



ISO/IEC 29341-4-14

Edition 1.0 2008-11

INTERNATIONAL STANDARD

**Information technology – UPnP Device Architecture –
Part 4-14: Audio Video Device Control Protocol –
Level 2 – Scheduled Recording Service**



THIS PUBLICATION IS COPYRIGHT PROTECTED
Copyright © 2008 ISO/IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester.

If you have any questions about ISO/IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

IEC Central Office
3, rue de Varembe
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub

The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,...). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00



ISO/IEC 29341-4-14

Edition 1.0 2008-11

INTERNATIONAL STANDARD

**Information technology – UPnP Device Architecture –
Part 4-14: Audio Video Device Control Protocol –
Level 2 – Scheduled Recording Service**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE

XE

ICS 35.200

ISBN 2-8318-1006-9

CONTENTS

FOREWORD	10
ORIGINAL UPNP DOCUMENTS (informative)	12
1 Overview and Scope	14
1.1 Introduction	14
1.2 Notation	14
1.2.1 Data Types	15
1.2.2 Strings Embedded in Other Strings	15
1.2.3 Extended Backus-Naur Form	15
1.3 Derived Data Types	16
1.3.1 Comma Separated Value (CSV) Lists	16
1.4 Management of XML Namespaces in Standardized DCPs	17
1.4.1 Namespace Prefix Requirements	19
1.4.2 Namespace Names, Namespace Versioning and Schema Versioning	20
1.4.3 Namespace Usage Examples	21
1.5 Vendor-defined Extensions	22
1.6 References	22
2 Service Modeling Definitions	25
2.1 ServiceType	25
2.2 Terms and Abbreviations	25
2.2.1 Abbreviations	25
2.2.2 Terms	25
2.3 ScheduledRecording Service Architecture	31
2.3.1 <u>recordSchedule</u>	31
2.3.2 <u>recordTask</u>	32
2.4 State Variables	33
2.4.1 State Variable Overview	33
2.4.2 <u>SortCapabilities</u>	35
2.4.3 <u>SortLevelCapability</u>	35
2.4.4 <u>StateUpdateID</u>	35
2.4.5 <u>LastChange</u>	36
2.4.6 <u>A ARG TYPE PropertyList</u>	38
2.4.7 <u>A ARG TYPE DataTypeId</u>	39
2.4.8 <u>A ARG TYPE ObjectID</u>	39
2.4.9 <u>A ARG TYPE ObjectIDList</u>	39
2.4.10 <u>A ARG TYPE PropertyInfo</u>	39
2.4.11 <u>A ARG TYPE Index</u>	39
2.4.12 <u>A ARG TYPE Count</u>	39
2.4.13 <u>A ARG TYPE SortCriteria</u>	39
2.4.14 <u>A ARG TYPE RecordSchedule</u>	39
2.4.15 <u>A ARG TYPE RecordTask</u>	40
2.4.16 <u>A ARG TYPE RecordScheduleParts</u>	40
2.5 Eventing and Moderation	41
2.6 Actions	41
2.6.1 <u>GetSortCapabilities()</u>	42
2.6.2 <u>GetPropertyList()</u>	43
2.6.3 <u>GetAllowedValues()</u>	44

2.6.4	<u>GetStateUpdateID()</u>	45
2.6.5	<u>BrowseRecordSchedules()</u>	46
2.6.6	<u>BrowseRecordTasks()</u>	50
2.6.7	<u>CreateRecordSchedule()</u>	52
2.6.8	<u>DeleteRecordSchedule()</u>	54
2.6.9	<u>GetRecordSchedule()</u>	56
2.6.10	<u>EnableRecordSchedule()</u>	57
2.6.11	<u>DisableRecordSchedule()</u>	58
2.6.12	<u>DeleteRecordTask()</u>	59
2.6.13	<u>GetRecordTask()</u>	60
2.6.14	<u>EnableRecordTask()</u>	61
2.6.15	<u>DisableRecordTask()</u>	62
2.6.16	<u>ResetRecordTask()</u>	63
2.6.17	<u>GetRecordScheduleConflicts()</u>	63
2.6.18	<u>GetRecordTaskConflicts()</u>	65
2.6.19	Common Error Codes	66
2.7	State Diagram of <u>recordTask</u>	67
2.7.1	A Full-Featured State Diagram	67
2.7.2	A Minimal-Implementation State Diagram	72
2.7.3	<u>recordTask</u> State Example	75
2.8	ScheduledRecording Service Priority Model	76
2.8.1	Introduction of the ScheduledRecording Service Priority Model	76
2.8.2	Ordered Priority within Each Priority Level	77
2.8.3	Setting the Initial Priority Level of a <u>recordSchedule</u>	78
2.8.4	Sorting <u>recordSchedule</u> Instances Based on their Current Priority Settings	80
2.9	Theory of Operation	80
2.9.1	Introduction	80
2.9.2	Checking the Capabilities of a ScheduledRecording Service	80
2.9.3	Adding a Scheduled Recording Entry to the List	90
2.9.4	Deleting a <u>recordSchedule</u>	103
2.9.5	Browsing <u>recordSchedule</u> and <u>recordTask</u> instances	103
2.9.6	Rating System	109
2.9.7	Conflict Detection and Resolution	110
3	XML Service Description	111
4	Test	120
Annex A	(normative) srs XML Document	121
A.1	<u>A ARG TYPE RecordSchedule</u> AVDT XML Document	121
A.2	<u>A ARG TYPE RecordTask</u> AVDT XML Document	122
A.3	<u>A ARG TYPE RecordScheduleParts</u> AVDT XML Document	122
Annex B	(normative) AV Working Committee Extended Properties	124
B.1	Base Properties	124
B.1.1	<u>@id</u>	124
B.1.2	<u>title</u>	125
B.1.3	<u>class</u>	125
B.1.4	<u>additionalStatusInfo</u>	125
B.1.5	<u>cdsReference</u>	126
B.2	Priority Properties	126
B.2.1	<u>priority</u>	126

B.2.2	<u>desiredPriority</u>	128
B.2.3	<u>desiredPriority@type</u>	129
B.3	Output Control Properties	130
B.3.1	<u>recordDestination</u>	130
B.3.2	<u>desiredRecordQuality</u>	132
B.4	Content Identification Related Properties	134
B.4.1	<u>scheduledCDSObjectID</u>	134
B.4.2	<u>scheduledChannelID</u>	135
B.4.3	<u>scheduledStartDateTime</u>	136
B.4.4	<u>scheduledDuration</u>	136
B.4.5	<u>scheduledProgramCode</u>	137
B.5	Matching Content Criteria Properties	137
B.5.1	<u>matchingName</u>	138
B.5.2	<u>matchingID</u>	139
B.6	Matching Qualifying Criteria Properties	140
B.6.1	<u>matchingChannelID</u>	140
B.6.2	<u>matchingStartDateTimeRange</u>	140
B.6.3	<u>matchingDurationRange</u>	141
B.6.4	<u>matchingRatingLimit</u>	141
B.6.5	<u>matchingEpisodeType</u>	143
B.7	Content Control Properties	144
B.7.1	<u>totalDesiredRecordTasks</u>	144
B.7.2	<u>scheduledStartDateTimeAdjust</u>	145
B.7.3	<u>scheduledDurationAdjust</u>	145
B.7.4	<u>activePeriod</u>	145
B.7.5	<u>durationLimit</u>	146
B.7.6	<u>channelMigration</u>	147
B.7.7	<u>timeMigration</u>	147
B.7.8	<u>allowDuplicates</u>	147
B.8	Storage Related Properties	148
B.8.1	<u>persistedRecordings</u>	148
B.9	Schedule State Properties	149
B.9.1	<u>scheduleState</u>	149
B.9.2	<u>abnormalTasksExist</u>	151
B.10	Statistics Properties	151
B.10.1	<u>currentRecordTaskCount</u>	151
B.10.2	<u>totalCreatedRecordTasks</u>	151
B.10.3	<u>totalCompletedRecordTasks</u>	152
B.11	Task General Properties	152
B.11.1	<u>recordScheduleID</u>	152
B.11.2	<u>recordedCDSObjectID</u>	152
B.12	Task Content Identification Properties	153
B.12.1	<u>taskCDSObjectID</u>	153
B.12.2	<u>taskChannelID</u>	154
B.12.3	<u>taskStartDateTime</u>	154
B.12.4	<u>taskDuration</u>	154
B.12.5	<u>taskProgramCode</u>	155
B.12.6	<u>recordQuality</u>	155
B.13	Task Matched Content Criteria Properties	155

B.13.1	<u>matchedName</u>	157
B.13.2	<u>matchedID</u>	158
B.14	Task Matched Qualifying Criteria Properties	158
B.14.1	<u>matchedRating</u>	158
B.14.2	<u>matchedRating@type</u>	159
B.14.3	<u>matchedEpisodeType</u>	159
B.15	Task Matched Content Control Properties	159
B.15.1	<u>taskStartDateTimeAdjust</u>	159
B.15.2	<u>taskDurationAdjust</u>	160
B.15.3	<u>taskDurationLimit</u>	160
B.15.4	<u>taskDurationLimit@effect</u>	160
B.15.5	<u>taskChannelMigration</u>	160
B.15.6	<u>taskTimeMigration</u>	161
B.16	Task State Properties	161
B.16.1	<u>taskState</u>	161
B.17	ContentDirectory Service Imported Properties	169
Annex C	(normative) AV Working Committee Class Definitions	174
C.1	Class Hierarchy	174
C.1.1	Relationships between Classes and Properties	175
C.1.2	<u>recordScheduleParts</u> Properties	176
C.1.3	<u>recordSchedule</u> Properties	180
C.1.4	<u>recordTask</u> Properties	183
C.2	Class Definitions	187
C.3	<u>object</u> Base Class	187
C.3.1	<u>object.recordSchedule</u> Class	189
C.3.2	<u>object.recordTask</u> Class	198
Annex D	(normative) EBNF Syntax Definitions	200
D.1	Priority Syntax	200
D.2	Date&time Syntax	200
D.3	Class Name Syntax	201
Annex E	(informative) ScheduledRecording Service Relationship to ContentDirectory Service	202
Annex F	(informative) ScheduledRecording Service Relationship to EPG	203
Annex G	(informative) AVDT Examples	204
G.1	<u>A ARG TYPE RecordSchedule</u> AVDT Example	204
G.2	<u>A ARG TYPE RecordTask</u> AVDT Example	220
G.3	<u>A ARG TYPE RecordScheduleParts</u> AVDT Example	238

LIST OF TABLES

Table 1-1:	EBNF Operators	16
Table 1-2:	CSV Examples	17
Table 1-3:	Namespace Definitions.....	18
Table 1-4:	Schema-related Information	19
Table 1-5:	Default Namespaces for the AV Specifications.....	20
Table 2-1:	Abbreviations	25
Table 2-1:	Properties in XML	27
Table 2-2:	State Variables	33
Table 2-3:	allowedValueList for the <u>DataTypeID</u> argument	35
Table 2-4:	Allowed Elements in <StateEvent> Element.....	36
Table 2-5:	Eventing and Moderation.....	41
Table 2-6:	Actions.....	41
Table 2-7:	Arguments for <u>GetSortCapabilities()</u>	42
Table 2-8:	Error Codes for <u>GetSortCapabilities()</u>	43
Table 2-9:	Arguments for <u>GetPropertyList()</u>	43
Table 2-10:	Error Codes for <u>GetPropertyList()</u>	44
Table 2-11:	Arguments for <u>GetAllowedValues()</u>	44
Table 2-12:	Error Codes for <u>GetAllowedValues()</u>	45
Table 2-13:	Arguments for <u>GetStateUpdateID()</u>	46
Table 2-14:	Error Codes for <u>GetStateUpdateID()</u>	46
Table 2-15:	Arguments for <u>BrowseRecordSchedules()</u>	46
Table 2-16:	Error Codes for <u>BrowseRecordSchedules()</u>	50
Table 2-17:	Arguments for <u>BrowseRecordTasks()</u>	51
Table 2-18:	Error Codes for <u>BrowseRecordTasks()</u>	52
Table 2-19:	Arguments for <u>CreateRecordSchedule()</u>	53
Table 2-20:	Error Codes for <u>CreateRecordSchedule()</u>	54
Table 2-21:	Arguments for <u>DeleteRecordSchedule()</u>	55
Table 2-22:	Error Codes for <u>DeleteRecordSchedule()</u>	55
Table 2-23:	Arguments for <u>GetRecordSchedule()</u>	56
Table 2-24:	Error Codes for <u>GetRecordSchedule()</u>	56
Table 2-25:	Arguments for <u>EnableRecordSchedule()</u>	57
Table 2-26:	Error Codes for <u>EnableRecordSchedule()</u>	57
Table 2-27:	Arguments for <u>DisableRecordSchedule()</u>	58
Table 2-28:	Error Codes for <u>DisableRecordSchedule()</u>	58
Table 2-29:	Arguments for <u>DeleteRecordTask()</u>	59
Table 2-30:	Error Codes for <u>DeleteRecordTask()</u>	59
Table 2-31:	Arguments for <u>GetRecordTask()</u>	60
Table 2-32:	Error Codes for <u>GetRecordTask()</u>	60
Table 2-33:	Arguments for <u>EnableRecordTask()</u>	61



Table 2-34:	Error Codes for <u>EnableRecordTask()</u>	61
Table 2-35:	Arguments for <u>DisableRecordTask()</u>	62
Table 2-36:	Error Codes for <u>DisableRecordTask()</u>	62
Table 2-37:	Arguments for <u>ResetRecordTask()</u>	63
Table 2-38:	Error Codes for <u>ResetRecordTask()</u>	63
Table 2-39:	Arguments for <u>GetRecordScheduleConflicts()</u>	64
Table 2-40:	Error Codes for <u>GetRecordScheduleConflicts()</u>	64
Table 2-41:	Arguments for <u>GetRecordTaskConflicts()</u>	65
Table 2-42:	Error Codes for <u>GetRecordTaskConflicts()</u>	65
Table 2-43:	Common Error Codes	66
Table 2-44:	<u>recordTask</u> State Timeline.....	76
Table 2-45:	Example 1: Fewer <u>recordSchedule</u> instances than the Number of Supported Priority Levels.	77
Table 2-46:	Example 2: More <u>recordSchedule</u> instances than the Number of Supported Priority Levels.	78
Table 2-47:	Existing <u>recordSchedule</u> Priorities	79
Table 2-48:	<u>desiredPriority</u> Property Set to “RS-C”	79
Table 2-49:	<u>desiredPriority</u> Property Set to “HIGHEST”, “L1 HI”, or “RS-A”	79
Table 2-50:	<u>desiredPriority</u> Property Set to “LOWEST”, “L3 LOW”, or “RS-B”	80
Table 2-51:	<u>desiredPriority</u> Property Set to “RS-C”	80
Table B-1:	Base Properties Overview	124
Table B-2:	allowedValueList for the <u>class</u> Property	125
Table B-3:	Priority Properties	126
Table B-4:	allowedValueList for the <u>priority</u> Property	127
Table B-5:	Primary allowedValueList for the <u>desiredPriority</u> Property	128
Table B-6:	Additional allowedValueList for the <u>desiredPriority</u> Property	129
Table B-7:	allowedValueList for the <u>desiredPriority@type</u> Property.....	130
Table B-8:	Output Control Properties	130
Table B-9:	<u>desiredRecordQuality</u> Example	132
Table B-10:	allowedValueList for the <u>desiredRecordQuality</u> Property	133
Table B-11:	allowedValueList for the <u>desiredRecordQuality@type</u> Property	134
Table B-12:	Content Identification Related Properties	134
Table B-13:	allowedValueList for the <u>scheduledChannelID@type</u> Property	136
Table B-14:	Matching Content Criteria Properties	137
Table B-15:	allowedValueList for the <u>matchingName@type</u> Property	138
Table B-16:	allowedValueList for the <u>matchingID@type</u> Property	139
Table B-17:	Matching Qualifying Criteria Properties	140
Table B-18:	allowedValueList for the <u>matchingRatingLimit</u> Property Using the MPAA Rating System (<u>matchingRatingLimit@type</u> = “MPAA.ORG”)	142
Table B-19:	allowedValueList for the <u>matchingRatingLimit</u> Property Using the RIAA Rating System (<u>matchingRatingLimit@type</u> = “RIAA.ORG”)	142

