Annex D (informative) Bibliography	184

IEEE Standard Protocol for Stream Management in Media Client Devices

IMPORTANT NOTICE: IEEE Standards documents are not intended to ensure safety, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IPR/disclaimers.html>.

1. Overview

1.1 Scope

This standard will define reference architectures and interfaces for intelligently routing and replicating content over heterogeneous networks to portable devices with local storage, without disrupting content providers’ direct relationship with end users.

1.2 Purpose

The purpose is to enable the delivery of richer media content to portable devices, in a way that is not limited by cost and bandwidth.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

3GPP™ TS 23.003, v10.4, Numbering, addressing, and identification, December 2011.^{1, 2}

3GPP TS 24.312, Access Network Discovery and Selection Function (ANDSF) Management Object (MO).

3GPP TS 25.331, v10.6, Radio Resource Control (RRC) Protocol specification, December 2011.

3GPP TS 26.346, Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs.

3GPP2 C.S0005-E, v3.0, Upper Layer (Layer 3) Signaling Standard for cdma2000 Spread Spectrum Systems, June 2011.³

3GPP2 C.S0016-D, v1.0, Over-the-Air Service Provisioning of Mobile Stations in Spread Spectrum Standards, January 2010.⁴

3GPP2 C.S0024-B, v3.0, cdma2000 High Rate Packet Data Air Interface Specification, September 2009.⁵

Common File Format 1.0.3, Digital Entertainment Content Ecosystem (DECE).⁶

IEC 62481-2, Digital living network alliance (DLNA®) home networked device interoperability guidelines—Part 2: DLNA media formats.^{7, 8}

IEEE Std 802.11™-2012, IEEE Standard for Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.^{9, 10}

IEEE Std 802.16e™-2005 and IEEE Std 802.16™-2004/Cor 1-2005, IEEE Standard for Local and metropolitan area networks—Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems—Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1.

IETF RFC 1738, Uniform Resource Locators.¹¹

IETF RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.

IETF RFC 2046, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.

IETF RFC 2231, MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations.

¹ 3GPP technical specifications are available at <http://www.3gpp.org/ftp/Specs/html-info/>.

² 3GPP is a registered trademark of ETSI.

³ Available from 3GPP2 at http://www.3gpp2.org/Public_html/specs/C.S0005-E_v3.0_cdma2000_1x_Layer-3_20110620.pdf.

⁴ Available from 3GPP2 at http://www.3gpp2.org/Public_html/specs/C.S0016-D%20v1.0_OTASP.pdf.

⁵ Available from 3GPP2 at http://www.3gpp2.org/Public_html/specs/C.S0024-B_v3.0_HRPD.pdf.

⁶ Available at <http://www.uvuu.com/uv-for-business.php>.

⁷ IEC publications are available from the International Electrotechnical Commission (<http://www.iec.ch/>). IEC publications are also available in the United States from the American National Standards Institute (<http://www.ansi.org/>).

⁸ DLNA is a registered trademark of the Digital Living Network Alliance. Digital Living Network Alliance is a service mark of the Digital Living Network Alliance.

⁹ IEEE publications are available from The Institute of Electrical and Electronics Engineers (<http://standards.ieee.org/>).

¹⁰ The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

¹¹ IETF RFCs are available from IETF at <https://datatracker.ietf.org/>.

IETF RFC 2616, Hypertext Transfer Protocol—HTTP/1.1.

IETF RFC 2818, HTTP Over TLS.

IETF RFC 3275, (Extensible Markup Language) XML-Signature Syntax and Processing.

IETF RFC 3926, FLUTE—File Delivery over Unidirectional Transport.

IETF RFC 4288, Media Type Specifications and Registration Procedures.

IETF RFC 4566, SDP: Session Description Protocol.

IETF RFC 4790, Internet Application Protocol Collation Registry.

IETF RFC 5052, Forward Error Correction (FEC) Building Block.

IETF RFC 5445, Basic Forward Error Correction (FEC) Schemes.

IETF RFC 5651, Layered Coding Transport (LCT) Building Block.

IETF RFC 5775, Asynchronous Layered Coding (ALC) Protocol Instantiation.

IETF RFC 6330, RaptorQ Forward Error Correction Scheme for Object Delivery.

ISO/IEC 23009-1, Information technology—Dynamic adaptive streaming over HTTP (DASH)—Part 1: Media presentation description and segment formats.¹²

ISO/IEC 29341-3-10:2008, Information Technology—UPnP™ Device Architecture—Part 3-10: Audio Video Device Control Protocol—Audio Video Transport Service.¹³

MovieLabs EMA Metadata, version 1.2.¹⁴

OMA Mobile Location Protocol.¹⁵

UPnP CM Service Template, UPnP ConfigurationManagement: 1, Version 1.01 for UPnP Version 1.0 (Standardized DCP), July 20, 2010.

UPnP MediaServer:4 Device, Standardized DCP.¹⁶

UPnP Device Architecture 1.1.¹⁷

W3C®, Cross-Origin Resource Sharing, Working Draft 3 April 2012.^{18, 19}

¹² ISO/IEC publications are available from the ISO Central Secretariat (<http://www.iso.org/>). ISO publications are also available in the United States from the American National Standards Institute (<http://www.ansi.org/>).

¹³ UPnP is a trademark of UPnP ForumSM.

¹⁴ Available at <http://www.movelabs.com/md/ema/v1.2/EMA%20Metadata%20v1.2.pdf>.

¹⁵ Available from Open Mobile Alliance at http://www.openmobilealliance.org/technical/release_program/mlp_v31.aspx.

¹⁶ Available from the UPnP Forum at <http://upnp.org/specs/av/UPnP-av-MediaServer-v4-Device.pdf>.

¹⁷ Available from the UPnP Forum at www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf.

¹⁸ Available at <http://www.w3.org/TR/cors/>.

¹⁹ W3C is a registered trademark of the World Wide Web Consortium (registered in numerous countries). Marks of W3C are registered and held by its host institutions: Massachusetts Institute of Technology (MIT), European Research Consortium for Information and Mathematics (ERCIM), and Keio University, Japan.

W3C Recommendation, Extensible Markup Language (XML) 1.0 (Fifth Edition).²⁰

W3C Recommendation, XML Schema Part 2: Datatypes, Second Edition, 28 October 2004.²¹

W3C Recommendation, XPath 1.0, XML Path Language (XPath) Version 1.0.²²

W3C Recommendation, XML Path Language (XPath) 2.0 (Second Edition).²³

W3C, Web IDL, Working Draft 19 April 2012.²⁴

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.²⁵

Where these terms refer to a specific part of the system described in this document, they are capitalized in the body of the specification.

application: *See:* **Local Application, User Agent.**

asset: Data represented by a single Universal Resource Locator (URL). Assets are addressed as single units and can be represented by files or URLs.

cache: A location in local, persistent storage dedicated to temporarily storing content.

client device: A physical hardware device, with at least one network interface and local storage. Typically, a Client Device is a mobile handset, tablet, or other general-purpose mobile device that can consume content.

command: A method invocation, directive in a header, or other statement that can be acted on by the receiving party.

content policy server: An Origin where the content provider sets the content distribution rules.

content profile: A configuration attribute describing characteristics of a consumable Asset.

content server: A server that hosts content that can be retrieved using suitable transport protocols for unicast and multicast delivery.

dynamic adaptive streaming over HTTP (DASH) client: Functional entity in the Client Device that processes metadata for the Asset in the form of the Media Presentation Description (MPD) to subsequently access the Media Segments comprising the Asset.

²⁰ Available at <http://www.w3.org/TR/2008/REC-xml-20081126/>.

²¹ Available at <http://www.w3.org/TR/xmlschema-2/>.

²² Available at <http://www.w3.org/TR/xpath/>.

²³ Available at <http://www.w3.org/TR/xpath20/>.

²⁴ Available from W3C at <http://www.w3.org/TR/2010/WD-WebIDL-20101021/>.

²⁵ *IEEE Standards Dictionary Online* subscription is available at http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.

intermediate device: A dual role client/server device on a network that can be used to stage Assets destined for a Client Device, and may be able to consume content directly. The Intermediate Device typically holds Assets temporarily until they can be delivered to the Client Device and does not consume them directly.

local application: A software application resident on the Client Device that can communicate with the queue/policy engine (QPE) using the Q2 interface protocol.

local application origin: An Origin that identifies a local process on the same device that executes the queue/policy engine (QPE).

local file system: The native file system in the Client Device.

network interface: A point of interconnection between a computer and a private or public network. This includes but is not limited to wired Ethernet interfaces, IEEE 802.11x™ wireless interfaces, and 3G/4G interfaces.

origin: Local Application or Content Server.

NOTE—*Origins* are described in 9.5.2.3.²⁶

player: A software or hardware subsystem on the Client Device that consumes Assets. The Player is generally responsible for rights management [using an associated Digital Rights Management (DRM) client], rendering, and any other consumptive activity related to a content Asset. Players may be incorporated in Local Applications.

playlist: A description of a sequence of Assets, which can be grouped together for delivery to a Player.

policy: A set of properties that can be assigned to one or more Queue Requests and Assets. The Policy includes properties that specify restrictions and parameters that affect the execution of the Queue Request as well as the storage and retention of Assets. There are two types of Policies defined in this specification: Network Policy and Content Provider Policy.

policy client: A subsystem of the queue/policy engine (QPE) that maintains the Policy objects. The Policy Client retrieves policies from the Policy Server and adjusts Queue Request behavior. In conjunction with Policy information, the Policy Client may acquire other key data such as local operating conditions (radio interface characteristics, quality of IP connectivity, power levels, application-specific requirements, etc.) and user preferences and use these inputs in making the access connectivity decision. It may also communicate such connection-related information to the Network Policy Servers. The Policy Client may also provide Policy information to Applications via the QPE.

policy server: An Origin that communicates with the queue/policy engine (QPE) and sets information in Policy objects. Policy Servers may be Content Policy Servers or Network Policy Servers.

profile-id: A user or user group specific identifier, optional for use in multicast delivery, to enable predictive download of Assets, by utilizing content server-resident knowledge about a user's content preferences. The Profile-ID represents a simple, opaque, alias that abstracts an individual user's content preferences, which may be determined by prior Asset retrieval/consumption history, preference settings, other usage behaviors, and/or heuristics. For a given end user, its value is known to the associated Client Device and the Content Server. Profile-ID, when carried in service announcement information, is tagged to one or more content items scheduled for multicast delivery, and intended for subsequent download and caching of that content item or Asset by the matching Client Device.

²⁶ Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

queue/policy engine (QPE): A software process on the Client Device that instantiates, at a minimum, the P1, S, D1, D2, and Q2 interface protocols. The QPE is responsible for maintaining queues on behalf of each Local Application and Content Server, interfacing with storage, and synchronizing Queue Requests with Policy.

queue request: A Command that can be sent via the Q2, Q3, or Q4 interfaces to the queue/policy engine (QPE) or Intermediate Device that identifies an Asset to queue for download.

remote origin: An Origin that is reachable through a network connection.

superuser: An Origin that is assigned privileges and rights beyond those assigned to all other Origins. Superuser represents the owner/user of the physical Client Device.

superuser application: A Local Application on the Client or Intermediate Device that manages global Policy and queue management on behalf of the user, using an Origin designated as the Superuser..

user agent: A software application—such as a Web browser or middleware—on the Client Device that can render and execute server-supplied applications.

virtual storage device (VSD): A representation of local storage accessible via the cache objects.