Table B-20:	allowedValueList for the matchingRatingLimit Property Using the ESRB Rating System (matchingRatingLimit@type = “ ESRB.ORG ”).....	142
Table B-21:	allowedValueList for the matchingRatingLimit Property Using the TVGUIDELINES Rating System (matchingRatingLimit@type = “ TVGUIDELINES.ORG ”).....	142
Table B-22:	allowedValueList for the matchingRatingLimit@type Property	143
Table B-23:	allowedValueList for the matchingEpisodeType Property	144
Table B-24:	Content Control Properties	144
Table B-25:	allowedValueList for the durationLimit@effect Property	147
Table B-26:	Storage Related Properties	148
Table B-27:	Schedule State Properties	149
Table B-28:	allowedValueList for the scheduleState Property	150
Table B-29:	allowedValueList for the scheduleState@currentErrors Property	150
Table B-30:	Statistics Properties	151
Table B-31:	Task General Properties.....	152
Table B-32:	Task Content Identification Properties.....	153
Table B-33:	recordQuality Example	156
Table B-34:	allowedValueList for the recordQuality Property.....	157
Table B-35:	Task Matched Content Criteria Properties	157
Table B-36:	Task Matched Qualifying Criteria Properties	158
Table B-37:	Task Matched Content Control Properties	159
Table B-38:	State Related Properties.....	161
Table B-39:	allowedValueList for the taskState Property	162
Table B-40:	allowedValueList for the taskState Property	163
Table B-41:	allowedValueList for the taskState@phase Property.....	164
Table B-42:	allowedValueList for the taskState@xxx Properties	167
Table C-1:	Class Properties Overview for recordScheduleParts usage.....	177
Table C-2:	Class Properties Overview for recordSchedule usage	180
Table C-3:	Class Properties Overview for recordTask usage	184
Table C-4:	object Base Class Properties	188
Table C-5:	object.recordSchedule Base Class Properties	189
Table C-6:	object.recordSchedule.direct Class Properties.....	190
Table C-7:	object.recordSchedule.direct.manual Class Properties.....	191
Table C-8:	object.recordSchedule.direct.cdsEPG Class Properties	192
Table C-9:	object.recordSchedule.direct.cdsNonEPG Class Properties.....	193
Table C-10:	object.recordSchedule.direct.programCode Class Properties.....	193
Table C-11:	object.recordSchedule.query Class Properties	194
Table C-12:	object.recordSchedule.query.contentName Class Properties.....	195
Table C-13:	object.recordSchedule.query.contentID Class Properties	197
Table C-14:	object.recordTask Base Class Properties	199

LIST OF FIGURES

Figure 1: Creating a new <u>recordSchedule</u>	31
Figure 2: Capability check.	32
Figure 3: Browse <u>recordSchedule</u>	32
Figure 4: Delete a <u>recordSchedule</u>	32
Figure 5: A Full-Featured State Diagram.....	68
Figure 6: A Minimal-Implementation State Diagram.....	73
Figure 7: Class hierarchy for the ScheduledRecording service.....	174

INFORMATION TECHNOLOGY – UPNP DEVICE ARCHITECTURE –

Part 4-14: Audio Video Device Control Protocol – Level 2 – Scheduled Recording Service

FOREWORD

- 1) ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards. Their preparation is entrusted to technical committees; any ISO and IEC member body interested in the subject dealt with may participate in this preparatory work. International governmental and non-governmental organizations liaising with ISO and IEC also participate in this preparation.
- 2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.
- 3) The formal decisions or agreements of IEC and ISO on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC and ISO member bodies.
- 4) IEC, ISO and ISO/IEC publications have the form of recommendations for international use and are accepted by IEC and ISO member bodies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC, ISO and ISO/IEC publications is accurate, IEC or ISO cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 5) In order to promote international uniformity, IEC and ISO member bodies undertake to apply IEC, ISO and ISO/IEC publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any ISO/IEC publication and the corresponding national or regional publication should be clearly indicated in the latter.
- 6) ISO and IEC provide no marking procedure to indicate their approval and cannot be rendered responsible for any equipment declared to be in conformity with an ISO/IEC publication.
- 7) All users should ensure that they have the latest edition of this publication.
- 8) No liability shall attach to IEC or ISO or its directors, employees, servants or agents including individual experts and members of their technical committees and IEC or ISO member bodies for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication of, use of, or reliance upon, this ISO/IEC publication or any other IEC, ISO or ISO/IEC publications.
- 9) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

IEC and ISO draw attention to the fact that it is claimed that compliance with this document may involve the use of patents as indicated below.

ISO and IEC take no position concerning the evidence, validity and scope of the putative patent rights. The holders of the putative patent rights have assured IEC and ISO that they are willing to negotiate free licences or licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of the putative patent rights are registered with IEC and ISO.

Intel Corporation has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Intel Corporation
Standards Licensing Department
5200 NE Elam Young Parkway
MS: JFS-98
USA – Hillsboro, Oregon 97124

Microsoft Corporation has informed IEC and ISO that it has patent applications or granted patents as listed below:

6101499 / US; 6687755 / US; 6910068 / US; 7130895 / US; 6725281 / US; 7089307 / US; 7069312 / US;
10/783 524 / US

Information may be obtained from:

Microsoft Corporation
One Microsoft Way
USA – Redmond WA 98052

Philips International B.V. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Philips International B.V. – IP&S
High Tech campus, building 44 3A21
NL – 5656 Eindhoven

NXP B.V. (NL) has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

NXP B.V. (NL)
High Tech campus 60
NL – 5656 AG Eindhoven

Matsushita Electric Industrial Co. Ltd. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Matsushita Electric Industrial Co. Ltd.
1-3-7 Shiromi, Chuoh-ku
JP – Osaka 540-6139

Hewlett Packard Company has informed IEC and ISO that it has patent applications or granted patents as listed below:

5 956 487 / US; 6 170 007 / US; 6 139 177 / US; 6 529 936 / US; 6 470 339 / US; 6 571 388 / US; 6 205 466 / US

Information may be obtained from:

Hewlett Packard Company
1501 Page Mill Road
USA – Palo Alto, CA 94304

Samsung Electronics Co. Ltd. has informed IEC and ISO that it has patent applications or granted patents.

Information may be obtained from:

Digital Media Business, Samsung Electronics Co. Ltd.
416 Maetan-3 Dong, Yeongtang-Gu,
KR – Suwon City 443-742

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. IEC and ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29341-4-14 was prepared by UPnP Implementers Corporation and adopted, under the PAS procedure, by joint technical committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

The list of all currently available parts of the ISO/IEC 29341 series, under the general title *Universal plug and play (UPnP) architecture*, can be found on the IEC web site.

This International Standard has been approved by vote of the member bodies, and the voting results may be obtained from the address given on the second title page.

ORIGINAL UPNP DOCUMENTS (informative)

Reference may be made in this document to original UPnP documents. These references are retained in order to maintain consistency between the specifications as published by ISO/IEC and by UPnP Implementers Corporation. The following table indicates the original UPnP document titles and the corresponding part of ISO/IEC 29341:

UPnP Document Title	ISO/IEC 29341 Part
UPnP Device Architecture 1.0	ISO/IEC 29341-1
UPnP Basic:1 Device	ISO/IEC 29341-2
UPnP AV Architecture:1	ISO/IEC 29341-3-1
UPnP MediaRenderer:1 Device	ISO/IEC 29341-3-2
UPnP MediaServer:1 Device	ISO/IEC 29341-3-3
UPnP AVTransport:1 Service	ISO/IEC 29341-3-10
UPnP ConnectionManager:1 Service	ISO/IEC 29341-3-11
UPnP ContentDirectory:1 Service	ISO/IEC 29341-3-12
UPnP RenderingControl:1 Service	ISO/IEC 29341-3-13
UPnP MediaRenderer:2 Device	ISO/IEC 29341-4-2
UPnP MediaServer:2 Device	ISO/IEC 29341-4-3
UPnP AV Datastructure Template:1	ISO/IEC 29341-4-4
UPnP AVTransport:2 Service	ISO/IEC 29341-4-10
UPnP ConnectionManager:2 Service	ISO/IEC 29341-4-11
UPnP ContentDirectory:2 Service	ISO/IEC 29341-4-12
UPnP RenderingControl:2 Service	ISO/IEC 29341-4-13
UPnP ScheduledRecording:1	ISO/IEC 29341-4-14
UPnP DigitalSecurityCamera:1 Device	ISO/IEC 29341-5-1
UPnP DigitalSecurityCameraMotionImage:1 Service	ISO/IEC 29341-5-10
UPnP DigitalSecurityCameraSettings:1 Service	ISO/IEC 29341-5-11
UPnP DigitalSecurityCameraStillImage:1 Service	ISO/IEC 29341-5-12
UPnP HVAC_System:1 Device	ISO/IEC 29341-6-1
UPnP HVAC_ZoneThermostat:1 Device	ISO/IEC 29341-6-2
UPnP ControlValve:1 Service	ISO/IEC 29341-6-10
UPnP HVAC_FanOperatingMode:1 Service	ISO/IEC 29341-6-11
UPnP FanSpeed:1 Service	ISO/IEC 29341-6-12
UPnP HouseStatus:1 Service	ISO/IEC 29341-6-13
UPnP HVAC_SetpointSchedule:1 Service	ISO/IEC 29341-6-14
UPnP TemperatureSensor:1 Service	ISO/IEC 29341-6-15
UPnP TemperatureSetpoint:1 Service	ISO/IEC 29341-6-16
UPnP HVAC_UserOperatingMode:1 Service	ISO/IEC 29341-6-17
UPnP BinaryLight:1 Device	ISO/IEC 29341-7-1
UPnP DimmableLight:1 Device	ISO/IEC 29341-7-2
UPnP Dimming:1 Service	ISO/IEC 29341-7-10
UPnP SwitchPower:1 Service	ISO/IEC 29341-7-11
UPnP InternetGatewayDevice:1 Device	ISO/IEC 29341-8-1
UPnP LANDevice:1 Device	ISO/IEC 29341-8-2
UPnP WANDevice:1 Device	ISO/IEC 29341-8-3
UPnP WANConnectionDevice:1 Device	ISO/IEC 29341-8-4
UPnP WLANAccessPointDevice:1 Device	ISO/IEC 29341-8-5
UPnP LANHostConfigManagement:1 Service	ISO/IEC 29341-8-10
UPnP Layer3Forwarding:1 Service	ISO/IEC 29341-8-11
UPnP LinkAuthentication:1 Service	ISO/IEC 29341-8-12
UPnP RadiusClient:1 Service	ISO/IEC 29341-8-13
UPnP WANCableLinkConfig:1 Service	ISO/IEC 29341-8-14
UPnP WANCommonInterfaceConfig:1 Service	ISO/IEC 29341-8-15
UPnP WANDSLLinkConfig:1 Service	ISO/IEC 29341-8-16
UPnP WANEthernetLinkConfig:1 Service	ISO/IEC 29341-8-17
UPnP WANIPConnection:1 Service	ISO/IEC 29341-8-18
UPnP WANPOTSLinkConfig:1 Service	ISO/IEC 29341-8-19
UPnP WANPPPConnection:1 Service	ISO/IEC 29341-8-20
UPnP WLANConfiguration:1 Service	ISO/IEC 29341-8-21
UPnP Printer:1 Device	ISO/IEC 29341-9-1
UPnP Scanner:1.0 Device	ISO/IEC 29341-9-2
UPnP ExternalActivity:1 Service	ISO/IEC 29341-9-10
UPnP Feeder:1.0 Service	ISO/IEC 29341-9-11
UPnP PrintBasic:1 Service	ISO/IEC 29341-9-12
UPnP Scan:1 Service	ISO/IEC 29341-9-13
UPnP QoS Architecture:1.0	ISO/IEC 29341-10-1
UPnP QoSDevice:1 Service	ISO/IEC 29341-10-10
UPnP QoSManager:1 Service	ISO/IEC 29341-10-11
UPnP QoSPolicyHolder:1 Service	ISO/IEC 29341-10-12
UPnP QoS Architecture:2	ISO/IEC 29341-11-1
UPnP QOS v2 Schema Files	ISO/IEC 29341-11-2

UPnP Document Title	ISO/IEC 29341 Part
UPnP QosDevice:2 Service	ISO/IEC 29341-11-10
UPnP QosManager:2 Service	ISO/IEC 29341-11-11
UPnP QosPolicyHolder:2 Service	ISO/IEC 29341-11-12
UPnP RemoteUIClientDevice:1 Device	ISO/IEC 29341-12-1
UPnP RemoteUIServerDevice:1 Device	ISO/IEC 29341-12-2
UPnP RemoteUIClient:1 Service	ISO/IEC 29341-12-10
UPnP RemoteUIServer:1 Service	ISO/IEC 29341-12-11
UPnP DeviceSecurity:1 Service	ISO/IEC 29341-13-10
UPnP SecurityConsole:1 Service	ISO/IEC 29341-13-11

1 Overview and Scope

This service definition is compliant with the UPnP Device Architecture version [1.0](#). It defines a service type referred to herein as ScheduledRecording service.

1.1 Introduction

The ScheduledRecording service is a UPnP service that allows control points to schedule the recording of content. Generally, this content is broadcast content, but this specification does not limit itself to broadcast content. This service type enables the following functions:

- Create a [recordSchedule](#) so that it is added to the list of [recordSchedule](#) instances. Each [recordSchedule](#) describes user-level recording instructions for the ScheduledRecording service.
- Browse a list of [recordSchedule](#) instances stored by the ScheduledRecording service.
- Delete a [recordSchedule](#) so that it is removed from the list of [recordSchedule](#) instances.
- Browse a list of [recordTask](#) instances, stored by the ScheduledRecording service. The ScheduledRecording service may create zero or more [recordTask](#) instances for each [recordSchedule](#). A [recordTask](#) represents a discrete recording operation of a [recordSchedule](#).
- Enable or disable individual [recordTask](#) instances.
- Enable or disable a [recordSchedule](#).
- Receive notifications indicating change of [recordSchedule](#) or [recordTask](#) list.

The ScheduledRecording service does not require a dependency on any UPnP services other than a co-located ContentDirectory service, which provides the following functions:

- A ContentDirectory service provides channel line-up to allow users to find recordable channels. A control point may use this metadata when creating a [recordSchedule](#) on a ScheduledRecording service.
- A ContentDirectory service may provide Electronic Program Guide (EPG) features to allow users to find recordable content. A control point may use this metadata when creating a [recordSchedule](#) on a ScheduledRecording service.
- Contents recorded by the ScheduledRecording service may be exposed by a ContentDirectory service.

The architectural relationship among the different concepts, defined by the ScheduledRecording service can be summarized as follows: A ScheduledRecording service owns a flat (that is: non-nested) list of [recordSchedule](#) instances, meaning that the ScheduledRecording service may create, destroy, or change [recordSchedule](#) instances. A [recordSchedule](#) represents user-level instructions to perform recording operations. Generally, a user constructs his instructions to a ScheduledRecording service via a control point that invokes UPnP actions that affect the list of [recordSchedule](#) instances. In all cases, the ScheduledRecording service MUST be able to describe discrete recording operations for a [recordSchedule](#) through a list of associated [recordTask](#) instances. A [recordTask](#) can only exist with a [recordSchedule](#) (that is: never orphaned). Thus when a [recordTask](#) is created by the ScheduledRecording service, its lifetime depends on its parent [recordSchedule](#). An individual [recordTask](#) can be selectively enabled or disabled.

This service template does not address:

- Implementations where the ScheduledRecording service and its associated ContentDirectory service are not co-located in the same device.

1.2 Notation

- In this document, features are described as Required, Recommended, or Optional as follows:

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this specification are to be interpreted as described in [RFC 2119].

In addition, the following keywords are used in this specification:

PROHIBITED – The definition or behavior is an absolute prohibition of this specification. Opposite of **REQUIRED**.

CONDITIONALLY REQUIRED – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is **REQUIRED**, otherwise it is **PROHIBITED**.

CONDITIONALLY OPTIONAL – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is **OPTIONAL**, otherwise it is **PROHIBITED**.

These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

- Strings that are to be taken literally are enclosed in “double quotes”.
- Words that are emphasized are printed in *italic*.
- Keywords that are defined by the UPnP AV Working Committee are printed using the *forum* character style.
- Keywords that are defined by the UPnP Device Architecture are printed using the *arch* character style.
- A double colon delimiter, “::”, signifies a hierarchical parent-child (parent::child) relationship between the two objects separated by the double colon. This delimiter is used in multiple contexts, for example: Service::Action(), Action()::Argument, parentProperty::childProperty.

1.2.1 Data Types

This specification uses data type definitions from two different sources. The UPnP Device Architecture defined data types are used to define state variable and action argument data types [DEVICE]. The XML Schema namespace is used to define property data types [XML SCHEMA-2].

For UPnP Device Architecture defined Boolean data types, it is strongly **RECOMMENDED** to use the value “0” for false, and the value “1” for true. However, when used as input arguments, the values “*false*”, “*no*”, “*true*”, “*yes*” may also be encountered and **MUST** be accepted. Nevertheless, it is strongly **RECOMMENDED** that all state variables and output arguments be represented as “0” and “1”.

For XML Schema defined Boolean data types, it is strongly **RECOMMENDED** to use the value “0” for false, and the value “1” for true. However, when used as input properties, the values “*false*”, “*true*” may also be encountered and **MUST** be accepted. Nevertheless, it is strongly **RECOMMENDED** that all properties be represented as “0” and “1”.

1.2.2 Strings Embedded in Other Strings

Some string variables and arguments described in this document contain substrings that **MUST** be independently identifiable and extractable for other processing. This requires the definition of appropriate substring delimiters and an escaping mechanism so that these delimiters can also appear as ordinary characters in the string and/or its independent substrings. This document uses embedded strings in two contexts – Comma Separated Value (CSV) lists (see Section 1.3.1, “Comma Separated Value (CSV) Lists”) and property values in search criteria strings. Escaping conventions use the backslash character, “\” (character code U+005C), as follows:

- a. Backslash (“\”) is represented as “\\” in both contexts.
- b. Comma (“,”) is
 1. represented as “\,” in individual substring entries in CSV lists
 2. not escaped in search strings
- c. Double quote (“””) is
 1. not escaped in CSV lists
 2. not escaped in search strings when it appears as the start or end delimiter of a property value
 3. represented as “\\” in search strings when it appears as a character that is part of the property value

1.2.3 Extended Backus-Naur Form

Extended Backus-Naur Form is used in this document for a formal syntax description of certain constructs. The usage here is according to the reference [EBNF].

1.2.3.1 Typographic conventions for EBNF

Non-terminal symbols are unquoted sequences of characters from the set of English upper and lower case letters, the digits “0” through “9”, and the hyphen (“-”). Character sequences between 'single quotes' are terminal strings and MUST appear literally in valid strings. Character sequences between (*comment delimiters*) are English language definitions or supplementary explanations of their associated symbols. White space in the EBNF is used to separate elements of the EBNF, not to represent white space in valid strings. White space usage in valid strings is described explicitly in the EBNF. Finally, the EBNF uses the following operators:

Table 1-1: EBNF Operators

Operator	Semantics
: :=	definition – the non-terminal symbol on the left is defined by one or more alternative sequences of terminals and/or non-terminals to its right.
	alternative separator – separates sequences on the right that are independently allowed definitions for the non-terminal on the left.
*	null repetition – means the expression to its left MAY occur zero or more times.
+	non-null repetition – means the expression to its left MUST occur at least once and MAY occur more times.
[]	optional – the expression between the brackets is optional.
()	grouping – groups the expressions between the parentheses.
-	character range – represents all characters between the left and right character operands inclusively.

1.3 Derived Data Types

This section defines a derived data type that is represented as a string data type with special syntax. This specification uses string data type definitions that originate from two different sources. The UPnP Device Architecture defined **string** data type is used to define state variable and action argument **string** data types. The XML Schema namespace is used to define property xsd:string data types. The following definition applies to both string data types.

1.3.1 Comma Separated Value (CSV) Lists

The UPnP AV services use state variables, action arguments and properties that represent lists – or one-dimensional arrays – of values. The UPnP Device Architecture, Version 1.0 [DEVICE], does not provide for either an array type or a list type, so a list type is defined here. Lists MAY either be homogeneous (all values are the same type) or heterogeneous (values of different types are allowed). Lists MAY also consist of repeated occurrences of homogeneous or heterogeneous subsequences, all of which have the same syntax and semantics (same number of values, same value types and in the same order). The data type of a homogeneous list is **string** or xsd:string and denoted by CSV (*x*), where *x* is the type of the individual values. The data type of a heterogeneous list is also **string** or xsd:string and denoted by CSV (*x*, *y*, *z*), where *x*, *y* and *z* are the types of the individual values. If the number of values in the heterogeneous list is too large to show each type individually, that variable type is represented as CSV (*heterogeneous*), and the variable description includes additional information as to the expected sequence of values appearing in the list and their corresponding types. The data type of a repeated subsequence list is **string** or xsd:string and denoted by CSV ({*x*, *y*, *z*}), where *x*, *y* and *z* are the types of the individual values in the subsequence and the subsequence MAY be repeated zero or more times.

- A list is represented as a **string** type (for state variables and action arguments) or xsd:string type (for properties).
- Commas separate values within a list.
- Integer values are represented in CSVs with the same syntax as the integer data type specified in [DEVICE] (that is: optional leading sign, optional leading zeroes, numeric ASCII)
- Boolean values are represented in state variable and action argument CSVs as either “0” for false or “1” for true. These values are a subset of the defined Boolean data type values specified in [DEVICE]: **0**, **false**, **no**, **1**, **true**, **yes**.

- Boolean values are represented in property CSVs as either “*0*” for false or “*1*” for true. These values are a subset of the defined Boolean data type values specified in [XML SCHEMA-2]: 0, false, 1, true.
- Escaping conventions for the comma and backslash characters are defined in Section 1.2.2, “Strings Embedded in Other Strings”.
- White space before, after, or interior to any numeric data type is not allowed.
- White space before, after, or interior to any other data type is part of the value.

Table 1-2: CSV Examples

Type refinement of string	Value	Comments
CSV (<i>string</i>) or CSV (xsd:string)	“+artist,-date”	List of 2 property sort criteria.
CSV (<i>int</i>) or CSV (xsd:integer)	“1,-5,006,0,+7”	List of 5 integers.
CSV (<i>boolean</i>) or CSV (xsd:Boolean)	“0,1,1,0”	List of 4 booleans
CSV (<i>string</i>) or CSV (xsd:string)	“Smith\, Fred,Jones\, Davey”	List of 2 names, “Smith, Fred” and “Jones, Davey”
CSV (<i>i4</i> , <i>string</i> , <i>ui2</i>) or CSV (xsd:int, xsd:string, xsd:unsignedShort)	“-29837, string with leading blanks,0”	Note that the second value is “ string with leading blanks”
CSV (<i>i4</i>) or CSV (xsd:int)	“3, 4”	Illegal CSV. White space is not allowed as part of an integer value.
CSV (<i>string</i>) or CSV (xsd:string)	“,”	List of 3 empty string values
CSV (heterogeneous)	“Alice,Marketing,5,Sue,R&D,21,Dave,Finance,7”	List of unspecified number of people and associated attributes. Each person is described by 3 elements: a name <i>string</i> , a department <i>string</i> and years-of-service <i>ui2</i> or a name xsd:string, a department xsd:string and years-of-service xsd:unsignedShort.

1.4 Management of XML Namespaces in Standardized DCPs

UPnP specifications make extensive use of XML namespaces. This allows separate DCPs, and even separate components of an individual DCP, to be designed independently and still avoid name collisions when they share XML documents. Every name in an XML document belongs to exactly one namespace. In documents, XML names appear in one of two forms: qualified or unqualified. An unqualified name (or no-colon-name) contains no colon (“:”) characters. An unqualified name belongs to the document’s default namespace. A qualified name is two no-colon-names separated by one colon character. The no-colon-name before the colon is the qualified name’s namespace prefix, the no-colon-name after the colon is the qualified name’s “local” name (meaning local to the namespace identified by the namespace prefix). Similarly, the unqualified name is a local name in the default namespace.

The formal name of a namespace is a URI. The namespace prefix used in an XML document is *not* the name of the namespace. The namespace name is, or should be, globally unique. It has a single definition that is accessible to anyone who uses the namespace. It has the same meaning anywhere that it is used, both inside and outside

XML documents. The namespace prefix, however, in formal XML usage, is defined only in an XML document. It must be locally unique to the document. Any valid XML no-colon-name may be used. And, in formal XML usage, no two XML documents are ever required to use the same namespace prefix to refer to the same namespace. The creation and use of the namespace prefix was standardized by the W3C XML Committee in [XML-NMSP] strictly as a convenient local shorthand replacement for the full URI name of a namespace in individual documents.

All AV object properties are represented in XML by element and attribute names, therefore, all property names belong to an XML namespace.

For the same reason that namespace prefixes are convenient in XML documents, it is convenient in specification text to refer to namespaces using a namespace prefix. Therefore, this specification declares a “standard” prefix for all XML namespaces used herein. In addition, this specification expands the scope where these prefixes have meaning, beyond a single XML document, to all of its text, XML examples, and certain string-valued properties. This expansion of scope *does not* supersede XML rules for usage in documents, it only augments and complements them in important contexts that are out-of-scope for the XML specifications.

All of the namespaces used in this specification are listed in the Tables “Namespace Definitions” and “Schema-related Information”. For each such namespace, Table 1-3, “Namespace Definitions” gives a brief description of it, its name (a URI) and its defined “standard” prefix name. Some namespaces included in these tables are not directly used or referenced in this document. They are included for completeness to accommodate those situations where this specification is used in conjunction with other UPnP specifications to construct a complete system of devices and services. The individual specifications in such collections all use the same standard prefix. The standard prefixes are also used in Table 1-4, “Schema-related Information”, to cross-reference additional namespace information. This second table includes each namespace’s valid XML document root elements (if any), its schema file name, versioning information (to be discussed in more detail below), and links to the entries in the Reference section for its associated schema.

The normative definitions for these namespaces are the documents referenced in Table 1-3. The schemas are designed to support these definitions for both human understanding and as test tools. However, limitations of the XML Schema language itself make it difficult for the UPnP-defined schemas to accurately represent all details of the namespace definitions. As a result, the schemas will validate many XML documents that are not valid according to the specifications.

The Working Committee expects to continue refining these schemas after specification release to reduce the number of documents that are validated by the schemas while violating the specifications, but the schemas will still be informative, supporting documents. Some schemas might become normative in future versions of the specifications.

Table 1-3: Namespace Definitions

Standard Name-space Prefix	Namespace Name	Namespace Description	Normative Definition Document Reference
<i>AV Working Committee defined namespaces</i>			
av:	urn:schemas-upnp-org:av:av	Common data types for use in AV schemas	[AV-XSD]
avs:	urn:schemas-upnp-org:av:avs	Common structures for use in AV schemas	[AVS-XSD]
avdt:	urn:schemas-upnp-org:av:avdt	Datastructure Template	[AVDT]
avt-event:	urn:schemas-upnp-org:metadata-1-0/AVT/	Evented <i>LastChange</i> state variable for AVTransport	[AVT]
didl-lite:	urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/	Structure and metadata for ContentDirectory	[CDS]
rscs-event:	urn:schemas-upnp-org:metadata-1-0/RCS/	Evented <i>LastChange</i> state variable for RenderingControl	[RCS]
srs:	urn:schemas-upnp-org:av:srs	Metadata and structure for ScheduledRecording	[SRS]
srs-event:	urn:schemas-upnp-org:av:srs-event	Evented <i>LastChange</i> state variable for ScheduledRecording	[SRS]

Standard Name-space Prefix	Namespace Name	Namespace Description	Normative Definition Document Reference
upnp:	urn:schemas-upnp-org:metadata-1-0/upnp/	Metadata for ContentDirectory	[CDS]
<i>Externally defined namespaces</i>			
dc:	http://purl.org/dc/elements/1.1/	Dublin Core	[DC-TERMS]
xsd:	http://www.w3.org/2001/XMLSchema	XML Schema Language 1.0	[XML SCHEMA-1] [XML SCHEMA-2]
xsi:	http://www.w3.org/2001/XMLSchema-instance	XML Schema Instance Document schema	Sections 2.6 & 3.2.7 of [XML SCHEMA-1]
xml:	http://www.w3.org/XML/1998/namespace	The “xml:” Namespace	[XML-NS]

Table 1-4: Schema-related Information

Standard Name-space Prefix	Relative URI and File Name • Form 1 • Form 2	Valid Root Element(s)	Schema Reference
<i>AV Working Committee Defined Namespaces</i>			
av:	- av-vn-yyyymmdd.xsd - av-vn.xsd	n/a	[AV-XSD]
avs:	- avs-vn-yyyymmdd.xsd - avs-vn.xsd	<Features> <stateVariableValuePairs>	[AVS-XSD]
avdt:	- avdt-vn-yyyymmdd.xsd - avdt-vn.xsd	<AVDT>	[AVDT]
avt-event:	- avt-event-vn-yyyymmdd.xsd - avt-event-vn.xsd	<Event>	[AVT-EVENT-XSD]
didl-lite:	- didl-lite-vn-yyyymmdd.xsd - didl-lite-vn.xsd	<DIDL-Lite>	[DIDL-LITE-XSD]
rce-event:	- rce-event-vn-yyyymmdd.xsd - rce-event-vn.xsd	<Event>	[RCS-EVENT-XSD]
srs:	- srs-vn-yyyymmdd.xsd - srs-vn.xsd	<srs>	[SRS-XSD]
srs-event:	- srs-event-vn-yyyymmdd.xsd - srs-event-vn.xsd	<StateEvent>	[SRS-EVENT-XSD]
upnp:	- upnp-vn-yyyymmdd.xsd - upnp-vn.xsd	n/a	[UPNP-XSD]
<i>Externally Defined Namespaces</i>			
dc:	Absolute URL: http://dublincore.org/schemas/xmls/simpledc20021212.xsd		[DC-XSD]
xsd:	n/a	<schema>	[XMLSCHEMA-XSD]
xsi:	n/a		n/a
xml:	n/a		[XML-XSD]

1.4.1 Namespace Prefix Requirements

There are many occurrences in this specification of string data types that contain XML names (property names). These XML names in strings will not be processed under namespace-aware conditions. Therefore, all occurrences in instance documents of XML names in strings MUST use the standard namespace prefixes as declared in Table 1-3. In order to properly process the XML documents described herein, control points and devices MUST use namespace-aware XML processors [XML-NMSP] for both reading and writing. Also, all

by [XML-NMSP], the namespace prefixes used in an instance document are at the sole discretion of the document creator. Therefore, the declared prefix for a namespace in a document MAY be different from the standard prefix. All devices MUST be able to correctly process any valid XML instance document, even when it uses a non-standard prefix for ordinary XML names. It is strongly RECOMMENDED that all devices use these standard prefixes for all instance documents to avoid confusion on the part of both human and machine readers. These standard prefixes are used in all descriptive text and all XML examples in this and related UPnP specifications. Also, each individual specification may assume a default namespace for its descriptive text. In that case, names from that namespace may appear with no prefix.