3.2 Acronyms and abbreviations

ACL	access control list
ALC	Asynchronous Layered Coding
ANDSF	Access Network Discovery and Selection Function
API	application programming interface
CDS	Content Directory Service
CFF	Common File Format
CORS	Cross-Origin Resource Sharing
CTL	Certificate Trust List
DASH	Dynamic Adaptive Streaming over HTTP
DCP	Device Control Protocol
DECE	Digital Entertainment Content Ecosystem
DLNA	Digital Lifestyle Networking Alliance
DM	device management
DMC	digital media controller
DMP	digital media player
DMR	digital media renderer

DMS	digital media server
DRM	digital rights management
E-UTRA	Evolved Universal Terrestrial Radio Access
EDR	Enhanced Data Rate
EMA	Entertainment Merchants Association
eMBMS	Evolved Multimedia Broadcast/Multicast Service
ESG	electronic service guide
FDT	file delivery table
FEC	forward error correction
FLUTE	File Delivery over Unidirectional Transport
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HTTP	Hypertext Transfer Protocol
HTTP/S	Secure Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
I/O	input/output
IP	Internet Protocol
LAN	local area network
LTE™	Long Term Evolution ²⁷
MBMS	Multimedia Broadcast Multicast Service
MIME	Multipurpose Internet Mail Extensions
MLP	Mobile Location Protocol
MPAA	Motion Picture Association of America
MPD	Media Presentation Description
OMA	Open Mobile Alliance
QPE	queue/policy engine

²⁷ LTE is a registered trademark of ETSI.

RDF	Resource Description Framework
RFC	request for comments
SD	Standard Definition
SDP	Session Description Protocol
SSDP	Simple Service Discovery Protocol
TLS	Transport Layer Security
TOI	transmission object identifier
UDP	Unreliable Datagram Protocol
UMTS™	Universal Mobile Telecommunications System ²⁸
UPnP	Universal Plug and Play
URI	Universal Resource Identifier
URL	Universal Resource Locator
USB	Universal Serial Bus
UTRAN	Universal Terrestrial Radio Access Network
VSD	virtual storage device
WebIDL	Web Interface Description Language
WLAN	wireless local area network
XML	Extensible Markup Language

4. Functional requirements

4.1 General requirements

This clause describes the requirements that drive the definition of IEEE Std 2200-2012. This clause references the interface definitions defined in Clause 6.

All Commands required in the Policy (P) and Queue Request (Q) interfaces shall require only a Client Device that includes at least one network interface, local storage, and local processing capabilities. The Q and P interface protocols may include optional commands that require additional hardware such as a display, GPS, security coprocessor, or multiple network interfaces.

Client Devices may support the P2 interface. If the Client Device supports the P2 interface, it should use platform-supplied interfaces such as an Open Mobile Alliance (OMA) device management (DM) Client as

²⁸ UMTS is a registered trademark of ETSI.

defined in the OMA Device Management Protocol [B10].²⁹ However, Client Devices that do not support the P2 interface shall still implement Content Policy via the P1 interface.

4.2 Requirements for queue management and networking

Reception of requested Assets managed by the queue shall be accomplished using unicast download, multicast download, or a combination of the two mechanisms, subject to the capabilities of the Queue/Policy Engine (QPE) and Client Device.

Local Applications and User Agents may be active when a download is started or completed. Processing of Queue Requests by the QPE shall not require the initiating Origin (which submitted a Queue Request via the Q2 or Q3 interface) to be active. During a standard unicast download procedure, no synchronous calls to the initiating application or User Agent may be required. However, a renegotiated access token may be used for unicast downloads. If the Q2 interface is used to initiate the Queue Request, it may also specify a callback that will be invoked to perform setup at the time of the download. However, sensitive personally identifiable information shall not be stored with a Queue Request.

Queue is centralized and may be subject to device-side and server-side overrides. The QPE shall maintain a single queue, although Commands defined in the Q interfaces may change the priority and order of Queue Requests. Commands may be expressed using the Q2 and P1 interface protocols that affect the entire queue, although such commands shall only affect the scope to which the calling origin is permitted to affect. Multicast delivery may be triggered at a fixed time and may affect queue service.

Negotiation to acquire Assets shall be done externally from the queue management system, either by resolving a unique content Uniform Resource Locator (URL) in advance or by calling the application to do pre-download negotiation at the time of acquisition. Queue Requests submitted via the Q2 interface protocol shall include either a complete URL that can be called from the context of the User Agent or calling Local Application (including any locally added request headers) to download an Asset, or include a local URL that invokes the calling Local Application in order to renegotiate the download. The User Agent shall add request headers as if the download was initiated by the user from the User Agent, as well as request headers that indicate details of the QPE, Queue Request, and effective Policy.

Queue Requests may specify a delivery mechanism that includes either unicast or multicast delivery. Requirements for unicast and multicast delivery methods are specified in 4.2.1 and 4.2.2, respectively.

4.2.1 Unicast delivery

Unicast delivery shall be over Hypertext Transfer Protocol (HTTP) or Secure HTTP (HTTP/S) transport of the content format as defined in ISO/IEC 23009-1, for deferred consumption or live streaming, using HTTP progressive download, or any other method supported by the Client Device and HTTP server.³⁰

NOTE—Content delivery is generally over a downloadable protocol rather than a streaming protocol.

4.2.2 Multicast delivery

Queue Requests shall specify one or more specific times at which they should be executed, to accommodate fixed scheduling of multicast delivery. If supported by the Client Device, the QPE shall be active during the time interval(s) selected by the queue management system to download Assets using multicast delivery. The fixed delivery schedule is provided by service announcement/discovery information, for example as carried in the *electronic service guide* (ESG). The QPE shall support the

²⁹ The numbers in brackets correspond to those of the bibliography in Annex D.

³⁰ Information on references can be found in Clause 2.

transport protocols and codecs, including Application Layer Forward Error Correction (FEC), as specified by the associated multicast technology.

In this specification, multicast delivery is assumed to be provided by a wireless broadcast distribution system, such as 3GPP Evolved Multimedia Broadcast/Multicast Service (eMBMS) as described in 3GPP TS 26.346.

4.3 Requirements for cache usage

Asset caching is opaque to Local Applications. Applications shall only use IEEE 2200 *application programming interfaces* (APIs) or interfaces to access the virtual storage device (VSD). The cache objects defined in Clause 9 shall be the only supported method used to access the cache storage.

Client Devices are not required to maintain a permanent copy of cached content. Assets stored in the cache shall be associated with the Origin that can be used to retrieve them. Users, servers, and the cache itself may delete Assets from the cache at any time, while retaining the metadata that allows the retrieval of the deleted Asset if still available at the Origin. A Queue Request shall optionally include a URL that can be used to request Assets again if the content gets deleted.

The QPE shall enforce cache controls sent by a Content Server.

The QPE shall allow staging of Assets into and out of the cache, but shall not allow direct write access to bits stored in the cache by Remote Origins via the Q2 and Q3 interfaces.

Content may be scheduled for deletion if it is not used based on application or server driven Policy. Assets may be deleted to make more room for other Assets, provided that no part of an Asset is deleted unless the entire Asset is deleted. Cached Assets may include a Policy that includes the maximum time to live. The cache may autonomously delete Assets after the time to live of the Asset has expired. In addition, Users and Servers may request deletion of a specific Asset using the Cache objects and the Q2/Q3 interface protocol.

4.4 Requirements for Asset lifecycle

Assets shall have a unique, addressable identifier assigned at the time of creation. The identifier shall be in the form of a Universal Resource Identifier (URI). The Asset URI shall be used to identify the Asset in the Q, D, P, and S interfaces. There shall be a Command in the S protocol that translates the Asset URI to an origin URI that can be used to retrieve the Asset again if it is deleted. The Origin URI shall be specified in the Queue Request. The retained URI may be different from the one actually used to retrieve the Asset if a one-time URI is used.

Removable storage used for caching might not be portable between devices. While removable storage used for caching could be portable between devices, this is not required.

Content may be capacity managed and may expire. However, an application shall be able to control these functions for Assets requested by that application. The Cache shall include Commands that—when invoked via the Q interfaces—allow a calling Origin to specify an expiration and capacity management Policy for Assets retrieved from that origin. This assumes that the desired control to be asserted by the Origin does not conflict with related Content Policy.

4.5 Security requirements

Origins shall be sandboxed from each other. Unless requested by the creating Origin, no Origin shall have access to Assets retrieved from a different Origin. A defined Superuser Origin shall have access to all Assets. No Origin other than the Superuser Origin shall have access to Assets retrieved from a different Origin.

Web sites shall be treated as Local Applications for the purpose of application sandboxing. The User Agent itself is not considered the Origin for permission purposes. Queue requests shall not be accepted and processed by the QPE in violation of Superuser policy. The QPE shall not accept a request from an Origin unless the identity of the Origin can be verified by the QPE.

Assets may be marked as visible to all applications or private to a specific Origin or group of Origins. Access rights to the Asset shall be assigned by the origin that the Asset was retrieved from. The Cache shall support association of access rights to Assets on a user, group, or world basis.

IEEE Std 2200-2012 is not intended to replace Digital Rights Management (DRM) functionality. However, implementations shall not interfere with, block, or otherwise reduce the experience enabled in DRM-protected Assets. VSDs are not required to provide security that prevents Local Applications from directly accessing Assets represented in storage, although compliant implementations may provide such security. No aspect of the environment shall be operable to prevent the full expression of digital rights management. No component in IEEE Std 2200-2012 shall bypass security in digital rights management or conditional access systems. No protocol in IEEE Std 2200-2012 shall override rights and enforcement specified in any digital rights management. Keys used for digital rights management shall not be directly interpreted or handled by IEEE Std 2200-2012.

A rights check shall be performed before transitioning to the ACTIVE state on the retrieving device, if the Content Policy specifies that a rights check shall be performed. Client Devices may report that they do not support rights checking. Intermediate devices shall not deliver content to Client Devices that report that they do not support rights checking if the rights check Content Policy is set by the Content Policy Server.

Some Assets may be downloaded by one Origin but consumed by other Origins. This includes advertising caches, music and video subscription servers, and Intermediate Devices. Origins shall have the ability, via the Q interfaces, to specify additional Origins that certain permissions may be delegated to. This feature shall not delegate rights associated with digital rights management systems, nor shall it enable access to plaintext content that is protected by digital rights management. However, encrypted data objects protected by digital rights management may be shared or delegated, provided that the key objects are not cached in the VSD system in plaintext form.

4.6 Superuser role requirements

The QPE implementation may include a privileged (Superuser) Origin, possibly integrated with the device. This Origin shall be uniquely identified. The QPE shall include a mechanism to authenticate and authorize the Superuser Origin. The Superuser shall use the Q2 interface for all privileged commands not available to other Origins.

The Superuser may specify global Policy, which shall override other application-level policies. The Superuser-specified Policy (specified using the Policy objects) shall override any other Policy specified by a Local Application origin. The global Policy may include bandwidth and power management.

The Superuser may prioritize Origins in the queue. The Q2 interface protocol shall include Commands that allow the Superuser to change the priority of Queue Requests either individually or based on calling Origin.

The Superuser shall have the ability to manage the cache. The Q2 interface protocol shall include Commands that allow the Superuser to select and tune Virtual Storage Device properties.