The assumed default namespace, if any, for each UPnP AV specification is given in Table 1-5, “Default Namespaces for the AV Specifications”.

Note: all UPnP AV schemas declare attributes to be “unqualified”, so namespace prefixes are never used with AV Working Committee defined attribute names.

Table 1-5: Default Namespaces for the AV Specifications

AV Specification Name	Default Namespace Prefix
AVTransport:2	avt-event:
ConnectionManager:2	<i>n/a</i>
ContentDirectory:2	didl-lite:
MediaRenderer:2	<i>n/a</i>
MediaServer:2	<i>n/a</i>
RenderingControl:2	rcs-event:
ScheduledRecording:1	srs:

1.4.2 Namespace Names, Namespace Versioning and Schema Versioning

Each namespace that is defined by the AV Working Committee is named by a URN.

In order to enable both forward and backward compatibility, the UPnP TC has established the general policy that namespace names will not change with new versions of specifications, even when the specification changes the definition of a namespace. But, namespaces still have version numbers that reflect definitional changes. Each time the definition of a namespace is changed, the namespace’s version number is incremented by one.

Therefore, namespace version information must be provided with each XML instance document so that the document’s receiver can properly understand its meaning. This is achieved by the following rules:

- Every release of a schema is identified by a version number and date of the form “*n-yyyymmdd*”, where *n* corresponds to the namespace definition version number and *yyyymmdd* is the year, month and day in the Gregorian calendar that the schema is released.

For example, the new version numbers of the pre-existing “DIDL-Lite” and “upnp” schemas are “2”. Versions for new schemas, such as “srs” are “1”.

For each schema, the version-date will appear in two places:

1. In the schema file name, according to the naming structure shown in Table 1-4, “Schema-related Information”.
2. As the value of the `version` attribute of each schema’s schema root element.

Namespaces are referenced in both schema and XML instance documents by namespace name. The namespace name appears as the value of an `xmlns` attribute. The `xmlns` attribute also declares a namespace prefix that will be used to qualify names from each namespace. Schemas are referenced in both schema and XML instance documents by URI in the `schemaLocation` attribute. See section 1.4.3, “Namespace Usage Examples”. Two different forms of URI are available, each with a different meaning. All UPnP AV-defined schema URIs share a common base path of “`http://www.upnp.org/schemas/av/`”. Each schema URI has two unique relative forms (see Table 1-4, “Schema-related Information”), according to which version of a namespace and its representative schema is of interest. The allowed relative URI forms are:

1. *schema-root-name* “-v” *version-date*
where *version-date* is a full version-date of the form *n-yyyymmdd*. This form references the schema

whose “root” name (typically the standardized prefix name used for the namespace that the schema represents) and version-date match *schema-root-name* and *version-date*, respectively.

2. *schema-root-name* “-v” *version*

where *version* is an integer representing the namespace’s version number. This form references the most recent version of the schema whose root name and namespace version number match *schema-root-name* and the *version*, respectively.

Usage rules for schema location URIs are as follows:

- All instance documents, whether generated by a service or a control point, MUST use Form 1.
- All UPnP AV published schemas that reference other UPnP AV schemas will also use Form 1.
- Validation of XML instance documents in UPnP AV systems potentially serves two purposes. The first is based on standard XML and XML Schema semantics: the document’s creator asserts that the document is syntactically correct with respect to the referenced schema. The receiving processor can confirm this with a validating parser that uses the referenced schema(s). The second is based on UPnP AV namespace semantics. The receiving processor knows that the XML instance document is supposed to conform to one or more specific UPnP AV specifications. Since the second context is actually the more important context for instance document processing, the receiving processor MAY validate the instance document against any version of a schema that satisfies its needs in assessing the acceptability of the received instance document.

1.4.3 Namespace Usage Examples

The `schemaLocation` attribute for XML instance documents comes from the XML Schema instance namespace “`http://www.w3.org/2002/XMLSchema-instance`”. A single occurrence of the attribute can declare the location of one or more schemas. The `schemaLocation` attribute value consists of a whitespace separated list of values: namespace name followed by its schema location URL. This pair-sequence is repeated as necessary for the schemas that need to be located for this instance document.

Example 1:

Sample *DIDL-Lite XML Document*. This document assumes version-date 2-20060531 of the “`didl-lite:`” namespace/schema combination and (a possible later) version 2-20061231 of “`upnp:`”. The lines with the gray background show how to express this versioning information in the instance document.

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
  xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
    http://www.upnp.org/schemas/av/didl-lite-v2-20060531.xsd
    urn:schemas-upnp-org:metadata-1-0/upnp/
    http://www.upnp.org/schemas/av/upnp-v2-20061231.xsd">
  <item id="18" parentID="13" restricted="0">
    ...
  </item>
</DIDL-Lite>
```

Example 2:

Sample *srs XML Document*. This document assumes version 1-20060531 of the “`srs:`” namespace/schema combination. Again, the lines with the gray background show how to express this versioning information in the instance document.

```
<?xml version="1.0" encoding="UTF-8"?>
<srs
  xmlns="urn:schemas-upnp-org:av:srs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    urn:schemas-upnp-org:av:srs
    http://www.upnp.org/schemas/av/srs-v1-20060531.xsd">
```

```

...
</srs>

```

1.5 Vendor-defined Extensions

Whenever vendors create additional vendor-defined state variables, actions or properties, their assigned names and XML representation MUST follow the naming conventions and XML rules as specified in [DEVICE], Section 2.5, “Description: Non-standard vendor extensions”.

1.6 References

This section lists the normative references used in the UPnP AV specifications and includes the tag inside square brackets that is used for each such reference:

[AVARCH] – *AVArchitecture:1*, UPnP Forum, June 25, 2002.

Available at: <http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v1-20020625.pdf>.

[AVDT] – *AV DataStructure Template:1*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/specs/av/UPnP-av-AVDataStructure-v1-20060531.pdf>.

Latest version available at: <http://www.upnp.org/specs/av/UPnP-av-AVDataStructure-v1.pdf>.

[AVDT-XSD] – *XML Schema for UPnP AV Datastructure Template:1*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/schemas/av/avdt-v1-20060531.xsd>.

Latest version available at: <http://www.upnp.org/schemas/av/avdt-v1.xsd>.

[AV-XSD] – *XML Schema for UPnP AV Common XML Data Types*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/schemas/av/av-v1-20060531.xsd>.

Latest version available at: <http://www.upnp.org/schemas/av/av-v1.xsd>.

[AVS-XSD] – *XML Schema for UPnP AV Common XML Structures*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/schemas/av/avs-v1-20060531.xsd>.

Latest version available at: <http://www.upnp.org/schemas/av/avs-v1.xsd>.

[AVT] – *AVTransport:2*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/specs/av/UPnP-av-AVTransport-v2-Service-20060531.pdf>.

Latest version available at: <http://www.upnp.org/specs/av/UPnP-av-AVTransport-v2-Service.pdf>.

[AVT-EVENT-XSD] – *XML Schema for AVTransport:2 LastChange Eventing*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/schemas/av/avt-event-v2-20060531.xsd>.

Latest version available at: <http://www.upnp.org/schemas/av/avt-event-v2.xsd>.

[CDS] – *ContentDirectory:2*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v2-Service-20060531.pdf>.

Latest version available at: <http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v2-Service.pdf>.

[CM] – *ConnectionManager:2*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v2-Service-20060531.pdf>.

Latest version available at: <http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v2-Service.pdf>.

[DC-XSD] – *XML Schema for UPnP AV Dublin Core*.

Available at: <http://www.dublincore.org/schemas/xmls/simpledc20020312.xsd>.

[DC-TERMS] – *DCMI term declarations represented in XML schema language*.

Available at: <http://www.dublincore.org/schemas/xmls>.

[DEVICE] – *UPnP Device Architecture, version 1.0*, UPnP Forum, June 13, 2000.

Available at: <http://www.upnp.org/specs/architecture/UPnP-DeviceArchitecture-v1.0-20000613.htm>.

Latest version available at: <http://www.upnp.org/specs/architecture/UPnP-DeviceArchitecture-v1.0.htm>.

[DIDL] – ISO/IEC CD 21000-2:2001, *Information Technology - Multimedia Framework - Part 2: Digital Item Declaration*, July 2001.

[DIDL-LITE-XSD] – *XML Schema for ContentDirectory:2 Structure and Metadata (DIDL-Lite)*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/schemas/av/didl-lite-v2-20060531.xsd>.

Latest version available at: <http://www.upnp.org/schemas/av/didl-lite-v2.xsd>.

[EBNF] – ISO/IEC 14977, *Information technology - Syntactic metalanguage - Extended BNF*, December 1996.

[HTTP/1.1] – *HyperText Transport Protocol – HTTP/1.1*, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999.
Available at: <http://www.ietf.org/rfc/rfc2616.txt>.

IEC 61883] – *IEC 61883 Consumer Audio/Video Equipment – Digital Interface - Part 1 to 5*.
Available at: <http://www.iec.ch>.

[IEC-PAS 61883] – *IEC-PAS 61883 Consumer Audio/Video Equipment – Digital Interface - Part 6*.
Available at: <http://www.iec.ch>.

[ISO 8601] – *Data elements and interchange formats – Information interchange -- Representation of dates and times*, International Standards Organization, December 21, 2000.
Available at: <ISO 8601:2000>.

[MIME] – *IETF RFC 1341, MIME (Multipurpose Internet Mail Extensions)*, N. Borenstein, N. Freed, June 1992.
Available at: <http://www.ietf.org/rfc/rfc1341.txt>.

[MR] – *MediaRenderer:2*, UPnP Forum, May 31, 2006.
Available at: <http://www.upnp.org/specs/av/UPnP-av-MediaRenderer-v2-Device-20060531.pdf>.
Latest version available at: <http://www.upnp.org/specs/av/UPnP-AV-MediaRenderer-v2-Device.pdf>.

[MS] – *MediaServer:2*, UPnP Forum, May 31, 2006.
Available at: <http://www.upnp.org/specs/av/UPnP-av-MediaServer-v2-Device-20060531.pdf>.
Latest version available at: <http://www.upnp.org/specs/av/UPnP-AV-MediaServer-v2-Device.pdf>.

[RCS] – *RenderingControl:2*, UPnP Forum, May 31, 2006.
Available at: <http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v2-Service-20060531.pdf>.
Latest version available at: <http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v2-Service.pdf>.

[RCS-EVENT-XSD] – *XML Schema for RenderingControl:2 LastChange Eventing*, UPnP Forum, May 31, 2006.
Available at: <http://www.upnp.org/schemas/av/rcs-event-v1-20060531.xsd>.
Latest version available at: <http://www.upnp.org/schemas/av/rcs-event-v1.xsd>.

[RFC 1738] – *IETF RFC 1738, Uniform Resource Locators (URL)*, Tim Berners-Lee, et. Al., December 1994.
Available at: <http://www.ietf.org/rfc/rfc1738.txt>.

[RFC 2119] – *IETF RFC 2119, Key words for use in RFCs to Indicate Requirement Levels*, S. Bradner, 1997.
Available at: <http://www.faqs.org/rfcs/rfc2119.html>.

[RFC 2396] – *IETF RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax*, Tim Berners-Lee, et al, 1998.
Available at: <http://www.ietf.org/rfc/rfc2396.txt>.

[RFC 3339] – *IETF RFC 3339, Date and Time on the Internet: Timestamps*, G. Klyne, Clearswift Corporation, C. Newman, Sun Microsystems, July 2002.
Available at: <http://www.ietf.org/rfc/rfc3339.txt>.

[RTP] – *IETF RFC 1889, Realtime Transport Protocol (RTP)*, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, January 1996.
Available at: <http://www.ietf.org/rfc/rfc1889.txt>.

[RTSP] – *IETF RFC 2326, Real Time Streaming Protocol (RTSP)*, H. Schulzrinne, A. Rao, R. Lanphier, April 1998.
Available at: <http://www.ietf.org/rfc/rfc2326.txt>.

[SRS] – *ScheduledRecording:1*, UPnP Forum, May 31, 2006.
Available at: <http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v1-Service-20060531.pdf>.
Latest version available at: <http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v1-Service-20060531.pdf>.

[SRS-XSD] – *XML Schema for ScheduledRecording:1 Metadata and Structure*, UPnP Forum, May 31, 2006.
Available at: <http://www.upnp.org/schemas/av/srs-v1-20060531.xsd>.
Latest version available at: <http://www.upnp.org/schemas/av/srs-v1.xsd>.

[SRS-EVENT-XSD] – *XML Schema for ScheduledRecording:1 LastChange Eventing*, UPnP Forum, May 31, 2006.
Available at: <http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd>.
Latest version available at: <http://www.upnp.org/schemas/av/srs-event-v1.xsd>.

[UAX 15] – *Unicode Standard Annex #15, Unicode Normalization Forms, version 4.1.0, revision 25*, M. Davis, M. Dürst, March 25, 2005.

Available at: <http://www.unicode.org/reports/tr15/tr15-25.html>.

[UNICODE COLLATION] – *Unicode Technical Standard #10, Unicode Collation Algorithm version 4.1.0*, M. Davis, K. Whistler, May 5, 2005.

Available at: <http://www.unicode.org/reports/tr10/tr10-14.html>.

[UPNP-XSD] – *XML Schema for ContentDirectory:2 Metadata*, UPnP Forum, May 31, 2006.

Available at: <http://www.upnp.org/schemas/av/upnp-v2-20060531.xsd>.

Latest version available at: <http://www.upnp.org/schemas/av/upnp-v2.xsd>.

[UTS 10] – *Unicode Technical Standard #10, Unicode Collation Algorithm, version 4.1.0, revision 14*, M. Davis, K. Whistler, May 5, 2005.

Available at: <http://www.unicode.org/reports/tr10/tr10-14.html>.

[UTS 35] – *Unicode Technical Standard #35, Locale Data Markup Language, version 1.3R1, revision 5*, M. Davis, June 2, 2005.

Available at: <http://www.unicode.org/reports/tr35/tr35-5.html>.

[XML] – *Extensible Markup Language (XML) 1.0 (Third Edition)*, François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, eds., W3C Recommendation, February 4, 2004.

Available at: <http://www.w3.org/TR/2004/REC-xml-20040204>.

[XML-NS] – *The “xml:” Namespace*, November 3, 2004.

Available at: <http://www.w3.org/XML/1998/namespace>.

[XML-XSD] – *XML Schema for the “xml:” Namespace*.

Available at: <http://www.w3.org/2001/xml.xsd>.

[XML-NMSP] – *Namespaces in XML*, Tim Bray, Dave Hollander, Andrew Layman, eds., W3C Recommendation, January 14, 1999.

Available at: <http://www.w3.org/TR/1999/REC-xml-names-19990114>.

[XML SCHEMA-1] – *XML Schema Part 1: Structures, Second Edition*, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, W3C Recommendation, 28 October 2004.

Available at: <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[XML SCHEMA-2] – *XML Schema Part 2: Data Types, Second Edition*, Paul V. Biron, Ashok Malhotra, W3C Recommendation, 28 October 2004.

Available at: <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

[XMLSCHEMA-XSD] – *XML Schema for XML Schema*.

Available at: <http://www.w3.org/2001/XMLSchema.xsd>.

2 Service Modeling Definitions

2.1 ServiceType

The following service type identifies a service that is compliant with this template:

urn:schemas-upnp-org:service: ScheduledRecording:1

ScheduledRecording service is used herein to refer to this service type.

2.2 Terms and Abbreviations

2.2.1 Abbreviations

Table 2-1: Abbreviations

Definition	Description
CDS	ContentDirectory Service.
EPG	Electronic Program Guide.
SRS	ScheduledRecording service.

2.2.2 Terms

2.2.2.1 CDS object

An object in a ContentDirectory service metadata hierarchy; that is: item or container.

2.2.2.2 User Channel

A User Channel is a ContentDirectory service object that exposes the (continuous) content stream of a particular broadcast channel. Usually, the actual channel that the User Channel exposes is determined by the user through some device-specific interaction. Examples are: manual programming of a number of channel presets; invoking of the auto-scan functionality of a device; predefined fixed channel assignments by the device manufacturer.

2.2.2.3 Channel Group

A Channel Group is a ContentDirectory service container that holds a number of User Channel items. Typically, a Channel Group contains User Channel items that are bound to a particular hardware resource. Examples include: a single analog cable TV tuner, a HDTV digital tuner, an AM/FM radio tuner, etc.

2.2.2.4 Channel Line-up

A service provider-generated list of channels with their associated content provider.

2.2.2.5 object

A recordSchedule or a recordTask (see definition of recordSchedule and recordTask below).

2.2.2.6 class

As defined in the ContentDirectory service specification, a class is used to assign a type to an object. It also identifies the minimum REQUIRED set of properties that MUST be present on that object and the OPTIONAL properties that MAY be present. Classes are organized in a hierarchy with certain classes being derived from others as in a typical object-oriented system. This specification defines two base classes (recordSchedule and recordTask) from which all other classes are derived.

2.2.2.7 object Modification

An object is considered modified when one of its properties (or its list of properties) is modified; that is: added, removed or changed in value (see definition of property below).

2.2.2.8 recordSchedule

A ScheduledRecording service construct that represents a complete set of recording instructions to the service, which allows the service to generate recordTask objects as necessary to record the desired content. The creator of the recordSchedule object assigns it a specific class, based on the type and complexity of the instructions, used to identify the content.

A recordSchedule is represented in XML as an `<item>...</item>` element.

2.2.2.9 Conflicting recordSchedule

A conflicting recordSchedule exists when one or more of its associated recordTask instances is in conflict with another recordTask instance.

2.2.2.10 recordTask

A ScheduledRecording service construct that represents a discrete recording operation of the underlying recording system. A recordTask is created by its parent recordSchedule and can not be directly created by the user. The parent-child relationship of recordSchedule and recordTask can be 1-to-zero or more.

A recordTask is represented in XML as an `<item>...</item>` element.

2.2.2.11 Conflicting recordTask

A conflicting recordTask exists when it overlaps in time with one or more other recordTask instances and the ScheduledRecording service has insufficient resources to record all of them. Existing pre-roll and post-roll adjustments (as defined by the scheduledStartDateTimeAdjust and scheduledDurationAdjust properties) MUST be taken into account when determining conflicts.

2.2.2.12 recordScheduleParts

A ScheduledRecording service construct that represents user-level recording instructions to the service, which provide a template to generate complete recordSchedule objects. The creator of the recordScheduleParts object assigns it a specific class, based on the type and complexity of the instructions, used to identify the content.

A recordScheduleParts is represented in XML as an `<item>...</item>` element.

2.2.2.13 Property-set Data Types

Certain ScheduledRecording service actions use *property-set* arguments that contain information about a set of properties, typically expressed in the form of an *srs XML Document* (for example, the Elements argument of the CreateRecordSchedule() action). The set of properties that can exist in a *property-set* argument is implementation dependent. Indeed, the set of optional properties that a particular ScheduledRecording service chooses to implement is vendor dependent.

This specification currently defines three different *property-set* data types:

- A_ARG_TYPE_RecordSchedule
- A_ARG_TYPE_RecordTask
- A_ARG_TYPE_RecordScheduleParts

Although these three types are different, they are very similar in nature and are defined using the same SRS schema [SRS-XSD], which defines all the properties that can ever occur in any of the three *property-set* data types. They differ only in the set of properties that can appear in them and in the values that are allowed for these properties.

2.2.2.14 Property

A property in the ScheduledRecording service represents a characteristic of an object. Properties are distinguished by their names. The ScheduledRecording service defines two kinds of properties – independent and dependent. Each independent property has zero or more dependent properties associated with it. Independent property names contain no “@” symbol; they may contain an XML namespace prefix (see below for an explanation of the relationship between properties and XML). Each dependent property is associated either with exactly one independent property or directly with a ScheduledRecording service class. The name of a dependent property that is associated with an independent property is the concatenation of three parts: its associated independent property name, the “@” symbol, and a name for the relationship between the two properties’ values. The name of a dependent property that is associated directly with a class is just the “@” symbol followed by the relationship name. Their data types and meanings are defined in Annex B.

Even though ScheduledRecording service properties are not XML objects, XML is used to express them in all exchanges between a control point and a ScheduledRecording service implementation. This creates an unavoidable relationship between XML syntax and property names and values. In XML, an independent property is represented as an element. The property name is used as the element name. The property value is the element content. A dependent property is represented as an attribute in XML. The dependent property’s relationship name is used as the attribute name. The dependent property’s value is the attribute value. For dependent properties that are associated with an independent property, the attribute appears in the start tag of the element that represents its associated independent property. For dependent properties that are associated directly with a class, the attribute appears in the top-level start tag for each object of that class.

Examples:

Table 2-1: Properties in XML

Property Name	XML Representation (srs declared as default namespace)
<u>title</u>	<title>...</title>
<u>taskProgramCode</u>	<taskProgramCode>...</taskProgramCode>
<u>taskProgramCode@type</u>	<taskProgramCode type="...">...</taskProgramCode>
<u>@id</u>	<item id="...">...</item>

2.2.2.15 Member Property

A property is a member of a particular class when the property is declared to be either REQUIRED or OPTIONAL for that class.

2.2.2.16 Supported Member Property

A supported member property is a member property that is supported by a particular ScheduledRecording service implementation, according to the information returned by the [GetPropertyList\(\)](#) action.

2.2.2.17 Multi-valued property

Some independent properties are multi-valued. This means that the property MAY occur more than once in an object.

2.2.2.18 Single-valued property

Most independent properties are single-valued. This means that the property MUST occur at most once in an object. Some single-valued properties can contain a CSV list of values. A dependent property is always considered single-valued, because it can occur at most once with each occurrence of its associated independent property, even though the independent property may be multi-valued.

2.2.2.19 XML Document

A string that represents a valid XML 1.0 document according to a specific schema. Every occurrence of the phrase “XML Document” is preceded by the appropriate root element name, italicized, as listed in column 3, “Valid Root Element(s)” of Table 1-4, “Schema-related Information”.

For example, the phrase “*srs XML Document*” refers to an XML document based on the SRS Schema as defined in [SRS-XSD]. Such a document comprises a single `<srs ...>` root element, optionally preceded by the XML declaration: `<?xml version="1.0" ...?>`.

Therefore, the string containing the *srs XML Document* will have one of the following two forms:

“`<srs ...>...</srs>`”

or

“`<?xml ...?>
<srs ...>...</srs>`”

2.2.2.20 XML Fragment

An *XML Fragment* is a sequence of XML elements that are valid direct or indirect child elements of the root element according to a specific schema. Every occurrence of the phrase “*XML Fragment*” is preceded by the appropriate root element name, italicized, as listed in column 3, “Valid Root Element(s)” of Table 1-4, “Schema-related Information”.

For example, the phrase “*srs XML Fragment*” refers to a sequence of XML elements that are defined in the SRS Schema as defined in [SRS-XSD]:

“`<item id="..." ...>...</item>`”

or

“`<recordDestination mediaType="..." preference="...">`

...

`</recordDestination>`”

or

“`<title>...</title>
<class>...</class>
<...>...</...>
...
<...>...</...>`”

2.2.2.21 actualScheduledStartTime

The actual scheduled start date&time of a program item is defined as:

$actualScheduledStartTime = \text{scheduledStartTime} + \text{scheduledStartTimeAdjust}$

where *scheduledStartTime* is the scheduled broadcast start date&time of the program item and *scheduledStartTimeAdjust* is a user-supplied adjustment to that date&time, for example for pre-roll purposes.

2.2.2.22 actualStartTime

The actual start date&time of a program item is defined as:

$actualStartTime = actualScheduledStartTime + \text{any device-specific record startup latency.}$

2.2.2.23 actualScheduledEndTime

The actual scheduled end time of a program item is defined as:

$actualScheduledEndTime = \text{scheduledStartTime} + \text{scheduledDuration} + \text{scheduledDurationAdjust}$

where *scheduledStartTime* is the scheduled broadcast start date&time of the program item, *scheduledDuration* is the scheduled broadcast duration of the program item and *scheduledDurationAdjust* is a user-supplied adjustment to that duration, for example to select just a part of the program for recording.

2.2.2.24 actualEndTime

The actual end date&time of a program item is defined as:

$actualEndTime = actualScheduledEndTime + \text{any device-specific record teardown latency.}$

2.2.2.25 actualScheduledDuration

The actual scheduled duration of a program item is defined as:

$$\begin{aligned} \text{actualScheduledDuration} &= \text{actualScheduledEndTime} - \text{actualScheduledStartTime} \\ &= \text{scheduledDuration} + \text{scheduledDurationAdjust} - \\ &\quad \text{scheduledStartTimeAdjust} \end{aligned}$$