The Superuser shall also assume responsibility for user experience and alerting within the Client Device. The Superuser shall have access, via the Q2 interface protocol, to APIs that allow querying of global status of the Client Device, as well as APIs that allow change in queue, cache, and policy management.

The Superuser shall have an interface and privileges needed to delete content in the cache. The Q2 interface protocol shall include access to functions and methods by which a Superuser application can view the properties of Assets in the cache and can delete cached Assets.

Intermediate Device Superuser Applications shall not be assigned the Superuser role on Client Devices that retrieve Assets from the Intermediate Device.

4.7 Playlists

A Playlist represents a manifest file describing the contents and the access methods available to a HTTP/HTTPS Client or a dynamic adaptive streaming over HTTP (DASH) Client. Playlists may include a mechanism for verifying the authenticity of delivered Playlist, when delivered over the D1 interface. If a mechanism is specified, the QPE shall not modify the sequence or other Content Policies defined in the Playlist and shall deliver the content over the D2 interface using the sequence defined in the Playlist.

Playlists may be associated with a Content Policy. The policy may include Assets that shall be delivered over the D2 interface in a specific sequence, in which byte-range reading may be restricted to a predefined sequence, or in which content will only be delivered to specific identifiable Players or Players with specific capabilities (such as DRM). Content Policies delivered in a Playlist shall remain associated with the Playlist and the QPE shall not deliver content over the D2 interface in violation of the policy. Superuser policy shall not grant permissions beyond the policy delivered in a Playlist.

5. Reference architecture

5.1 General

The reference architecture defines the logical blocks that are referred to in IEEE Std 2200-2012. The behavior of the components and the protocols linking them are defined in other clauses within this document.

5.2 Block diagram

In Figure 1, dashed lines represent external components whose behavior is not defined in this standard. Solid lines represent components fully defined in this standard.

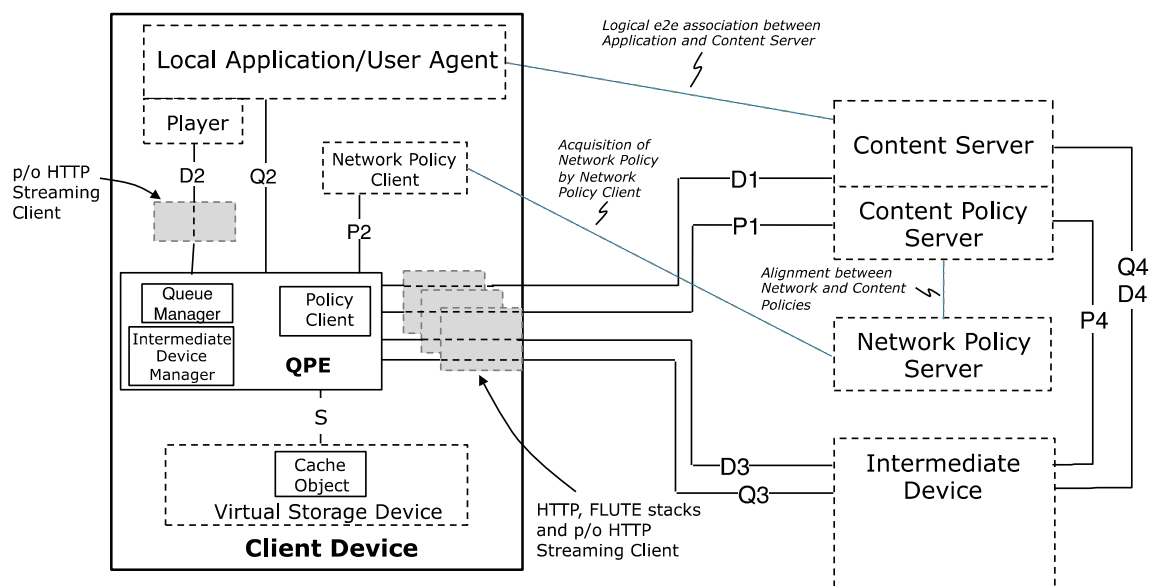


Figure 1—Reference architecture block diagram

6. Protocols and transports

6.1 Interface groups

IEEE Std 2200-2012 defines four interface groups. The **P** interface group defines Policy and links between the QPE and Policy Servers. The **Q** interface group defines Queue Request handling and is the primary command channel, associating between the Content Server, Intermediate Devices, and the QPE. The **Q** interface group also allows caching functionality to be invoked by Local Applications and User Agents. The **S** interface abstracts storage and cache capabilities to the QPE. The **D** interface group is used for data transport.

The S and D2 interfaces (see Table 1 for the definition of the D2 interface) may transform data from physical storage. DRM enforcement, FEC decoding, and other transformations may be performed during write and read of data via the S interface or when streaming data out via the D2 interface.

6.2 Endpoints

The interfaces are further defined by their endpoints. Table 1 defines the interfaces, their endpoints, and the protocols used. In this table, a server is defined as the endpoint that listens for incoming connections, and the client is defined as the endpoint that initiates the connection to the server.

Table 1—Interface endpoints

Interface	Server	Client	Transport protocol	Notes
D1	Content Server	QPE	As defined in Clause 10	
P1	Content Policy Server	QPE	HTTP/1.1 or HTTPS/1.1	
D2	QPE	Player	As defined in 10.2.2	
Q2	QPE	Application	HTTP/1.1 or HTTPS/1.1 or an implementation-specific transport	Local applications may use an implementation-specific transport if supported by the Client.
P2	Network Policy Client	QPE	Implementation-specific inter-process (IPC) transport	
D3	Intermediate Device	QPE	HTTP/1.1 or HTTPS/1.1	
Q3	QPE	Intermediate Device	UPnP/HTTP 1.1 as defined in Clause 10	
D4	Content Server	Intermediate Device	HTTP/1.1 or HTTPS/1.1	
P4	Content Server	Intermediate Device	HTTP/1.1 or HTTPS/1.1	Only Policy XML structures may be downloaded in this manner.
Q4	Content Server	Intermediate Device	HTTP/1.1 or HTTPS/1.1	Q4 transactions may be initiated by the Content Server via an external trigger not defined in this specification.
S	Virtual Storage Device	QPE	Implementation-specific transport	

Where a function is described using Web Interface Description Language (WebIDL), and the transport interface (Qx, Px, or S) supports a specific language, language bindings shall be as defined in the W3C Web IDL specification. Where an implementation-specific transport is used, the underlying invocation methods shall also be implementation-specific and are not defined in this specification.

Remote origins using the Q3 and P4 interfaces may use XML forms of Queue Request and Policy structures. If enabled via the device capabilities discovery function, a QPE may signal to an Intermediate Device or an Intermediate Device may signal to a Content Server the mechanism for invoking commands.

6.3 Concurrency in objects

Multiple Origins may simultaneously invoke methods, which may modify data. When invoking remote calls via the Q3 or P4 interfaces, the underlying objects being addressed may not be consistent from call to call.

To reduce concurrency errors, the following assumptions are made in the API definitions in Clause 7 and Clause 9:

- a) Any instance of an object is accessed only by a single thread at a given time. Calling applications should implement locks to prevent multiple concurrent calls.

- b) Attributes that may modified by calls, such as the responseCode attribute, are guaranteed not to be modified only until the next method invocation by the calling Origin.
- c) If multiple Origins instantiate an object, a different instance is assigned to each one. Method invocations that change the underlying state of an object (such as state changes on a QueueRequest or cached Asset modifications) will be controlled by an internal mutex (mutual exclusion lock) within the Client such that while one change is executed, the other method invocations are suspended.
- d) RequestIds and storageIds are guaranteed to be valid only for the Origin that retrieved them. Sharing IDs is not supported between different Origins.
- e) If a method is invoked on stale data, a relevant error code will be returned to the calling Origin.
- f) All method invocations are blocking and synchronous. The calling thread does not resume until the method invoked either completes or fails.

6.4 Available application programming interfaces

Table 2 lists all of the APIs defined in IEEE Std 2200-2012 and the valid interfaces for each. APIs called from an invalid interface shall return the ERR_NOT_SUPPORTED error code. As a general rule, methods that access or manipulate QueueRequests may be called from the Q interfaces, methods that affect policy may be called from the Q or P interfaces, and methods that relate to the cache may be called from the Q, D, or S interface groups.

It should be noted that even if an interface is valid, the method may still fail with ERR_NOT_SUPPORTED. This may occur, for example, if the Playlist method is called on a VSD without the PLAYLIST function group. Permission errors may also cause a method to fail even on a supported interface.

The API names are listed using the WebIDL name where one is defined and the URI where there is no WebIDL equivalent. For HTTP URIs corresponding to each WebIDL method name, see the corresponding method definition.

In Table 2, an interface marked with V denotes a valid and permitted invocation of the corresponding API.

Table 2—API/interface mappings

	Q2	Q3	Q4	P1	P2	P4	D1	D2	D4	S
RequestManager										
requestCount	V	V	V	—	—	—	—	—	—	—
getRequestIds	V	V	V	—	—	—	—	—	—	—
getRequest	V	V	V	—	—	—	—	—	—	—
submitRequest	V	V	V	—	—	—	—	—	—	—
cancelRequest	V	V	V	—	—	—	—	—	—	—
suspendRequest	V	V	V	—	—	—	—	—	—	—
resumeRequest	V	V	V	—	—	—	—	—	—	—
getProgress	V	V	V	—	—	—	—	—	—	—
getState	V	V	V	—	—	—	—	—	—	—
getPriority	V	V	V	—	—	—	—	—	—	—
setPriority	V	V	V	—	—	—	—	—	—	—
getEffectivePolicy	V	V	V	V	V	V	—	—	—	—
setGlobalPolicy	V	—	—	V	V	V	—	—	—	—
setDebugOutput	V	—	—	—	—	—	—	—	—	—

Table 2—API/interface mappings (continued)