where scheduledDuration is the scheduled broadcast duration of the program item, scheduledDurationAdjust is a user-supplied adjustment to that duration, and scheduledStartTimeAdjust is a user-supplied adjustment to the scheduled start date&time.

2.2.2.26 Lexical Sort Order

Lexical sort order refers to string sorting – also called collation – based on language and regional conventions. It is *not* based on the binary codes of the characters in strings. Furthermore, lexical sorting is not based on character sets; a single character set may have multiple sort orders, again according to language and regional conventions. It is also possible to have lexical sorts that are further refined according to user preference. For a complete discussion of this topic see [UTS 10], and the related standards [UAX 15] and [UTS 35]. [UTS 10] defines the lexical sort algorithms. It uses a secondary algorithm defined in [UAX 15] and supporting data tables defined in [UTS 35]. These three references together – [UAX 15], [UTS 10] and [UTS 35] – should be sufficient to implement a robust lexical sort.

Simple example: one of the most familiar examples is case-insensitive sorting on the ASCII subset of Unicode. In a binary ASCII sort, all lower case letters sort after the upper case “Z” because “Z” has a character code of 0x5A, and all lower case character codes are greater than or equal to 0x61.

More complex example: the “ö” character in German sorts between “n” and “p” characters whereas in Swedish, it sorts after “z”.

2.2.2.27 Lexical Matching

Lexical matching compares two (sub)strings for equality under certain lexical sorting conditions. *It is important to note* that equality in lexical matching is often less restrictive than equality in lexical sorting. In other words, two strings that are equal under a lexical sort will always be a lexical match. However, two strings that are a lexical match might not be equal under a lexical sort for the same language and region. In some cases, an implementation’s lexical sort might consider all alphabetic characters with diacritical marks (accents, umlauts, circumflexes, etc.) to be distinct, yet the same implementation might ignore diacritical marks in lexical matching. For example, the strings “resumé”, “resume” and “résumé” might sort as “resume” < “resumé” < “résumé”, but when a lexical match using the string “resume”, might find all three strings “resumé”, “resume” and “résumé”. For implementation techniques, see [UTS 10] Section 8, “Searching and Matching”.

2.2.2.28 Simple Non-case-sensitive Sort Order

A simple non-case-sensitive sort order applies only to Roman alphabetic characters. All lower case ASCII alphabetic characters **MUST** sort the same as their uppercase equivalent, except when compared directly with their upper case equivalent, in which case the upper case character **SHOULD** sort before its lower case equivalent. This means that of the following three ordering relations, #1 **MUST** be true, at least one of #2 and #3 **MUST** be true, and #2 **SHOULD** be true.

“A” ≤ “a” < “B” ≤ “b” < ... < “Y” ≤ “y” < “Z” ≤ “z”

“A” < “a” < “B” < “b” < ... < “Y” < “y” < “Z” < “z”

“A” = “a” < “B” = “b” < ... < “Y” = “y” < “Z” = “z”

Additionally, the same upper and lower case relationships **SHOULD** hold for non-ASCII Roman alphabetic characters. That is, lower case alphabetic characters with diacritical marks **SHOULD** sort as their upper case equivalent, except when compared directly with their upper case equivalent, in which case the upper case character should sort before its lower case equivalent. The ordering relation between ASCII and non-ASCII alphabetic characters is left unspecified. Also, the ordering relation between non-ASCII alphabetic characters that are not upper or lower case equivalents of each other is left unspecified. This may be summarized in the following relations. In each, the letter “c” represents any non-ASCII Roman alphabetic character. #4 **SHOULD** be true for all “c”. #5 **SHOULD** be true for all “c”. If #5 is false for any “c”, it should be false for all “c” and #6 **SHOULD** be true for all “c”.

$\text{upper}(c) \leq \text{lower}(c)$

$\text{upper}(c) < \text{lower}(c)$

$\text{upper}(c) = \text{lower}(c)$

2.2.2.29 Simple Non-case-sensitive Matching

In a simple non-case-sensitive match, relation #0 above MUST be true, and relation #0 above SHOULD be true.

2.2.2.30 Numeric Sort Order

A sort order in which values are compared numerically. If the type of an individual value is numeric, the numeric value is used. If the type of an individual value is string, the string is converted to a number and that numeric value is used.

2.2.2.31 Boolean Sort Order

Boolean values are sorted with “0” (false) being less than “1” (true).

2.2.2.32 Sequenced Sort

A sequenced sort is a sort applied to a set of values, each of which is composed of a sequence of subvalues. The sequence is often in a CSV list, but there are other kinds of sequences used for sorting in this specification. The sequenced sort starts by sorting based on the first subvalue in the sequence. If all values differ in the first subvalue, the sort is finished. Otherwise, each subset of equal subvalues is then sorted based on the next subvalue in the sequence. This process repeats iteratively until there are no more subsets of equal subvalues or the sequence is exhausted.

2.2.2.33 Sequenced Lexical Sort

A sequenced sort in which all subvalues are strings and the subvalues are compared lexically.

2.2.2.34 Sequenced Numeric Sort

A sequenced sort in which each subvalue is either a number or the number represented by a string.

2.2.2.35 Lexical Numeric Sort

A lexical numeric sort is one where one or more substrings are known to represent numbers. The strings are then sorted using a sequenced sort, where the sequence is composed of the sequence of non-numeric and numeric substrings from the larger string.

For example, assume a property has the form <letter>-<number>, where <number> ranges from 1 to 10. In a straight ascending lexical sort, the values “A-10”, “A-1”, “A-2” would sort as: “A-1”, “A-10”, “A-2”. “A-10” sorts before “A-2” because they are equal in the first two character positions, but in the third position, “1” < “2”. However, in a lexical numeric sort, each string is considered to be a sequence of a letter and number separated by a hyphen. These values then sort as “A-1”, “A-2”, “A-10” because all three are equal in the first subvalue, “A”, but the second subvalue sorts as 1, 2, 10 in numeric order.

2.2.2.36 **type** Relationship Sort

This is a sort defined exclusively for independent properties that have a dependent property relationship named “**type**”. These properties are sorted as a sequence of two subvalues: the first subvalue is the value of the property’s **xxx@type** dependent property; the second subvalue is the value of the independent property **xxx** itself. The **xxx@type** subvalues are sorted as specified for the dependent **xxx@type** property in its own section. The independent property subvalues are sorted according to the order specified in its section. Sorting of the independent property may vary with the value of the dependent property.

2.3 ScheduledRecording Service Architecture

2.3.1 recordSchedule

A ScheduledRecording service implementation has a single, flat list of recordSchedule instances. A recordSchedule represents the user-level recording instructions to the ScheduledRecording service. These user-level instructions have various levels of complexity. For example, a simple instruction may state: “record channel 15 at 4PM on March 19, 2004,” while a more complex instruction may state: “record all episodes of the *DIY Home Improvement Show* on any channel that has the show for the next month.” The behavior of a recordSchedule is described by one or more properties, and these properties can be manipulated through several actions.

As shown in Figure 1, when a control point requests a new scheduled recording to the ScheduledRecording service via the CreateRecordSchedule() action, the control point sets a number of properties and passes them to the ScheduledRecording service to express user-desired instructions to the scheduled recording. Then, as a response to the CreateRecordSchedule() action, the ScheduledRecording service creates a recordSchedule, assigns a unique ID to the recordSchedule and returns the recordSchedule with the complete set of initial property settings. The ScheduledRecording service MUST add OPTIONAL properties to the recordSchedule when a control point did not specify them. Additionally, the ScheduledRecording service MAY add some informative properties.

If a control point specifies unsupported or unknown properties as input to the CreateRecordSchedule() action, the ScheduledRecording service MUST gracefully ignore these. A control point can always parse the generated recordSchedule returned in the Result argument of the CreateRecordSchedule action to verify whether certain properties were rejected by the ScheduledRecording service. If unsupported values are set for supported properties, the ScheduledRecording service MUST return an error and the recordSchedule MUST NOT be created.

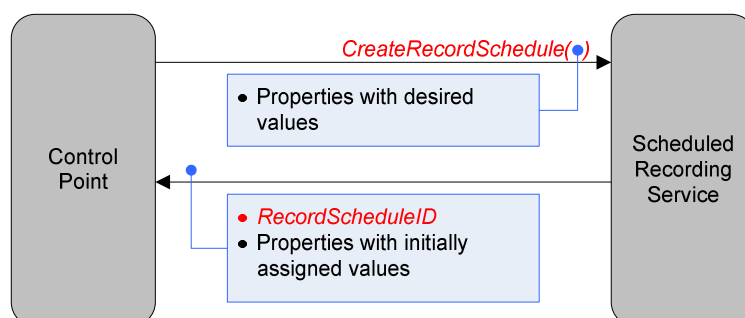


Figure 1: Creating a new recordSchedule

Some properties are defined as optional in the ScheduledRecording service. Therefore, a control point needs to determine which properties a ScheduledRecording service implementation actually supports. Since support levels and allowed values for properties can be different for *recordScheduleParts*, *recordSchedule* or *recordTask* usage, a pair of actions (*GetPropertyList()* and *GetAllowedValues()*) are provided to retrieve the relevant information. Figure 2 illustrates the concept.

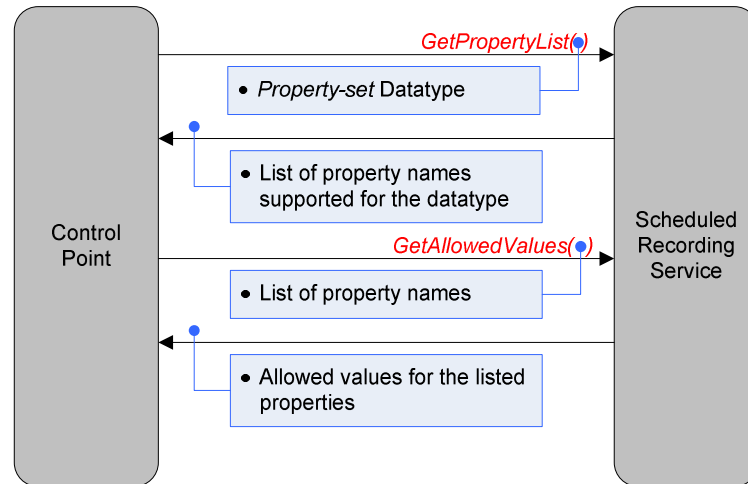


Figure 2: Capability check.

Figure 3 illustrates how *recordSchedule* instances can be browsed by the control point after they have been created, to retrieve the updated/current values of the properties.

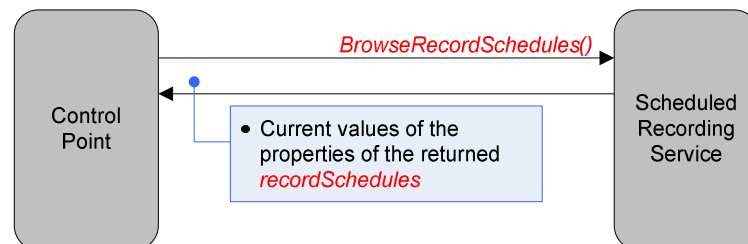


Figure 3: Browse *recordSchedule*.

Figure 4 illustrates how a control point can delete a *recordSchedule* from the ScheduledRecording service.

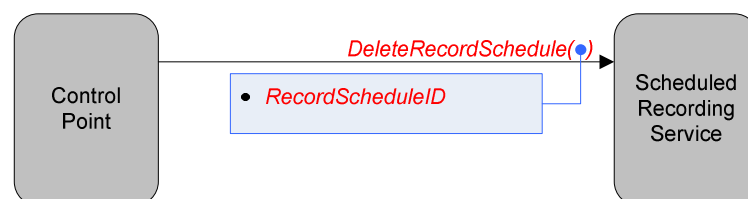


Figure 4: Delete a *recordSchedule*

2.3.2 *recordTask*

A *recordSchedule* will generate one *recordTask* for each recording operation that matches the criteria of the *recordSchedule*. A *recordTask* also has properties indicating its behavior. A *recordTask* is different from a *recordSchedule* in that it always represents a single recording operation whereas a *recordSchedule* may actually represent multiple recording operations. For example, a ScheduledRecording service that interprets a *recordSchedule* to lead to three different recording operations could generate three different *recordTask* instances over its lifetime. At a given time, a *recordSchedule* can have zero (no recording operations currently scheduled) or more *recordTask* instances associated with it. A ScheduledRecording service MUST report at least one *recordTask* when the underlying system is performing a recording operation for some *recordSchedule*.

When a *recordSchedule* is created, the ScheduledRecording service generates necessary *recordTask* instances associated with each scheduled recording occurrence. The ScheduledRecording service may also later add a new

recordTask whenever a new scheduled recording occurrence arrives. Similarly, a ScheduledRecording service may delete recordTask instances when they are no longer needed. This MAY happen in a device dependent manner. For example, some ScheduledRecording service implementations delete a recordTask when the recording is finished while other ScheduledRecording service implementations keep maintaining finished recordTask instances. A recordTask can only be created by the ScheduledRecording service as a result of a trigger from a recordSchedule. A control point can never create a recordTask directly. Both a recordTask and a recordSchedule MAY be deleted by the ScheduledRecording service or a control point.

The lifetime of a recordTask is determined in a vendor dependent way. Some implementations maintain a recordTask even after it finishes its recording while others may delete the recordTask once the recording finishes. However, in any implementation, when a recordSchedule is deleted, the ScheduledRecording service MUST delete all of its associated recordTask instances.

2.4 State Variables

Like the ContentDirectory service, the ScheduledRecording service is primarily action-based. The service state variables exist primarily to support argument passing within service actions. Information is not exposed directly through explicit state variables. Instead, a client retrieves ScheduledRecording service information via the return arguments of the actions defined below. The majority of state variables defined below exist simply to provide data type information for the arguments of the various actions of this service.

Reader Note: For a first-time reader, it may be more helpful to read the action definitions before reading the state variable definitions.

2.4.1 State Variable Overview

Table 2-2: State Variables

Variable Name	R/O ¹	Data Type	Allowed Value	Default Value	Eng. Units
<u>SortCapabilities</u>	<u>R</u>	<u>string</u>	CSV (<u>string</u>)		
<u>SortLevelCapability</u>	<u>R</u>	<u>ui4</u>			
<u>StateUpdateID</u>	<u>R</u>	<u>ui4</u>			
<u>LastChange</u>	<u>R</u>	<u>string</u>			
<u>A_ARG_TYPE_PropertyList</u>	<u>R</u>	<u>string</u>	CSV (<u>string</u>)		
<u>A_ARG_TYPE_DataTypeID</u>	<u>R</u>	<u>string</u>	See Table 2-3		
<u>A_ARG_TYPE_ObjectID</u>	<u>R</u>	<u>string</u>			
<u>A_ARG_TYPE_ObjectIDList</u>	<u>O</u> ²	<u>string</u>	CSV (<u>string</u>)		
<u>A_ARG_TYPE_PropertyInfo</u>	<u>R</u>	<u>string</u>			
<u>A_ARG_TYPE_Index</u>	<u>R</u>	<u>ui4</u>			
<u>A_ARG_TYPE_Count</u>	<u>R</u>	<u>ui4</u>			
<u>A_ARG_TYPE_SortCriteria</u>	<u>R</u>	<u>string</u>	CSV (<u>string</u>)		
<u>A_ARG_TYPE_RecordSchedule</u>	<u>R</u>	<u>string</u>			
<u>A_ARG_TYPE_RecordTask</u>	<u>R</u>	<u>string</u>			
<u>A_ARG_TYPE_RecordScheduleParts</u>	<u>R</u>	<u>string</u>			

¹ R = Required, O = Optional, X = Non-standard.

² CONDITIONALLY REQUIRED. This argument type variable is REQUIRED when the GetRecordScheduleConflicts() or GetRecordTaskConflicts() actions are implemented. See Sections 2.6.17.

“*GetRecordScheduleConflicts()*” and 2.6.18, “*GetRecordTaskConflicts()*” to determine when these actions MUST be implemented.

Table 2-3: allowedValueList for the DataTypeID argument

Value	R/O ¹
<u>“A_ARG_TYPE_RecordSchedule”</u>	<u>R</u>
<u>“A_ARG_TYPE_RecordTask”</u>	<u>R</u>
<u>“A_ARG_TYPE_RecordScheduleParts”</u>	<u>R</u>
<i>Vendor-defined</i>	<u>X</u>

¹ R = REQUIRED, O = OPTIONAL, X = Non-standard.

2.4.2 SortCapabilities

This state variable contains a CSV list of property names that the ScheduledRecording service can use to sort the information returned in the Result argument of various actions, such as BrowseRecordSchedules() and BrowseRecordTasks(). An empty string indicates that the device does not support any kind of sorting. A wildcard “srs:*” indicates that any supported property within the srs namespace can be used for sorting.

2.4.3 SortLevelCapability

This state variable contains an integer that indicates the maximum number of property names that can be specified in the SortCriteria argument at the same time.

2.4.4 StateUpdateID

This state variable is a ScheduledRecording service system-wide numeric value. Its initial value is 0.

- StateUpdateID MUST be incremented by 1 whenever any of the following occurs:
 - A recordSchedule or recordTask is created or deleted.
 - A recordSchedule or recordTask is modified, which means that one or more properties are added, deleted or had their value changed.
 - Any other change to the state of the ScheduledRecording service that could be observed by a control point. This includes any vendor- or other future-defined behavior.
- When the value of StateUpdateID is equal to the ui4 maximum value of 4294967295 ($2^{32}-1$), incrementing it causes it to roll over to the value 0.
- The increment and the operation that caused it must occur atomically relative to all information visible to any control point – including both action out arguments and evented variable values.

For example, consider the case where a control point invokes CreateRecordSchedule() to create a new recordSchedule that also immediately spawns exactly one recordTask. Assume that StateUpdateID is 10 when the control point invokes the action and that for a short time period around this invocation, no other activity occurs that affects the value of StateUpdateID. During this time period, exactly one of the following MUST be true as seen by all external observations (including the returned values from this CreateRecordSchedule() invocation):

- StateUpdateID is 10; and the new recordSchedule has not been created; and the new recordTask has not been created.
- StateUpdateID is 11; and the recordSchedule has been created; and the new recordTask has not been created; and the recordSchedule's value of currentRecordTaskCount is 0, indicating that no recordTask has been created.

- StateUpdateID is 12; and the recordSchedule has been created; and the new recordTask has been created; and the recordSchedule's value of currentRecordTaskCount is 1, indicating that the child recordTask has been created.

ScheduledRecording service implementations SHOULD maintain the same value for StateUpdateID through power cycles and any other disappearance/reappearance of the service on the network. Control points can use a change in the value of this variable to determine if there has been a change in the ScheduledRecording service.

The value of the StateUpdateID state variable, returned within events and returned as an output argument of certain actions should be monitored very closely by control points. Indeed, whenever an action returns with a StateUpdateID value in its UpdateID argument that is less than the StateUpdateID value received in the updateID attribute from the most recent LastChange event, the information returned by that action is potentially stale. A control point may want to refresh that information for instance by invoking the appropriate Browsexxx() or Getxxx() action. It is safe to use the information as long as the StateUpdateID value returned in the UpdateID argument of the action is greater than or equal to the StateUpdateID value received in the updateID attribute from the most recent LastChange event.

2.4.5 LastChange

Note: It is assumed that the default namespace for this sub-section (2.4.5, "LastChange") of the specification is srs-lc.

This state variable is used for eventing purposes to allow clients to receive meaningful event notifications whenever a recordSchedule or recordTask in the ScheduledRecording service changes. [SRS-EVENT-XSD] defines the schema for the *StateEvent XML Document* used in this state variable. The optional XML header `<?xml version="1.0" ?>` is allowed. One root element, `<StateEvent>` MAY have zero or more elements, each of which represent one update to a recordSchedule or recordTask instance. Six types of update elements are defined as shown in Table 2-4, "Allowed Elements in `<StateEvent>` Element". Future ScheduledRecording service specifications MAY add other types of update elements. A vendor MAY add vendor-defined elements. The ScheduledRecording:1 service does not define the value for these elements. Vendor-defined element names MUST follow the rules set forth in Section 1.5, "Vendor-defined Extensions". Note that future ScheduledRecording service specifications MAY define sub-elements for the elements. Also note that this state variable MUST be properly escaped as defined in [XML].

Table 2-4: Allowed Elements in `<StateEvent>` Element

Element Name	Description
RecordScheduleCreated	A new <u>recordSchedule</u> is created.
RecordScheduleModified	One or more properties of a <u>recordSchedule</u> are modified (added, deleted or values are changed).
RecordScheduleDeleted	A <u>recordSchedule</u> is deleted.
RecordTaskCreated	A new <u>recordTask</u> is created.
RecordTaskModified	One or more properties of a <u>recordTask</u> are modified (added, deleted or values are changed).
RecordTaskDeleted	A <u>recordTask</u> is deleted.
<i>Vendor-defined</i>	See Section 1.5, "Vendor-defined Extensions".

Each element MUST have one updateID attribute, which is set to the value of the StateUpdateID state variable at the time of the update and one objectID attribute, whose value is set to the value of the @id property of the updated recordSchedule or recordTask instance. Future ScheduledRecording service specifications MAY add other attributes to existing update elements. A vendor MAY add vendor-defined attributes for existing update elements.

Example (before XML escaping)