	Q2	Q3	Q4	P1	P2	P4	D1	D2	D4	S
QueueRequest										
getRequestId	V	—	—	—	—	—	—	—	—	—
getState	V	V	V	—	—	—	—	—	—	—
getProgress	V	V	V	—	—	—	—	—	—	—
getProperty	V	V	V	—	—	—	—	—	—	—
setProperty	V	V	V	—	—	—	—	—	—	—
getPropertyType	V	V	V	—	—	—	—	—	—	—
removeProperty	V	V	V	—	—	—	—	—	—	—
StorageManager										
VSDCount	V	V	V	—	—	—	V	V	V	V
getStorageIds	V	V	V	—	—	—	V	V	V	V
getStorage	V	—	—	—	—	—	—	—	—	V
getFunctionGroups	V	V	V	—	—	—	—	—	—	V
VSD										
storageId	V	V	V	—	—	—	V	V	V	V
name	V	V	V	—	—	—	V	V	V	V
functionGroups	V	V	V	—	—	—	V	V	V	V
allObjects	V	V	V	—	—	—	V	V	V	V
getObject	V	V	V	—	—	—	V	V	V	V
createObject	V	V	V	—	—	—	V	—	V	V
removeObject	V	V	V	—	—	—	V	—	V	V
getPropertyKeys	V	V	V	—	—	—	V	V	V	V
getProperty	V	V	V	—	—	—	V	V	V	V
getPropertyType	V	V	V	—	—	—	V	V	V	V
setProperty	V	V	V	—	—	—	V	V	V	V
removeProperty	V	V	V	—	—	—	V	V	V	V
findObjects	V	V	V	—	—	—	V	V	V	V
getPlaylistObjects	V	V	V	—	—	—	V	V	V	V
createPlaylist	V	V	V	—	—	—	V	—	V	V
removePlaylist	V	V	V	—	—	—	V	—	V	V
createObjectInPlaylist	V	V	V	—	—	—	V	—	V	V
issueCommand	V	V	V	—	—	—	V	—	V	V
getCommandStatus	V	V	V	—	—	—	V	—	V	V
ContentObject										
properties	V	V	V	—	—	—	V	V	V	V
getProperty	V	V	V	—	—	—	V	V	V	V
getPropertyType	V	V	V	—	—	—	V	V	V	V
setProperty	V	V	V	—	—	—	V	V	V	V
removeProperty	V	V	V	—	—	—	V	V	V	V
size	V	V	V	—	—	—	V	V	V	V
open	V	—	—	—	—	—	—	—	—	V
close	V	—	—	—	—	—	—	—	—	V
remove	V	V	V	—	—	—	V	—	V	V
read	V	V	V	—	—	—	V	V	V	V
write	V	V	V	—	—	—	V	—	V	V
seek	V	—	—	—	—	—	—	—	—	V
tell	V	—	—	—	—	—	—	—	—	V
getStreamingUri	V	—	—	—	—	—	—	—	—	V
/CurrentPlay	—	V	—	—	—	—	—	V	V	—
/Device/Description	—	V	V	—	—	—	—	V	V	—
/Device/AuthorizeCORS	—	V	—	—	—	—	—	V	V	—
/Device/Cached	—	V	—	—	—	—	—	V	V	—

7. Queue objects

7.1 QueueRequest

7.1.1 General

A QueueRequest object is a mechanism for queuing Assets for later download. QueueRequests are sent over the Q2 and Q3 interfaces to the QPE by Local Applications, Intermediate Devices, and Content Servers (via User Agents).

A QueueRequest contains information about what, where, when, and how Assets are to be transferred between a client and server through a network. QueueRequests contain metadata describing the content to be transferred as well as optional metadata that further describes the content to be transferred. QueueRequests may also include additional metadata describing the Asset, which shall be persistently stored with the Asset.

QueueRequests include the URI from which to obtain the Asset. The URI shall be used to retrieve the Asset via the D interfaces. Content and network policies may be referenced in a QueueRequest.

For a request to be valid, it shall contain all mandatory properties, using the structure defined in 7.5.4. This standard defines a set of predefined property keys in 7.5.5, 7.5.6, and 7.5.7.

7.1.2 Request life cycle

Once submitted, a valid QueueRequest may be in a state as defined in 7.2. In order for a request to transition to the *QUEUED* state, the QueueRequest shall first be validated and if free from errors, shall be assigned a RequestId. An invalid QueueRequest shall not be placed in the queue and shall not be assigned a RequestId.

The QueueRequest may have a Policy (see Clause 8) that establishes the criteria or rules that further constrain how and when the QueueRequest shall be processed. If no Policy was specified, the behavior for download and cache shall always be as if the Policy was set to a constant of true.

Depending on changes in the state of the Client Device and the rules in the Policy, the requests in the queue shall be evaluated to determine if a request should be moved into a new state. Depending on the request's Policy, and the current operating condition of the Client Device, a newly submitted *QUEUED* request shall be moved into *BLOCKED*, *WAITING*, or *ACTIVE* state (see Figure 2).

A QueueRequest shall be transitioned to the *BLOCKED* state if the current operating conditions of the Client Device do not meet the criteria defined by the request's Policy. A QueueRequest shall be transitioned to the *WAITING* state if the current operating conditions of the Client Device meet the request's criteria; however, other higher priority requests are being processed ahead of the *WAITING* request. A QueueRequest shall be transitioned to the *ACTIVE* state if the current operating conditions of the Client Device satisfy the request's Policy, and the request is the highest priority request for download. An implementation of this specification may support multiple *ACTIVE* QueueRequests simultaneously. The total number of requests that may be processed shall be reported in the Client's deviceInfo structure, as described in 11.5.

A request in the *ACTIVE* state may transition to the *WAITING* state if a request of higher priority requires processing. Similarly, a request in the *ACTIVE* state may transition to the *BLOCKED* state if the constraints defined in the request's Policy are no longer met by the current operating condition of the Client Device, or

if the request's execution triggers an error that requires application interaction (such as an HTTP 402 Payment Required status code). If one request is assigned a mandatory time for execution, and there are no constraints in the Policy or Client Device environment that prevent the execution of the request, it shall transition to the *ACTIVE* state at that time. If multiple requests are assigned a mandatory time for execution, and the Client Device cannot serve all of them due to client-side limitations, the highest-priority requests shall transition to the *ACTIVE* state, pausing requests that do not have an assigned time. Requests assigned a specific time that cannot be served due to overriding higher-priority time-critical requests shall transition to the *BLOCKED* state. Requests paused due to the execution of a request that has a mandatory time for execution shall be transitioned to the *WAITING* state.

A request may be placed in the *BLOCKED* state if an unresolved error condition occurs while in the *ACTIVE* state. In addition, an application may transition a request into the *SUSPENDED* state by explicitly suspending it. A request in the *SUSPENDED* or *BLOCKED* states can be resumed or re-queued by an application by explicitly resuming a request after addressing any errors associated with it.

During any state, except the *COMPLETED* state, a *QueueRequest* may be cancelled. If a request is in the *ACTIVE* state when a request to cancel is made, the request shall first be placed in the *SUSPENDED* state prior to being canceled.

A *QueueRequest* shall transition to the *COMPLETED* state when all content to be transferred by the request has been sent or received.

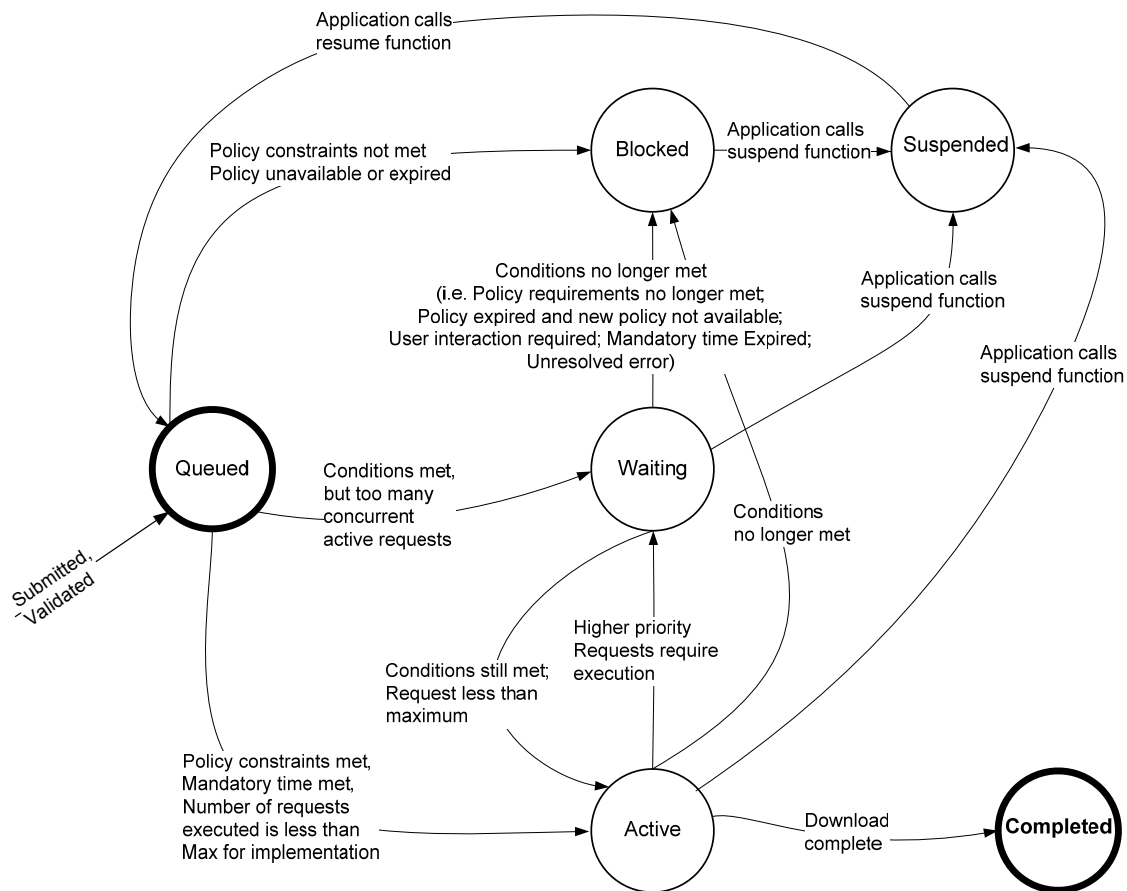


Figure 2—Overall state diagram

7.2 QueueRequest states

The following data structure defines the valid states for a QueueRequest:

```
WebIDL:
interface QueueRequestState {
  const unsigned short UNDEFINED = 0;
  const unsigned short QUEUED = 1;
  const unsigned short ACTIVE = 2;
  const unsigned short BLOCKED = 3;
  const unsigned short WAITING = 4;
  const unsigned short COMPLETED = 5;
  const unsigned short SUSPENDED = 6;
}
```

7.2.1 QUEUED state

In the QUEUED state, the QueueRequest has been submitted, validated to be free from errors, and placed on an internal queue. When a QueueRequest is first submitted it is set in the QUEUED state. The QueueRequest will not transition out of the QUEUED state if the policy is unavailable or expired. From the QUEUED state, a QueueRequest may enter the ACTIVE, BLOCKED, or WAITING states.

7.2.2 ACTIVE state

A QueueRequest enters the ACTIVE state when the content is actively being transferred. To enter this state, all the criteria imposed on the Request shall be satisfied, and the Request shall have the highest priority over all other submitted Requests or be triggered due to a time-specific schedule. If multiple requests have the same priority and that priority is higher than all other requests in the WAITING state, the number of requests and the order in which they are served shall be QPE-specific. From the ACTIVE state, a QueueRequest may enter the BLOCKED, WAITING, COMPLETED, or SUSPENDED state.

While a QueueRequest is in the ACTIVE state, it may allocate space in the VSD corresponding to the expected size of the Asset when it completes download.

Figure 3 illustrates the ACTIVE state.

7.2.3 BLOCKED state

A QueueRequest enters the BLOCKED state when criteria required for the QueueRequest have not been met, or if the Policy associated with the QueueRequest is unavailable or expired. As an example, if a QueueRequest requires a wireless connection, but one is not available, the QueueRequest is considered blocked.

If the QueueRequest includes a mandatory time for execution, and at that time it is not possible to serve the request, it shall enter the BLOCKED state. In addition, if an error condition occurs while executing a QueueRequest, it shall enter the BLOCKED state.

From the BLOCKED state, a QueueRequest may enter the ACTIVE or WAITING state. A QueueRequest may also enter the SUSPENDED state if an application explicitly suspends the request.

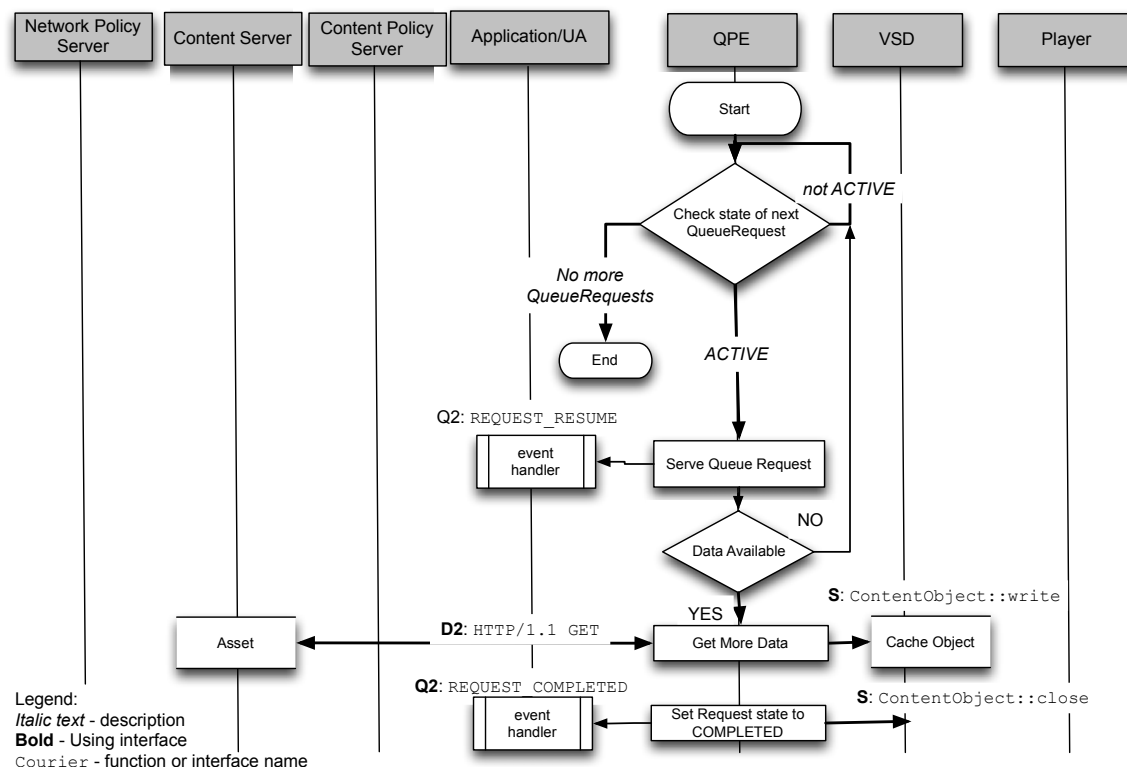


Figure 3—Active state process

7.2.4 WAITING state

A QueueRequest enters the WAITING state when all the criteria required for the QueueRequest has been satisfied, however, at least one other QueueRequest of the same or higher priority is currently ACTIVE and the QPE cannot activate this request at this time. From the WAITING state, a QueueRequest may enter the ACTIVE, SUSPENDED, or BLOCKED state. A QueueRequest may also enter the SUSPENDED state if an application explicitly suspends the request.

7.2.5 COMPLETED state

A QueueRequest enters the COMPLETED state when all content for a QueueRequest has been successfully transferred. This is a terminal state and the QueueRequest is completed and no further action is required. After a QueueRequest enters the COMPLETED state, applications may use the VSD accessor functions to enumerate or access downloaded Assets.

Once in the COMPLETED state, the QueueRequest may be purged. The device capabilities discovery structure shall indicate how long a QueueRequest is retained after it enters the COMPLETED state.

7.2.6 SUSPENDED state

A QueueRequest enters the SUSPENDED state when an application purposefully suspends the request, user initiated or otherwise.

From the SUSPENDED state, a QueueRequest may enter the QUEUED state after an application resumes a previously suspended request, or resubmits the request after addressing any existing error condition.

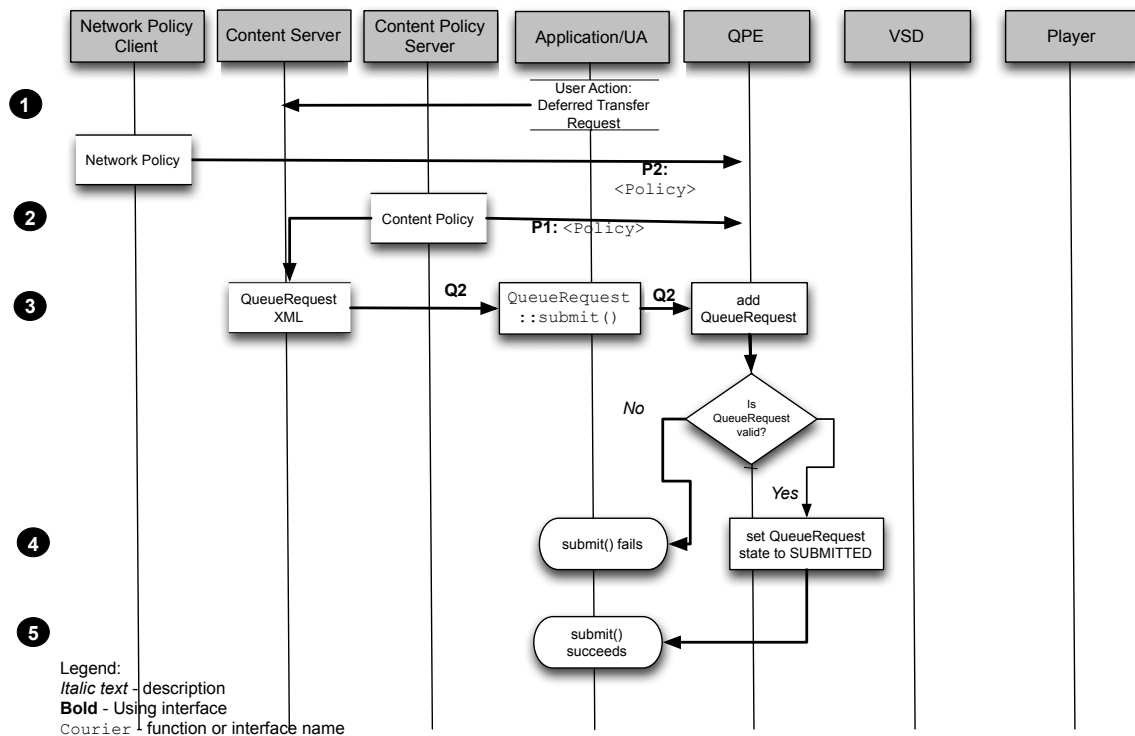
7.3 RequestId

A RequestId is assigned to a QueueRequest once the request has been submitted and validated and placed on the QPE's internal queue. The RequestId shall not change for the lifetime for the QueueRequest.

The RequestId is used to identify a particular QueueRequest on a Client Device or Intermediate Device. A Local Application or User Agent may retain a list of RequestIds to subsequently retrieve status information regarding previously submitted requests, or use the RequestId to manage, update, or cancel a previously submitted QueueRequest. An application can use the QueueManager interface to query for the RequestId if it does not retain a list of RequestIds.

7.4 Sample flow

Figure 4 illustrates the typical process of submission of a QueueRequest.



NOTE—In step 1, the application (in response to a user request or otherwise) creates a deferred transfer request—a QueueRequest object. The application may do this by retrieving an XML QueueRequest structure (see 7.7) from a Content Server or otherwise generating the QueueRequest structure. The QueueRequest XML structure includes (step 2) a URI reference to a Policy (see Clause 8) that is located on a Content Policy Server, which may include a Rule object specifying a specific network rule retrieved from a Network Policy Client. In step 3, the application submits the QueueRequest by calling RequestManager::submitRequest (see 7.6.5) over the Q2 interface. The QPE receives the QueueRequest and validates it. This may involve checking for possible compliance with the Policy, checking for supported D interface schemes, and validating the properties in the QueueRequest. If the QueueRequest is valid, the QPE sets the QueueRequest state to SUBMITTED and assigns the QueueRequest a RequestID, which it then returns (step 5) to the calling application. If the QueueRequest cannot be submitted, the submitRequest() method call fails (step 4) and an appropriate error code (see 7.5.2) is returned.

Figure 4—QueueRequest flow

7.5 Application programming interface

7.5.1 General

This subclause describes a standard interface for submitting, updating, and managing QueueRequests.

7.5.2 Error codes

NOTE—These error/response codes are also used for all other APIs.

WebIDL:

```
[NoInterfaceObject]
interface ResponseCodes {
  const long STATUS_SUCCESS      = 1;
  const long STATUS_NO_RESULT    = 2;
  const long STATUS_PROCESSING   = 3;
  const long ERR_INVALID_ARGUMENT = -1;
  const long ERR_NOT_FOUND       = -2;
  const long ERR_TIMEOUT         = -3;
  const long ERR_PENDING_OPERATION = -4;
  const long ERR_IO              = -5;
  const long ERR_NOT_SUPPORTED   = -6;
  const long ERR_PERMISSION_DENIED = -7;
  const long ERR_VSD_UNAVAILABLE = -8;
  const long ERR_INVALID_REQUEST_ID = -9;
  const long ERR_CANCEL_FAILED   = -10;
  const long ERR_SUSPEND_FAILED  = -11;
  const long ERR_RESUME_FAILED   = -12;
  const long ERR_INVALID_POLICY  = -13;
  const long ERR_INVALID_PROPERTY = -14;
  const long ERR_GENERAL         = -15;
  const long ERR_VSD_FULL        = -16;
  const long ERR_UNSUPPORTED_CONTENT_FORMAT = -17;
  const long ERR_SCHEDULE_CONFLICT = -18;
  const long ERR_INVALID_REPRESENTATION = -19;
  const long ERR_PLAYLIST_INVALID = -20;
  const long ERR_PLAYLIST_MISSING_REFERENCE = -21;
  const long ERR_PAYMENT_REQUIRED = -22;
  const long ERR_SERVER_ACCESS_DENIED = -23;
  const long ERR_SERVER_BUSY      = -24;
  const long ERR_SERVER_SUBSCRIPTION_EXPIRED = -25;
  const long ERR_USER_ACTION_REQUIRED = -26;
  const long ERR_DRM_NOT_FOUND     = -27;
  const long ERR_QUEUE_REQUEST_NOT_POSSIBLE = -28;
  const long ERR_POLICY_NOT_POSSIBLE = -29;
  const long ERR_CODEC_NOT_FOUND   = -30;
  const long ERR_INTERMEDIATE_DEVICE_NOT_FOUND = -31;
  const long ERR_CLIENT_BUSY       = -32;
  const long ERR_CONTENT_NOT_PERMITTED = -33;
  const long ERR_VERSION_NOT_RECOGNIZED = -34;
  const long ERR_PLAYLIST_NOT_TRUSTED = -35;
  const long ERR_UNRECOGNIZED_DRM   = -36;
  const long ERR_UNSOLICITED_CONTENT_PROHIBITED = -37;
```

```
readonly attribute long code;  
};
```

See 12.5 for a mapping of these error codes to HTTP return values.

Table 3 describes each error code.