```
<?xml version="1.0" encoding="UTF-8"?>
<StateEvent
  xmlns="urn:schemas-upnp-org:av:srs-event"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    urn:schemas-upnp-org:av:srs-event
    http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd">
  <RecordScheduleCreated updateID="213" objectID="s001"/>
  <RecordTaskCreated updateID="214" objectID="s001-001"/>
  <RecordTaskModified updateID="215" objectID="s001-001"/>
</StateEvent>
```

The *LastChange* state variable is evented and moderated. When multiple updates occurred within a *LastChange* moderation period, the new *LastChange* state variable reports more than one update at the same time. A series of updates and the resulting eventing activity are illustrated in their temporal order in the following example.

Example

0: ScheduledRecording service activity = Power-on.

StateUpdateID = 0

LastChange (before XML escaping):

```
<?xml version="1.0" encoding="UTF-8"?>
<StateEvent
  xmlns="urn:schemas-upnp-org:av:srs-event"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    urn:schemas-upnp-org:av:srs-event
    http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd">
</StateEvent>
```

GENA behavior: None

1: ScheduledRecording service activity = a *recordSchedule* with *@id* = "s001" is created.

StateUpdateID = 1

LastChange (before XML escaping):

```
<?xml version="1.0" encoding="UTF-8"?>
<StateEvent
  xmlns="urn:schemas-upnp-org:av:srs-event"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    urn:schemas-upnp-org:av:srs-event
    http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd">
  <RecordScheduleCreated updateID="1" objectID="s001">
  </RecordScheduleCreated>
</StateEvent>
```

GENA behavior: Nothing is evented since there are no current subscribers.

2: ScheduledRecording service activity = new control point signs up for events.

StateUpdateID = 1

LastChange (before XML escaping):

```
<?xml version="1.0" encoding="UTF-8"?>
<StateEvent
  xmlns="urn:schemas-upnp-org:av:srs-event"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    urn:schemas-upnp-org:av:srs-event
    http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd">
  <RecordScheduleCreated updateID="1" objectID="s001">
```

```
</RecordScheduleCreated>
</StateEvent>
```

GENA behavior: Send initial Notify with the LastChange value above.

- 3: ScheduledRecording service activity** = a recordTask with @id = "t001-000" is created. Its associated recordSchedule with @id = "s001" is modified by the ScheduledRecording service at the same time because its currentReordTaskCount property is updated to reflect the existence of the new recordTask.

StateUpdateID = 3

LastChange (before XML escaping):

```
<?xml version="1.0" encoding="UTF-8"?>
<StateEvent
  xmlns="urn:schemas-upnp-org:av:srs-event"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    urn:schemas-upnp-org:av:srs-event
    http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd">
  <RecordTaskCreated updateID="2" objectID="t001-000">
  </RecordTaskCreated>
  <RecordScheduleModified updateID="3" objectID="s001">
  </RecordScheduleModified>
</StateEvent>
```

GENA behavior: Wait for the next moderation period to elapse and then send Notify with the LastChange value above.

- 4: ScheduledRecording service activity** = a recordTask with @id = "t001-001" is created. Its associated recordSchedule with @id = "s001" is modified by the ScheduledRecording service at the same time because its currentReordTaskCount property is updated to reflect the existence of the new recordTask. Within the same moderation period, a recordTask with @id = "t001-002" is also created. Its associated recordSchedule with @id = "s001" is modified by the ScheduledRecording service at the same time because its currentReordTaskCount property is updated to reflect the existence of the new recordTask.

StateUpdateID = 7

LastChange (before XML escaping):