Table 3—Error code descriptions

Error code	Description
ERR_INVALID_ARGUMENT	One or more arguments passed as parameters of method were invalid. As an example, null is passed where an object is expected, or a value passed as a parameter exceeds the expected range of values.
ERR_NOT_FOUND	An object, object within a database, or other construct that is to be operated on by the invoked method could not be found.
ERR_TIMEOUT	The expected duration of an invoked method has been exceeded.
ERR_PENDING_OPERATION	The invoked method could not be executed due to a previously pending operation. This may occur when a shared resource requires access that is taken by another pending operation.
ERR_IO	A method that depends on an I/O device has encountered an error. As an example, a hardware failure would result in this error being returned.
ERR_NOT_SUPPORTED	The method, or operation, feature, or function is not supported by this implementation.
ERR_PERMISSION_DENIED	The caller that invoked the method does not have the appropriate permissions to execute the method.
ERR_VSD_UNAVAILABLE	The invoked method could not access the VSD required to store the Asset.
ERR_INVALID_REQUEST_ID	The RequestId passed as an argument could not be found in the request queue.
ERR_CANCEL_FAILED	The QueueRequest could not be canceled. As an example, the application may try to cancel a request that completes before the cancellation is executed.
ERR_SUSPEND_FAILED	The QueueRequest could not be suspended. As an example, the application may try to suspend a request that completes before the cancellation is executed.
ERR_RESUME_FAILED	The QueueRequest could not be resumed. As an example, the application may try to resume a request that has an erroneous property or other setting that prohibits the request from resuming.
ERR_INVALID_POLICY	This error is encountered when one or more rules comprising the Policy was not recognized or a property value could not be parsed properly.
ERR_INVALID_PROPERTY	This error is encountered when a property key is not recognized or property value could not be parsed properly.
ERR_GENERAL	An unidentified error occurred when attempting to execute the method. This could be a database error, out-of-memory condition, or other general system failure.
ERR_VSD_FULL	A QueueRequest could not be executed because the destined VSD is full.
ERR_UNSUPPORTED_CONTENT_FORMAT	No content format defined in a QueueRequest or a Playlist is supported by the Device Description on the Client Device or a device associated with the Intermediate Device. This error can also be returned if the VSD is limited to specific content formats.

Table 3—Error code descriptions (continued)

Error code	Description
ERR_SCHEDULE_CONFLICT	The QueueRequest defines a schedule that cannot be met, or a Mandatory Time of Day rule cannot be executed due to other conflicting QueueRequests.
ERR_INVALID_REPRESENTATION	A Representation defined in a Playlist has an invalid URI or, when retrieved, has invalid content.
ERR_PLAYLIST_INVALID	A Playlist is corrupt, invalid, or does not meet the structure and criteria defined in 10.6. This error is also returned if the Playlist is missing required attributes or could not be parsed.
ERR_PLAYLIST_MISSING_REFERENCE	A Playlist references a URI that cannot be located or uses an unrecognized schema.
ERR_PAYMENT_REQUIRED	A server returned a 402 Payment Required error or otherwise indicated that content cannot be downloaded unless payment is processed.
ERR_SERVER_ACCESS_DENIED	A server returned a 403 Forbidden error in response to executing a QueueRequest.
ERR_SERVER_BUSY	A server returned a 408 Timeout or 502 Server Busy error in response to executing a QueueRequest. The QueueRequest may be retried at a later time.
ERR_SERVER_SUBSCRIPTION_EXPIRED	A server returned an error indicating that content cannot be downloaded unless a subscription is renewed.
ERR_USER_ACTION_REQUIRED	A server requires a user action (such as agreeing to new Terms of Service or updating information in a Web site) before the QueueRequest can be processed.
ERR_DRM_NOT_FOUND	The Content Policy requires a DRM implementation not available on the Client.
ERR_QUEUE_REQUEST_NOT_POSSIBLE	The QueueRequest cannot be executed since it has preconditions that can never be met on this Client. (For example, the download URL may use an unsupported schema.)
ERR_POLICY_NOT_POSSIBLE	The policy can never be met. For example, the Download clause always evaluates to false.
ERR_CODEC_NOT_FOUND	The Content Policy requires a codec implementation not available on the Client.
ERR_INTERMEDIATE_DEVICE_NOT_FOUND	The Network Policy, Content Policy, or QueueRequest indicates an Intermediate Device via the Q3 interface, but the device is not found.
ERR_CLIENT_BUSY	The QPE is busy.
ERR_CONTENT_NOT_PERMITTED	The content specified is not permitted due to Content Policy or Superuser settings. For example, the parental guidance rating specified for the content is not permitted on the Client.
ERR_VERSION_NOT_RECOGNIZED	The QueueRequest or Policy is of a version later than the Client accepting it.
ERR_PLAYLIST_NOT_TRUSTED	The Playlist is not signed and the Client only accepts signed Playlists, or the root certificate that attests to the validity of the Playlist signature is not in the Client's Certificate Trust List (CTL).
ERR_UNRECOGNIZED_DRM	The remote Device Description structure indicates a content protection scheme, but it is not recognized by this QPE.
ERR_UNSOLICITED_CONTENT_PROHIBITED	The current policy does not permit unsolicited QueueRequests to be submitted.
STATUS_NO_RESULT	A called method succeeded but did not have any values to return. For example, a search returned no results. This is not an error.
STATUS_PROCESSING	A called method is still in progress, and an event will be triggered when it is complete. This is not an error.
STATUS_UNKNOWN_CAPABILITIES	The remote Device Description structure indicates fields that are not recognized by this QPE. This is not an error.

To aid in error handling and provide a consistent mechanism for returning errors, all WebIDL objects (except built-in types) defined in this specification have a `responseCode` attribute representing the last error associated with the object. If a method returns a primitive datatype other than a long integer, the `responseCode` shall be indicated in the object called. If the method returns a complex datatype defined in this specification, the returned object's `responseCode` attribute shall indicate any error in invoking the method.

7.5.3 Version codes

The `deviceInfo` structure specifies the version of the API supported in each device. `QueueRequests` have a `REQPROP_VERSION` attribute that is automatically populated with the version code of the creating Client Device. This is persistently stored in the VSD for each object.

Calls over HTTP and Simple Service Discovery Protocol (SSDP) protocols (Q3, Q4, P1, P2, D1, D2, D3, D4) shall include a User Agent header that specifies the version of the API supported in the calling device. The called device shall return a Server header that also includes the version of the API supported.

If the recipient of a Command is of a version older than the version reported by the calling entity, and the recipient cannot emulate or accept Commands of that version, the call shall fail with a `ERR_VERSION_NOT_RECOGNIZED` error.

This document specifies version 1.0 of the API.

7.5.4 Properties

7.5.4.1 General

Properties are used to further define the objects to which they are associated.

7.5.4.2 Description

For `QueueRequests`, properties are used to describe metadata associated with a request, details about the request (source and destination URIs), and set of rules associated with a request.

The Property interface defines a simple type for storing a single key/value pair.

WebIDL:

```
interface Property {  
    attribute DOMString key;  
    attribute DOMString type;  
    attribute any value;  
    attribute long responseCode;  
};
```

XML:

```
<Property key="key" type="mimeType" >value</Property>
```

XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>  
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
elementFormDefault="qualified">  
  <xs:element name="Property">  
    <xs:complexType mixed="true">  
      <xs:attribute name="key" use="required" type="xs:string"/>  
      <xs:attribute name="type" type="xs:string"/>  
    </xs:complexType>  
  </xs:element>  
</xs:schema>
```

7.5.4.3 Property.key

The key is a DOMString identifying the key associated with the stored value.

7.5.4.4 Property.type

The type is a DOMString identifying the MIME type of the stored value. By default, this is text/plain.

7.5.4.5 Property.value

The property value attribute indicates the value associated with the stored key.

7.5.4.6 Property.responseCode

This value is set to an error code as defined in 7.5.2 if the Property is invalid as a result of an error. It is not represented in XML forms of the Property object and is not persistent.

7.5.4.7 Stringification

Binary property values can be represented in their original binary form or as DOMString values, depending on the calling protocol and API. When a MIME type represents a binary format, content is converted using Base64 content encoding algorithm as defined in IETF RFC 3548. Other content encoding algorithms (such as Quoted-Printable, Uuencode) are not directly supported. However, applications may explicitly encode and decode content using these algorithms and indicate the content encoding algorithm as part of the MIME type string.

Content with no specified MIME type is assumed to be in text/plain format, as is content with a primary type of message or text. All other MIME types (including all vnd subtypes) shall indicate a binary encoding.

7.5.5 Mandatory QueueRequest properties

Table 4 describes the mandatory properties for a QueueRequest that shall be set prior to submitting the request. If these properties are not set, the request will be in error and will not be processed.

The MIME type for all mandatory properties shall be text/plain or unspecified.

Table 4—Mandatory properties

Mandatory properties		
Key	Data type	Description
REQPROP_SOURCE_URI	DOMString	One or more source URIs where the content is to be retrieved.
REQPROP_STORE_NAME	DOMString	The name used to store the cached object on the VSD.
REQPROP_TYPE	DOMString	MIME content type of the cached object as described in IETF RFC 2045, IETF RFC 2046, IETF RFC 4288, IETF RFC 2231. Multiple MIME types may be specified, separated by a comma.
REQPROP_TOTAL_LENGTH	long	Size of the cached object in bytes. This is mandatory if the URI describes a scheme that requires a fixed size object (see Clause 10).

Multiple URIs may be specified for REQPROP_SOURCE_URI, delimited by spaces. If more than one URI is specified, the QPE may choose any of the URIs and switch between them at any time. All source URIs shall refer to the same Asset, and this Asset shall not be modified for the lifetime of the QueueRequest without changing the Etag associated with the URI.

7.5.6 Optional QueueRequest properties

Table 5 describes a set of optional properties for a QueueRequest. The MIME type for all optional properties shall be text/plain or unspecified.

Table 5—Optional Properties

Optional properties		
Key	Data type	Description
REQPROP_POLICY	XML element	Policy structure set by the calling Origin.
REQPROP_IMMEDIATE	Boolean	When true, this property indicates the request is an extremely high priority request and execution of the request should start immediately. The actual priority applied for this setting is implementation dependent. If multiple immediate requests are submitted simultaneously, and the QPE cannot activate all of them, the same Policy priorities will be used as for non-immediate requests.
REQPROP_PERMISSIONS_USER	long	This property describes the permissions assigned to the Origin or application that created the QueueRequest. For more information regarding permissions, refer to 9.2.3.
REQPROP_PERMISSIONS_GROUP	long	This property describes the permissions assigned to a group of application Origins. The REQPROP_GROUP is used to identify the Origins within the Group.

Table 5—Optional Properties (continued)

Optional properties		
Key	Data type	Description
REQPROP_PERMISSIONS_WORLD	long	This property describes the permissions assigned to all applications for a particular ContentObject created by the QueueRequest.
REQPROP_GROUP	DOMString	This property describes a list of Origins delimited by a space that creates a group of applications that are permitted to access the ContentObject created by the QueueRequest.
REQPROP_PRIORITY	long	This property defines the priority of the QueueRequest.
REQPROP_VERSION	DOMString	The version of the IEEE 2200 specification used. This should be set to “1.0.”
REQPROP_ISMPD	Boolean	When true, this property indicates that the QueueRequest is for a Playlist in the form of the Media Presentation Description (MPD) as defined in UPnP DCP Connection Manager [B11]. If Playlists are supported, the QPE shall download the Assets indicated in the Playlist as described in 10.6.2.
REQPROP_PROFILE_ID	Sequence <DOMString>	This property represents a list of one or more profile identifiers. Profile-IDs are opaque and do not have user-readable information. This information could be used by the content provider in supporting user profile based predictive download of contents.
REQPROP_ACCESS_ID	DOMString	See 7.5.6.1.
REQPROP_GEOLOCATION	DOMString	This property represents the Longitude and Latitude component of the geographical location of the Client Device. It comprises a set of coordinates of the device’s location at the time of the Queue Request submission. Longitude and Latitude information is formatted as “DDD:MM:SS.SSS,” where D indicates degrees, M indicates minutes of arc, and S indicates seconds of arc. The longitude and latitude are comma delimited. This information could be used by the content provider in supporting location-based predictive download of contents.
REQPROP_IS_UNSOLICITED	Boolean	When “true” this property indicates that the submitted QueueRequest, instead of being explicitly initiated by the user to request content, is an unsolicited request for predictive content delivery. Execution of this type of Queue Request shall comply with Content and/or Network Policy limiting or regulating predictive downloads.
REQPROP_SDP	DOMString	See 10.1.

7.5.6.1 REQPROP_ACCESS_ID

This property represents the location of the Client Device as retrieved from the wireless access network in a form generically referred to as a “Cell Identifier” or similar entity (e.g., Sector ID, Base Station ID, SSID, etc.). The format of this string is network-specific and corresponds to the device’s location at the time of the QueueRequest submission. This information could be used by the content provider in supporting location-based predictive download of contents.

In the scope of this specification, valid Cell-ID types are limited to those defined in 3GPP TS 24.312, under the *UE_Location* node of the ANDSF Management Object, i.e., ‘<X>/UE_Location/<X>’. For this specification, valid wireless access technology types associated with *UE_Location* are as follows:

- 3GPP GERAN
- 3GPP UTRAN
- 3GPP E-UTRAN
- 3GPP2 cdma2000-1X
- 3GPP2 cdma2000-HRPD
- WiMAX™ (IEEE Std 802.16 [B6])³¹
- Wireless local area network (WLAN) (IEEE Std 802.11-2012)

The following subclauses of 3GPP TS 24.312 should be referenced: 5.6.7 (GERAN_CI), 5.6.8 (UTRAN_CI), 5.6.9 (E-UTRAN_CI), 5.6.15 (cdma2000-1x Base_ID), 5.6.18 (cdma2000 HRPD Sector_ID), 5.6.23 (WiMAX BS-ID), and 5.6.27 (WLAN SSID).

In addition, for the above referenced subclauses, the following Cell-ID encoding formats shall apply:

For 3GPP GERAN, the Cell Global Identity (CGI) is as defined in 4.3.1 of 3GPP TS 23.003. The GERAN_CI (Cell Identity) portion of the CGI value is 16 bits in length.

For 3GPP UTRAN, the Cell Global Identity (CGI) is as defined in 4.3.1 of 3GPP TS 23.003. The UTRAN_CI value is equivalent to the UTRAN Cell Identity as defined in 10.3.3.2 of 3GPP TS 25.331 and this portion of the CGI is 28 bits in length.

For 3GPP E-UTRAN, the Evolved Cell Global Identifier (ECGI) is as defined in 19.6 of 3GPP TS 23.003. The EUTRA_CI value is equivalent to the ECI portion of the ECGI and is 28 bits in length.

For 3GPP2 cdma2000-1X, the concatenation of SID (System ID), NID (Network ID), and Base_ID. SID and NID are defined in 3.5.7.1.3.2 of 3GPP2 C.S0016. The format of the Base_ID is defined by 1.1.2.2 and 3.7.2.3.2.1 of 3GPP2 C.S0005.

For 3GPP2 cdma2000 HRPD, the concatenation of SID, NID and SectorID. The format of the SectorID is defined in 13.9 of 3GPP2 C.S0024-0. The length of Sector ID contained in this node is 128 bits in length.

For WiMAX, the format of the BS-ID is that of the Base Station ID as defined in 6.3.2.3.2 of IEEE Std 802.16e™-2005 and IEEE Std 802.16™-2004/Cor 1-2005.

For WLAN, the format of the SSID is defined in 7.3.2.1 in IEEE Std 802.11-2012.

³¹ WiMAX is a trademark of the WiMAX Forum.

7.5.7 Transient QueueRequest properties

Transient properties are not retained in the cached Asset. The MIME type for all transient properties will be returned as text/plain or unspecified. Transient QueueRequest properties are described in Table 6.

Table 6—Transient QueueRequest properties

Key	Type	Description
REQPROP_REQUEST_ID	long	This property provides a unique ID set when the request is submitted.
REQPROP_REQUEST_STATE	long	This property describes the state of the QueueRequest.
REQPROP_CURRENT_BYTES_TRANSFERRED	long	This property describes the total bytes transferred at the time the property was retrieved.
REQPROP_LAST_MODIFICATION_DATE	dateTime	This property describes the date the content was last modified. This provides an indication when content was transferred between the source URI and the Queue Manager.

7.5.8 Asset Properties

Table 7 lists Asset property keys that may be set within each QueueRequest. This set of metadata, if present in the QueueRequest, is added to the Asset's ContentObject in the Cache, and is retrievable using the ContentObject interface in 9.4. Unless otherwise specified, Asset properties shall have the MIME type of text/plain or unspecified.

Table 7—Asset properties in QueueRequests

Tag	Set automatically	Type	Description
ASSET_NAME	Yes	DOMString	Describes the name used to store the cached object within the Virtual Storage Device. This may be the same as the name stored in the physical media. This is derived from the REQPROP_STORE_NAME property.
ASSET_SIZE	Yes	long	Describes the size of the cached object in bytes. If not set, this may be derived from the REQPROP_TOTAL_LENGTH property.
ASSET_SOURCE_URI	Yes	DOMString	Describes the source URI where this Asset can be retrieved using the D interface. This is derived from the REQPROP_SOURCE_URI property.
ASSET_ORIGIN	Yes	DOMString	The Origin that owns this Asset. This is set automatically by the QPE and is derived from the calling Origin. Callers may not change this value.
ASSET_LOCKED	Yes	Boolean	Identified if the Asset is locked. This property is set automatically by the QPE.
ASSET_TYPE	Yes	DOMString	Describes the MIME type of the cached object. This property is derived from the REQPROP_TYPE property.
ASSET_METADATA	No	DOMString	Optional metadata may be specified using the XML specification defined in the MovieLabs EMA Metadata, version 1.2.
ASSET_POLICY	Yes	DOMString	Effective Policy as an XML string. This may be different from the initial Policy set by the calling Origin via the REQPROP_POLICY property and includes any other Policies applied to the Asset.

Table 7—Asset properties in QueueRequests (continued)

Tag	Set automatically	Type	Description
ASSET_CONTENTPROFILE	No	DOMString	Content profile supported by the Client Device (as defined in Annex B).
ASSET_RIGHTSCHECK	No	Boolean	A rights check should be performed before transitioning to the ACTIVE state. Rights checking is performed as defined in 8.3.
ASSET_VALIDITYCHECK	No	Boolean	A VSD-specific validity check should be performed before playing or consuming the content Asset. Validity checking is VSD-specific.
ASSET_PLAYLIST	Yes	DOMString	This Asset belongs to the Playlist (MPD) file specified in this property.
ASSET_ISMPD	Yes	Boolean	This Asset is a Playlist in the form of an MPD file. Assets indicated within this MPD may be cached.
ASSET_IS_UNSOLICITED	Yes	Boolean	This Asset was not solicited by the user.
ASSET_GEOLOCATION	Yes	DOMString	One or more locations in which the Asset was accessed from the cache for rendering. The representation of location is by latitude and longitude coordinates in accordance with the format indicated in 7.5.6, Table 5 for REQPROP GEOLOCATION.
ASSET_LOCATION_CELL_ID	Yes	DOMString	One or more locations in which the Asset was accessed, by CellID. The allowed types and formats of Cell-IDs are as defined in 7.5.6.1.

7.6 Request manager interface

7.6.1 General

The RequestManager interface represents the internal request queue of the QPE. Applications and servers interact with the queue by creating QueueRequests and submitting them to the RequestManager. After submission, the RequestManager may also be used to get status and properties of a QueueRequest and to manage a QueueRequest throughout its life cycle.

Figure 5 illustrates an interaction between an application and the RequestManager.

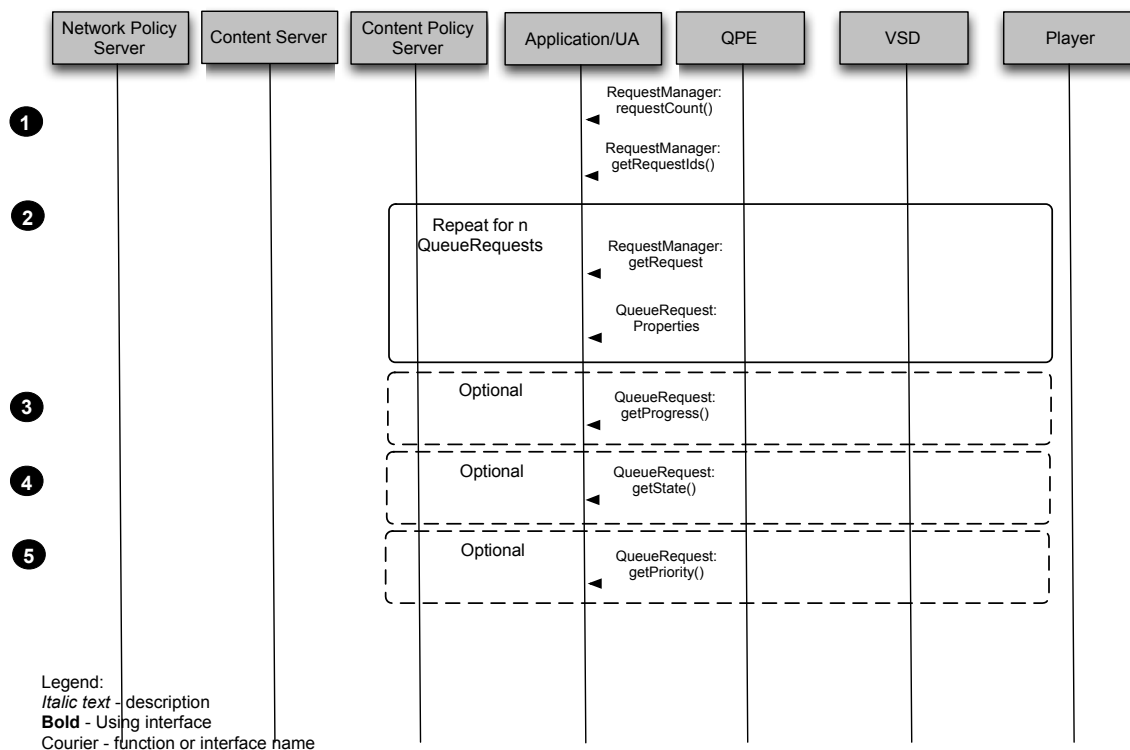


Figure 5—RequestManager

The WebIDL definition of the RequestManager interface is as follows:

WebIDL:

```

interface ReqEvent : Event {
  const unsigned short REQUEST_COMPLETE = 1;
  const unsigned short REQUEST_SUSPENDED = 2;
  const unsigned short REQUEST_RESUMED = 3;
  const unsigned short REQUEST_CONTENT_AVAILABLE = 4;

  readonly attribute unsigned short event;
  readonly attribute Property[] info;
};