```
<?xml version="1.0" encoding="UTF-8"?>
<StateEvent
  xmlns="urn:schemas-upnp-org:av:srs-event"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    urn:schemas-upnp-org:av:srs-event
    http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd">
  <RecordTaskCreated updateID="4" objectID="t001-001">
  </RecordTaskCreated>
  <RecordScheduleModified updateID="5" objectID="s001">
  </RecordScheduleModified>
  <RecordTaskCreated updateID="6" objectID="t001-002">
  </RecordTaskCreated>
  <RecordScheduleModified updateID="7" objectID="s001">
  </RecordScheduleModified>
</StateEvent>
```

GENA behavior: Wait for the next moderation period to elapse and then send Notify with the LastChange value above.

2.4.6 A ARG TYPE PropertyList

This state variable is introduced to provide type information for various action arguments that contain a CSV list of property names. Namespace prefixes MUST be included with all property names (see

Section 1.4, “Management of XML Namespaces”). The exact semantics of these property names depend on the associated action.

2.4.7 A ARG TYPE DataTypeID

This state variable is introduced to provide type information for various action arguments that are used to identify a specific *property-set* data type (see Section 2.2.2.13, “Property-set Data Types”). An argument of type A ARG TYPE DataTypeID can have the values listed in Table 2-3, “allowedValueList for the DataTypeID argument”.

2.4.8 A ARG TYPE ObjectID

This state variable is introduced to provide type information for various action arguments that uniquely identify an individual recordSchedule or a recordTask by their object ID.

2.4.9 A ARG TYPE ObjectIDList

This state variable is introduced to provide type information for various action arguments that contain a CSV list of object IDs (@id) used to identify a collection of either recordSchedule or recordTask instances (the list MUST be homogeneous).

2.4.10 A ARG TYPE PropertyInfo

This state variable is introduced to provide type information for various action arguments that contain detailed XML-based information on supported properties and their interdependencies for a particular ScheduledRecording service implementation. The format of these arguments is similar to the XML Service Description (SCPD), but instead of describing state variables and actions, they describe properties, their allowed values, and interdependencies.

Refer to [AVDT] for the definition of the AVDT Datastructure Template.

Note that since the format of these arguments is based on XML, it needs to be escaped (using the normal XML rules: [XML] Section 2.4 Character Data and Markup) before embedding in a SOAP response message.

2.4.11 A ARG TYPE Index

This state variable is introduced to provide type information for various action arguments that specify an offset into an arbitrary set of objects. A value of 0 represents the first object in the set.

2.4.12 A ARG TYPE Count

This state variable is introduced to provide type information for various action arguments that specify a number of arbitrary objects.

2.4.13 A ARG TYPE SortCriteria

This state variable is introduced to provide type information for various action arguments that contain a CSV list of property names prefixed by one or more sort modifiers. Namespace prefixes MUST be included with all property names that do not belong to the srs namespace. Namespace prefixes MAY be included with property names that belong to the srs namespace (see Section 1.4, “Management of XML Namespaces”). The “+” and “-” sort modifier prefixes indicate that the sort is in ascending or descending order, respectively, with regard to the value of the prefixed property name.

2.4.14 A ARG TYPE RecordSchedule

This state variable is introduced to provide type information for various action arguments that contain a list of zero or more recordSchedule objects. All instances of this data type MUST comply with the SRS schema. See Annex A for details.

The structure of an argument of data type [A_ARG_TYPE_RecordSchedule](#) is an *srs XML Document*:

- Optional XML declaration `<?xml version="1.0" ?>`
- `<srs>` is the root element.
- The `<srs>` element MUST have zero or more `<item>` elements, each representing a [recordSchedule](#) object.
- Each `<item>` element has a set of property values describing the [recordSchedule](#) object. Each property is expressed either as the content of an XML element or as the value of an XML attribute.
- See [SRS-XSD] for more details on the structure. The ScheduledRecording service-defined names for metadata are described in Annex B.

Note that since the SRS format of an argument of data type [A_ARG_TYPE_RecordSchedule](#) is XML, it needs to be escaped (using the normal XML rules: [XML] Section 2.4 Character Data and Markup) before embedding in a SOAP response message.

2.4.15 [A_ARG_TYPE_RecordTask](#)

This state variable is introduced to provide type information for various action arguments that contain a list of zero or more [recordTask](#) objects. All instances of this data type MUST comply with the SRS schema. See Annex A for details.

The structure of an argument of data type [A_ARG_TYPE_RecordTask](#) is an *srs XML Document*:

- Optional XML declaration `<?xml version="1.0" ?>`
- `<srs>` is the root element.
- The `<srs>` element MUST have zero or more `<item>` elements, each representing a [recordTask](#) object.
- Each `<item>` element has a set of property values describing the [recordTask](#) object. Each property is expressed either as the content of an XML element or as the value of an XML attribute.
- See [SRS-XSD] for more details on the structure. The ScheduledRecording service-defined names for metadata are described in Annex B.

Note that since the SRS format of an argument of data type [A_ARG_TYPE_RecordTask](#) is XML, it needs to be escaped (using the normal XML rules: [XML] Section 2.4 Character Data and Markup) before embedding in a SOAP response message.

2.4.16 [A_ARG_TYPE_RecordScheduleParts](#)

This state variable is introduced to provide type information for various action arguments that contain a single [recordScheduleParts](#) object. A [recordScheduleParts](#) object indicates the desired values for a subset of properties that provide a template for other [recordSchedule](#) objects. Typically, a [recordScheduleParts](#) is used to create new [recordSchedule](#) objects. All instances of this data type MUST comply with the SRS schema. See Annex A for details.

The structure of an argument of data type [A_ARG_TYPE_RecordScheduleParts](#) is an *srs XML Document*:

- Optional XML declaration `<?xml version="1.0" ?>`
- `<srs>` is the root element.
- The `<srs>` element MUST have a single `<item>` element, representing the [recordScheduleParts](#) object.
- The `<item>` element has a set of property values describing the [recordScheduleParts](#) object. Each property is expressed either as the content of an XML element or as the value of an XML attribute.

- See [SRS-XSD] for more details on the structure. The ScheduledRecording service-defined names for metadata are described in Annex B.

Note that since the SRS format of an argument of data type *A_ARG_TYPE_RecordScheduleParts* is XML, it needs to be escaped (using the normal XML rules: [XML] Section 2.4 Character Data and Markup) before embedding in a SOAP response message.

2.5 Eventing and Moderation

Table 2-5: Eventing and Moderation

Variable Name	Evented	Moderated Event	Max Event Rate ¹	Logical Combination	Min Delta per Event ²
<i>SortCapabilities</i>	<i>NO</i>	<i>NO</i>			
<i>SortLevelCapability</i>	<i>NO</i>	<i>NO</i>			
<i>StateUpdateID</i>	<i>NO</i>	<i>NO</i>			
<i>LastChange</i>	<i>YES</i>	<i>YES</i>	0.2 seconds		
<i>A_ARG_TYPE_PropertyList</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_DataTypeID</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_ObjectID</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_ObjectIDList</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_PropertyInfo</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_Index</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_Count</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_SortCriteria</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_RecordSchedule</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_RecordTask</i>	<i>NO</i>	<i>NO</i>			
<i>A_ARG_TYPE_RecordScheduleParts</i>	<i>NO</i>	<i>NO</i>			

¹ Determined by N, where Rate = (Event)/(N secs).

² (N) * (allowedValueRange Step).

2.6 Actions

Table 2-6: Actions

Name	R/O ¹
<i>GetSortCapabilities()</i>	<i>R</i>
<i>GetPropertyList()</i>	<i>R</i>
<i>GetAllowedValues()</i>	<i>R</i>
<i>GetStateUpdateID()</i>	<i>R</i>
<i>BrowseRecordSchedules()</i>	<i>R</i>
<i>BrowseRecordTasks()</i>	<i>R</i>
<i>CreateRecordSchedule()</i>	<i>R</i>

Name	R/O ¹
<u>DeleteRecordSchedule()</u>	<u>R</u>
<u>GetRecordSchedule()</u>	<u>R</u>
<u>EnableRecordSchedule()</u>	<u>O</u> ²
<u>DisableRecordSchedule()</u>	<u>O</u> ²
<u>DeleteRecordTask()</u>	<u>O</u>
<u>GetRecordTask()</u>	<u>R</u>
<u>EnableRecordTask()</u>	<u>O</u> ³
<u>DisableRecordTask()</u>	<u>O</u> ³
<u>ResetRecordTask()</u>	<u>O</u> ³
<u>GetRecordScheduleConflicts()</u>	<u>O</u> ⁴
<u>GetRecordTaskConflicts()</u>	<u>O</u> ⁵

¹ [R](#) = REQUIRED, [O](#) = OPTIONAL, [X](#) = Non-standard.

² CONDITIONALLY REQUIRED. The [EnableRecordSchedule\(\)](#) and [DisableRecordSchedule\(\)](#) actions MUST be implemented as a combination. If one action is implemented, then the other action MUST also be implemented.

³ CONDITIONALLY REQUIRED. The [EnableRecordTask\(\)](#), [DisableRecordTask\(\)](#), and [ResetRecordTask\(\)](#) actions MUST be implemented as a combination. If one action is implemented, then the other actions MUST also be implemented.

⁴ CONDITIONALLY REQUIRED. See Section 2.6.17, “[GetRecordScheduleConflicts\(\)](#)” to determine when this action MUST be implemented.

⁵ CONDITIONALLY REQUIRED. See Section 2.6.18, “[GetRecordTaskConflicts\(\)](#)” to determine when this action MUST be implemented.

2.6.1 [GetSortCapabilities\(\)](#)

This action returns a CSV list of property names that can be used in the [SortCriteria](#) argument of various actions.

2.6.1.1 Arguments

Table 2-7: Arguments for [GetSortCapabilities\(\)](#)

Argument	Direction	relatedStateVariable
<u>SortCaps</u>	<u>OUT</u>	<u>SortCapabilities</u>
<u>SortLevelCap</u>	<u>OUT</u>	<u>SortLevelCapability</u>

2.6.1.1.1 [SortCaps](#)

This argument contains a CSV list of property names that the ScheduledRecording service can use to sort the information returned in the [Result](#) argument of various actions, such as [BrowseRecordSchedules\(\)](#) and [BrowseRecordTasks\(\)](#). The appropriate namespace prefixes (either “srs:” or “<vendor-defined namespace prefix>:”) MUST be included with the returned property names (see Section 1.4, “Management of XML Namespaces”). An empty string indicates that the device does not support any kind of sorting. A wildcard “srs:*” indicates that any property within the srs namespace can be used for sorting. See also Section 2.4.2, “[SortCapabilities](#)”

2.6.1.1.2 SortLevelCap

This argument contains an integer that indicates the maximum number of property names that can be specified at the same time in the SortCriteria argument of various actions. See also Section 2.4.3, “SortLevelCapability.”

2.6.1.2 Dependency on State

None.

2.6.1.3 Effect on State

None.

2.6.1.4 Errors

Table 2-8: Error Codes for GetSortCapabilities()

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.6.2 GetPropertyList()

The GetPropertyList() action provides a means to retrieve from a particular ScheduledRecording service implementation which properties are actually supported for a specific *property-set* data type. The GetPropertyList() action returns a CSV list of property names that may appear in action arguments of the *property-set* data type, specified in the DataTypeID input argument. This CSV list MUST include property names of imported properties from other namespaces as well as any vendor-defined property names. For example, the ContentDirectory service imported properties (such as dc:title) that are included as part of the value of the cdsReference property, MUST be returned.

The appropriate namespace prefixes MUST be included with *all* property names (see Section 1.4, “Management of XML Namespaces”).

The set of allowed values for srs properties and vendor-defined properties (when used for the specified *property-set* data type) can be obtained via the GetAllowedValues() action. The set of allowed values for imported properties cannot be retrieved by the GetAllowedValues() action.

2.6.2.1 Arguments

Table 2-9: Arguments for GetPropertyList()

Argument	Direction	relatedStateVariable
<u>DataTypeID</u>	<u>IN</u>	<u>A_ARG_TYPE_DataTypeID</u>
<u>PropertyList</u>	<u>OUT</u>	<u>A_ARG_TYPE_PropertyList</u>

2.6.2.1.1 DataTypeID

The DataTypeID argument identifies the *property-set* data type for which the set of property names is to be returned. See Section 2.4.7, “A_ARG_TYPE_DataTypeID” for details regarding its format. The set of allowed values is listed in Table 2-3, “allowedValueList for the DataTypeID argument”.

2.6.2.1.2 **PropertyList**

The **PropertyList** argument contains the set of property names (including their namespace prefixes) that may appear in action arguments of the *property-set* data type, specified by the **DataTypeID** input argument.

2.6.2.2 **Dependency on State**

None.

2.6.2.3 **Effect on State**

None.

2.6.2.4 **Errors**

Table 2-10: Error Codes for **GetPropertyList()**

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
711	Invalid DataTypeID	An invalid value has been specified in the <u>DataTypeID</u> input argument.

2.6.3 **GetAllowedValues()**

This action is used to determine the allowed values and dependencies for srs properties that can appear within action arguments of the specified *property-set* data type. The set of allowed values that are returned is static and does not depend on the current state of the ScheduledRecording service. The property information is returned in an *AVDT XML Document* as defined in [AVDT]. The set of properties for which information is returned is determined by the intersection of the property names in the **Filter** argument and the names of the properties supported by the implementation for the specified *property-set* data type in the **DataTypeID** argument. All property names MUST belong either to the srs namespace or a vendor-defined namespace.

The set of allowed values for imported properties cannot be retrieved by the **GetAllowedValues()** action.

2.6.3.1 **Arguments**

Table 2-11: Arguments for **GetAllowedValues()**

Argument	Direction	relatedStateVariable
<u>DataTypeID</u>	<u>IN</u>	<u>A_ARG_TYPE_DataTypeID</u>
<u>Filter</u>	<u>IN</u>	<u>A_ARG_TYPE_PropertyList</u>
<u>PropertyInfo</u>	<u>OUT</u>	<u>A_ARG_TYPE_PropertyInfo</u>

2.6.3.1.1 **DataTypeID**

See Section 2.6.2.1.1, “**DataTypeID**”.

2.6.3.1.2 **Filter**

The **Filter** argument contains a CSV list of property names that indicates for which properties allowed value information is to be returned in the *AVDT XML Document*, contained in the **PropertyInfo** output argument. The **Filter** argument SHOULD only include property names that are returned in the **PropertyList** argument of the **GetPropertyList()** action when specifying the same value in the **DataTypeID** argument. ScheduledRecording service implementations MUST gracefully ignore other property names. The “srs:” namespace prefix MUST be included with srs property names in the **Filter** argument. Likewise, a namespace prefix MUST be included with all vendor-defined property names in the **Filter** argument (see Section 1.4, “Management of XML Namespaces”).

If the **Filter** argument is set to “*:*”, then allowed values for all supported properties (including srs properties and vendor-defined properties, but excluding imported properties) for the specified *property-set* data type MUST be returned. If the **Filter** argument is set to “srs:*”, then allowed values for all supported properties in the srs namespace MUST be returned. If the **Filter** argument is set to “<vendor-defined namespace prefix>:*”, then allowed values for all vendor-defined properties in that namespace MUST be returned. If the **Filter** argument is set to the empty string, no information is provided (an *AVDT XML Document* with an empty root element is returned).

Examples of valid **Filter** argument values include:

- “srs:@id,srs:priority@orderedValue”
- “srs:title,srs:class”
- “*:*”
- “srs:*”

2.6.3.1.3 **PropertyInfo**

The **PropertyInfo** argument MUST only include allowed value and dependency information on properties that are specified in the **Filter** argument. The **PropertyInfo** argument MUST be properly escaped as defined in [XML]. The particular *AVDT XML Document* that is returned in the **PropertyInfo** argument depends on the *property-set* data type, specified in the **DataTypeID** input argument. See Annex A for further details.

2.6.3.2 **Dependency on State**

None.

2.6.3.3 **Effect on State**

None.

2.6.3.4 **Errors**

Table 2-12: Error Codes for **GetAllowedValues()**

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
711	Invalid DataTypeID	An invalid value has been specified in the <u>DataTypeID</u> input argument.

2.6.4 **GetStateUpdateID()**

This action returns the current value of the **StateUpdateID** state variable in the **Id** output argument. This action can be used to poll the ScheduledRecording service for any change in the service that might have

occurred since the last time this action was invoked. If the returned *Id* value is different from the value that was returned the last time this action was invoked, then there has been a change in one or more *recordSchedule* or *recordTask* objects in the ScheduledRecording service. See Section 2.4.4, “*StateUpdateID*” for more information.

2.6.4.1 Arguments

Table 2-13: Arguments for *GetStateUpdateID()*

Argument	Direction	Related State Variable
<u><i>Id</i></u>	<u><i>OUT</i></u>	<u><i>StateUpdateID</i></u>

2.6.4.1.1 *Id*

The *Id* argument contains the current value of the *StateUpdateID* state variable.

2.6.4.2 Dependency on State

None.

2.6.4.3 Effect on State

None.

2.6.4.4 Errors

Table 2-14: Error Codes for *GetStateUpdateID()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.6.5 *BrowseRecordSchedules()*

This action is used to browse the set of *recordSchedule* objects in the ScheduledRecording service.

2.6.5.1 Arguments

Table 2-15: Arguments for *BrowseRecordSchedules()*

Argument	Direction	relatedStateVariable
<u><i>Filter</i></u>	<u><i>IN</i></u>	<u><i>A_ARG_TYPE_PropertyList</i></u>
<u><i>StartingIndex</i></u>	<u><i>IN</i></u>	<u><i>A_ARG_TYPE_Index</i></u>
<u><i>RequestedCount</i></u>	<u><i>IN</i></u>	<u><i>A_ARG_TYPE_Count</i></u>
<u><i>SortCriteria</i></u>	<u><i>IN</i></u>	<u><i>A_ARG_TYPE_SortCriteria</i></u>
<u><i>Result</i></u>	<u><i>OUT</i></u>	<u><i>A_ARG_TYPE_RecordSchedule</i></u>
<u><i>NumberReturned</i></u>	<u><i>OUT</i></u>	<u><i>A_ARG_TYPE_Count</i></u>
<u><i>TotalMatches</i></u>	<u><i>OUT</i></u>	<u><i>A_ARG_TYPE_Count</i></u>
<u><i>UpdateID</i></u>	<u><i>OUT</i></u>	<u><i>StateUpdateID</i></u>

2.6.5.1.1 **Filter**

The **Filter** argument contains a CSV list of property names that indicates which properties are to be returned in the *srs XML Document*, contained in the **Result** output argument. Namespace prefixes MUST be included with all property names, specified in the **Filter** argument (see Section 1.4, “Management of XML Namespaces”).

The **Filter** argument has no impact on the *number* of objects returned in the **Result** argument. Instead, the **Filter** argument allows control points to control the *complexity* of the object metadata that is returned in the *srs XML Document* for each object. It allows a control point to specify a subset of the supported properties for inclusion in the *srs XML Document*. Properties that are REQUIRED by the SRS Schema MUST always be returned. Compliant ScheduledRecording service implementations MUST NOT return optional properties unless they are explicitly requested in the **Filter** input argument or are needed to create a valid XML document. For example, specifying a dependent property in the **Filter** argument, such as **priority@orderedValue**, will cause its associated independent property, **priority**, to be included in the *srs XML Document*.

In all cases, a compliant ScheduledRecording service implementation MUST always respond to query requests with the smallest, valid *srs XML Document* in the **Result** argument that satisfies the **Filter** input argument. If the **Filter** argument is set to the empty string (“”), then only the REQUIRED properties are returned.

If the **Filter** argument is equal to “* : *”, then all supported properties for all supported namespaces MUST be returned. If the **Filter** argument is equal to “<namespace prefix> : *”, then all of the REQUIRED srs properties and all of the supported properties within that single namespace MUST be returned. For example, “srs : *” is equivalent to listing all srs namespace properties supported by the device.

Properties defined in the ContentDirectory service MUST only be imported through the multi-valued **cdsReference** property. Therefore, if the **Filter** argument contains property names from namespaces defined in the ContentDirectory service specification, the appropriate **cdsReference** property values MUST be included in the **Result** output argument and those values MUST be filtered, according to what is specified in the **Filter** argument but also preserving the validity of the *DIDL-Lite XML Document*, returned in the **cdsReference** property.

Examples of valid **Filter** argument values include:

- “srs:@id,srs:priority@orderedValue”
- “srs:title,dc:title”
- “* : *”
- “upnp:* ,dc:* ,didl_lite:*”

A compliant ScheduledRecording service implementation MUST also ignore optional properties requested in the **Filter** input argument which are not actually present in the matching objects. For example, a **BrowseRecordSchedules()** **Filter** input argument of the form “srs:activePeriod” is successful and returns a **Result** value that complies with the other **BrowseRecordSchedules()** input arguments, even in the case where the objects represented in the **Result** argument do not have an **activePeriod** property defined.

2.6.5.1.2 **StartingIndex** and **RequestedCount**

This action returns a specified number of **recordSchedule** objects from the list as indicated by the **RequestedCount** argument and starting from a specified index in the list, as indicated by the **StartingIndex** argument. The first **recordSchedule** in the list MUST be indexed by an index value of 0. Specifying 0 in the **RequestedCount** argument is PROHIBITED. If the range indicated by the **StartingIndex** and **RequestedCount** arguments reaches beyond the end of the list, then the ScheduledRecording service MUST return all **recordSchedule** objects up to the end of the list and starting from the specified **StartingIndex**.

2.6.5.1.3 SortCriteria

The order of the recordSchedule objects in the Result argument is determined by the SortCriteria argument. When an empty string is specified in the SortCriteria argument, then the order is device dependent. Additionally, this device dependent ordering MUST remain constant unless the UpdateID argument value has changed since the last BrowseRecordSchedules() action. In other words, any two objects that appear in a Result argument MUST always appear in the same relative order as long as the UpdateID argument value (and therefore the StateUpdateID state variable) did not change.

The SortCriteria argument contains a CSV list of property names (namespace prefixes MUST always be included). Each property name MUST be prefixed by either a “+” or a “-” sort modifier. The “+” and “-” modifiers indicate that the sort is in ascending or descending order, respectively, with regard to the value of its associated property.

The ScheduledRecording service MUST NOT accept any property name in the SortCriteria argument that is not included in the SortCapabilities state variable.

The objects are first sorted on the value of the first property in the SortCriteria argument. If all values differ in the first property, the sort is finished. If any values of the first property are equal, each subset of equal values is then sorted based on the next property in the SortCriteria argument. This process repeats iteratively until there are no more subsets of equal values or the SortCriteria argument list is exhausted.

For example, a value for the SortCriteria argument of the BrowseRecordSchedules() action of:

```
“+srs:scheduledStartTime, -
srs:scheduledChannelID, +srs:matchingName”
```

would sort the returned recordSchedule instances first by start date&time in ascending order, then for each date&time, the instances would be sorted by descending channel ID and finally, for each channel ID, the instances would be sorted by ascending program name.

Sorting rules for each property depend on that property’s semantics. Sorts for individual properties can be any of: numeric sort, lexical sort, lexical numeric sort, Boolean sort, sequenced sort, type relationship sort, or property specific, according to an explicit ordering of values defined individually for that property. The definition of each kind of sort may be found in Section 2.2.2.26, “Lexical Sort Order”. The specific sort order rules that MUST be used for each property are given in Annex B.

When a SortCriteria argument contains property names of optional and/or multi-valued properties, the following rules apply:

If the property is prefixed by “+” then:

- Objects that do not have a value for the property are returned first in their group.
- Objects that have at least one value for the property are returned next in their group. Objects that have multiple values for the property (either multi-valued or CSV list) are sorted based on the property value that would cause the object to appear earliest in the list.

If the property is prefixed by “-” then:

- Objects that have at least one value for the property are returned first in their group. Objects that have multiple values (either multi-valued or CSV list) for the property are sorted based on the property value that would cause the object to appear earliest in the list.
- Objects that do not have a value for the property are returned last in their group.

Example:

Assume a ScheduledRecording service contains the following items and the current date is Tuesday, June 21, 2005:

```
<item id="1">
...
<scheduledStartTime>2006-02-
07T15:30:00</ScheduledStartTime>
...
</item>
```