[Callback=FunctionOnly, NoInterfaceObject]
interface ReqManagerCallback {
  void handleEvent(in ReqEvent event);
};

interface RequestManager {
  long requestCount(optional QueueRequestState state);
  sequence<long> getRequestIds(optional
  QueueRequestState state);
  QueueRequest? getRequest(long requestId);

  long submitRequest(QueueRequest request);

  // Operations on queued requests
  long cancelRequest(long requestId);
  
```

```
long suspendRequest(long requestId);
long resumeRequest(long requestId);

long getProgress(long requestId);
QueueRequestState getState(long requestId);
long getPriority(long requestId);
long setPriority(long requestId, long priority);

//event callback
attribute ReqManagerCallback reqCallback;

DOMString getEffectivePolicy(long requestId);
long setGlobalPolicy(DOMString policyUrl);

long setDebugInfo(DOMString location);
attribute long responseCode;
};
```

RequestManager does not have a direct representation in XML.

NOTE—In the HTTP API descriptions described in Clause 7, Clause 8, and Clause 9, italicized text is a placeholder for a parameter and should be replaced with the actual value of the parameter. Text in *Courier* represents code.

HTTP API examples do not include required HTTP headers and are illustrative only.

7.6.2 requestCount

The requestCount method retrieves a count of queued QueueRequests. This method takes an optional argument of type QueueRequestState where the count can be limited to QueueRequests in a particular state.

This method returns a long value. If the value is non-negative, it represents the request count. If the value is negative, it is an error code as defined in the following table.

WebIDL

```
long requestCount(optional QueueRequestState state);
```

HTTP

```
POST /RequestManager/requestCount HTTP/1.1
<Parameters>
  <Parameter name="state">QueueRequestState</Parameter>
</Parameters>
```

Valid interfaces

The requestCount method may be called from any Q interface.

Arguments

(optional QueueRequestState) a RequestState to filter by. If specified, only requests matching this state will be counted.

Return value

(long) the number of requests matching the filter. A negative number indicates an error code as listed in the following table.

```
<returnValue>request-count</returnValue>
```

Error codes

RequestManager::requestCount error codes

Error code	Description
ERR_INVALID_ARGUMENT	The QueueRequest state argument was not a valid state.
ERR_TIMEOUT	The expected duration of an invoked method has been exceeded.
ERR_PENDING_OPERATION	The invoked method could not be executed due to a previously pending operation. This may occur when a shared resource requires access that is taken by another pending operation.
ERR_PERMISSION_DENIED	The caller is a blacklisted or unauthorized Origin.
ERR_GENERAL	An unidentified error occurred when attempting to execute the method. This could be a database error, out-of-memory condition, or other general system failure.
ERR_CLIENT_BUSY	The QPE is busy.

7.6.3 getRequestIds

This method retrieves an array of RequestIds associated with the currently queued QueueRequests. This method takes an optional argument of type QueueRequestState. If the `state` argument is specified, the returned array will be limited to the RequestIds for QueueRequests in a particular state. The method will only return QueueRequests visible to the calling Origin.

This method returns an array of RequestIds. If an error occurs, the method will return an empty sequence and set the RequestManager's `responseCode` attribute to an error code. An HTTP invocation will return an error code in the status response.

WebIDL

```
sequence<long> getRequestIds(optional QueueRequestState state);
```

HTTP

```
POST /RequestManager/requestIds HTTP/1.1
<Parameters>
  <Parameter name="state">QueueRequestState</Parameter>
</Parameters>
```

Valid interfaces

This function may be called from any Q interface.

Arguments

(optional QueueRequestState) a RequestState to filter by. If specified, only requests matching this state will be counted.

Return value

(sequence<long>) a sequence of requestIDs matching the filter. A negative number indicates an error code.

<returnValue>*requestID1, requestID2, ...*</returnValue>

Error codes

RequestManager::requestIds error codes

Error code	Description
ERR_INVALID_ARGUMENT	The QueueRequest state argument was not a valid state.
ERR_TIMEOUT	The expected duration of an invoked method has been exceeded.
ERR_PENDING_OPERATION	The invoked method could not be executed due to a previously pending operation. This may occur when a shared resource requires access that is taken by another pending operation.
ERR_NOT_SUPPORTED	The method, or operation, feature, or function is not supported by this implementation.
ERR_PERMISSION_DENIED	The caller is a blacklisted Origin or does not have Read permission for any QueueRequests.
ERR_GENERAL	An unidentified error occurred when attempting to execute the method. This could be a database error, out-of-memory condition, or other general system failure.
ERR_CLIENT_BUSY	The QPE is busy.

7.6.4 getRequest

This method retrieves a QueueRequest object from the request queue. This method takes one argument of type long, which is used to identify the QueueRequest of interest.

This method returns a QueueRequest object. If an error occurs, the method will return NULL. The error that caused the getRequest method to fail will be specified in the RequestManager's responseCode attribute.

Only QueueRequests visible to the calling Origin may be returned using this function. If an application requests a QueueRequest with a RequestId that is not visible to it using the permissions defined in each request, the method will fail and return NULL, and the ERR_PERMISSION_DENIED error code will be specified in the RequestManager's responseCode attribute.

WebIDL

```
QueueRequest? getRequest(long requestId);
```

HTTP

```
GET /RequestManager/Request/requestID HTTP/1.1
```

Valid interfaces

This function may be called from any Q interface.

Arguments

(long RequestID) the requestID indicating the request ID of interest.

Return value

(QueueRequest) the QueueRequest object

```
<returnValue>
<QueueRequest>...
</QueueRequest>
</returnValue>
```

See 7.7.1 for an XML representation of a QueueRequest.

Error codes

RequestManager::getRequest error codes

Error code	Description
ERR_INVALID_ARGUMENT	The RequestID passed in the URI could not be parsed.
ERR_TIMEOUT	The expected duration of an invoked method has been exceeded.
ERR_NOT_SUPPORTED	The method, or operation, feature, or function is not supported by this implementation. This is also returned if the function was called over the D or P interfaces.
ERR_PERMISSION_DENIED	The caller does not have Read permission for this requestID.
ERR_INVALID_REQUEST_ID	The RequestId passed as an argument could not be found in the request queue.
ERR_GENERAL	An unidentified error occurred when attempting to execute the method. This could be a database error, out-of-memory condition, or other general system failure.

7.6.5 submitRequest

This method submits a QueueRequest to the request queue. This method takes one argument of type QueueRequest that is used to convey all the details associated with a request.

This method returns a long value. If the method completes successfully, the newly assigned request ID is returned. If the submission failed (as described in 7.2.1), an error code will be returned and the RequestManager's responseCode attribute will be set to the returned error code.

After a QueueRequest has been submitted, and the application retains the QueueRequest object, any modifications made to the QueueRequest shall be resubmitted using this method in order for the changes to take effect.

It should be noted that calling submitRequest does not immediately return errors associated with executing the QueueRequest. For example, if the server returns an error message when the QueueRequest transitions to ACTIVE, the submitRequest call may still return STATUS_SUCCESS. The actual error code will be in the QueueRequest object when the QueueRequest transitions. Implementations may, but are not required to, preflight QueueRequests in order to return error codes before the QueueRequest transitions to ACTIVE.

WebIDL

```
long submitRequest(QueueRequest request);
```

HTTP

```
POST /RequestManager/submitRequest HTTP/1.1
<Parameters>
  <Parameter name="request">
    <QueueRequest>...</QueueRequest>
  </Parameter>
</Parameters>
```

See 7.7.1 for an XML representation of a QueueRequest.

Valid interfaces

This function may be called from any Q interface.

Arguments

(QueueRequest) *request* the QueueRequest object to submit.

Return value

(long) the requestID assigned to the submitted request, or an error code on failure

```
<returnValue>requestID</returnValue>
```

Error codes

RequestManager::submitRequest error codes

Error code	Description
ERR_INVALID_ARGUMENT	The QueueRequest object could not be recognized, or a parameter name was not matched. This error is also returned if the function was called over the P or D interfaces.
ERR_TIMEOUT	The expected duration of an invoked method has been exceeded.
ERR_PENDING_OPERATION	The invoked method could not be executed due to a previously pending operation. For example, the RequestManager is locked due to a concurrent operation as described in 6.3.
ERR_NOT_SUPPORTED	The method, or operation, feature, or function is not supported by this implementation. This is also returned if the function was called over the D or P interfaces.
ERR_PERMISSION_DENIED	The caller is blacklisted or is not authorized to submit QueueRequests.
ERR_VSD_UNAVAILABLE	The QueueRequest specifies a VSD that is not installed on the device. (If the VSD is installed but not currently available, such as removable media that is not currently mounted, this error is not returned and the submitRequest succeeds.)

ERR_INVALID_POLICY	This error is encountered when one or more rules comprising the Policy in the submitted QueueRequest was not recognized or a property value could not be parsed properly.
ERR_INVALID_PROPERTY	This error is encountered when a property key is not recognized or property value could not be parsed properly. The implementation may include additional information specifying the property that was not recognized using the debug log.
ERR_GENERAL	An unidentified error occurred when attempting to execute the method. This could be a database error, out-of-memory condition, or other general system failure.
ERR_UNSUPPORTED_CONTENT_FORMAT	No content format defined in the submitted QueueRequest is supported by the Device Description on the Client Device or a device associated with the Intermediate Device.
ERR_SCHEDULE_CONFLICT	The QueueRequest defines a schedule that cannot be met, or a Mandatory Time of Day rule cannot be executed due to other conflicting QueueRequests.
ERR_USER_ACTION_REQUIRED	A server requires a user action (such as agreeing to new Terms of Service or updating information in a Web site) before the QueueRequest can be processed.
ERR_DRM_NOT_FOUND	The Content Policy requires a DRM implementation not available on the Client.
ERR_QUEUE_REQUEST_NOT_POSSIBLE	The QueueRequest cannot be executed since it has preconditions that can never be met on this Client. (For example, the download URL may use an unsupported schema.)
ERR_POLICY_NOT_POSSIBLE	The policy can never be met. For example, the Download clause always evaluates to false.
ERR_CODEC_NOT_FOUND	The Content Policy requires a codec implementation not available on the Client.
ERR_INTERMEDIATE_DEVICE_NOT_FOUND	The Network Policy, Content Policy, or QueueRequest indicates an Intermediate Device via the Q3 interface, but the device is not found.
ERR_CLIENT_BUSY	The QPE is busy.
ERR_CONTENT_NOT_PERMITTED	The content specified is not permitted due to Content Policy or Superuser settings. For example, the parental guidance rating specified for the content is not permitted on the Client.
ERR_VERSION_NOT_RECOGNIZED	The QueueRequest or Policy is of a version later than the Client accepting it.
STATUS_PROCESSING	A called method is still in progress, and an event will be triggered when it is complete. This is not an error.

7.6.6 cancelRequest

This method cancels a previously submitted QueueRequest in the request queue. This method takes one argument of type `long` that is used to identify the QueueRequest.

This method returns a `long` value. If the method completes successfully, the value `STATUS_SUCCESS` is returned. If an error occurs, an error code is returned, and the RequestManager's `responseCode` attribute will be set to the returned error code.

Only requests for which the calling origin has Delete permission may be cancelled using this function.

WebIDL

```
long cancelRequest(long requestId);
```