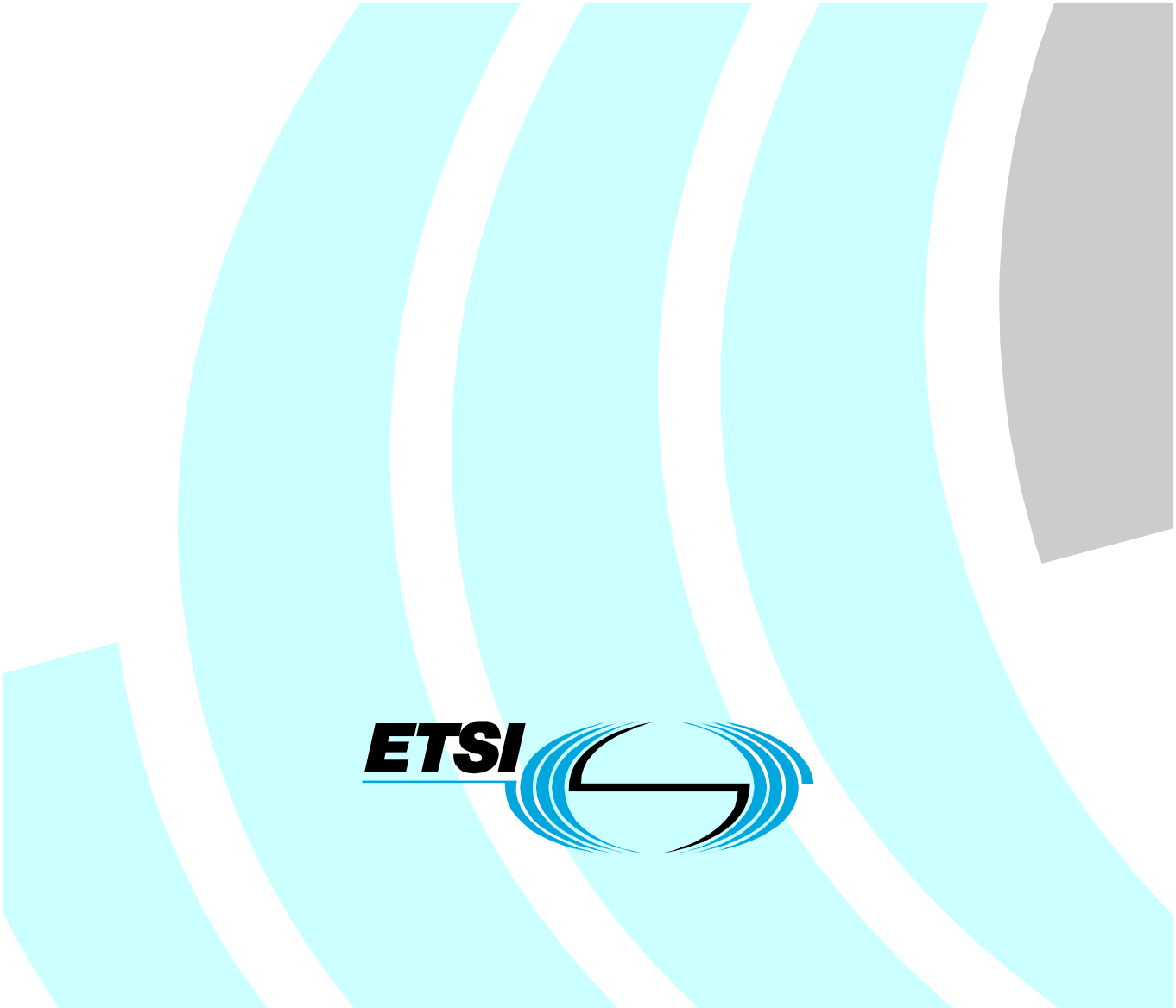
ETSI TS 102 822-6-1 V1.1.1 (2003-10)

Technical Specification

Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1");
Part 6: Delivery of metadata over a bi-directional network;
Sub-part 1: Service and transport



Reference
DTS/JTC-TVA-PH1-06-1

Keywords
broadcasting, content, service, transport, TV, video

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights 5
Foreword 5
Introduction 5
1 Scope 6
2 References 7
3 Definitions, abbreviations and conformance 8
3.1 Definitions 8
3.2 Abbreviations 9
3.3 Conformance 10
4 Introduction 10
4.1 Types and Functionalities of Metadata Services 11
4.1.1 Metadata Retrieval 11
4.1.2 Submission of User-centric Metadata 13
4.2 Metadata Service Capability Descriptions 13
4.3 Metadata Service Discovery 13
4.3.1 Non-standardized Discovery 14
4.3.2 Unidirectional Delivery of Discovery Information 14
4.3.3 Client-Initiated Discovery Using the Bi-directional Network 14
5 Metadata Service Types 14
5.1 get_Data Operation 14
5.1.1 Request Format 15
5.1.1.1 Query constraint parameters 15
5.1.1.1.1 Identifying contextNodes 17
5.1.1.1.2 Identifying fields 19
5.1.1.1.3 Primary index CRID fields 22
5.1.1.1.4 Restrictions on the use of QueryConstraints 23
5.1.1.1.5 Evaluating a predicate 24
5.1.1.2 View on returned data 25
5.1.1.2.1 Sort criteria 26
5.1.1.3 Size limit parameter 27
5.1.1.4 Interpretation of Query Predicates 27
5.1.1.5 Definition of server behaviour 28
5.1.2 Response Format 28
5.1.2.1 Indicating the sorting of the response 29
5.1.2.2 Indicating the service version 29
5.1.2.3 Truncating the result set 30
5.1.2.4 Updating the result set 30
5.2 submit_Data Operation 31
5.2.1 Usage and user preference data submission policy (informative) 31
5.2.2 Request Format 31
5.2.3 Response Format 32
6 Transport Protocol 32
6.1 SOAP 33
6.2 Error Codes 34
6.2.1 General error conditions 35
6.2.2 get_Data operation error conditions 35
6.3 HTTP 36
6.4 Encapsulation of Metadata 36
6.4.1 Encapsulation of get_Data response 36
6.5 Encoding of Metadata 37
6.6 Metadata Service Security 37

7 Metadata Service Capability Descriptions 37
7.1 describe_get_Data 37
7.1.1 Use of the AuthorityList element 39
7.1.2 AvailableTables information 39
7.1.2.1 Operations that can deliver content referencing information 41
7.1.2.2 Operations that can deliver programme metadata 42
7.1.3 Extended Field List 42
7.1.4 Description of update capabilities 42
7.2 describe_submit_Data 42

Annex A (normative): Formal Definition of Metadata Services 44

Annex B (normative): TV Anytime defined field and contextNode identifiers 47

Annex C (informative): Examples of get_Data Requests 48

C.1 Requesting data on specific CRIDs 48
C.2 Requesting specific fragments 48
C.3 Searching for the film "Titanic" 48
C.4 Searching for a comedy drama that does not star Jim Carrey 49
C.5 Searching for a programme with a rating of more than 8 49
C.6 Searching for a ClassificationScheme Table 49
C.7 Creating an EPG 50
C.8 Searching for programmes with a review 50
C.9 Updating a fragment 51
C.10 Requesting update fragments 51

Annex D (informative): Examples of get_Data Operation's Capability Description 52

D.1 Pure location resolution service 52
D.2 Pure metadata retrieval service for broadcast enhancement 52
D.3 A rich metadata service that allows users to search for movies 53
D.4 Broadcaster provided metadata service used for constructing traditional EPGs 54
D.5 Pure metadata retrieval service for bi-directional channel 54

Annex E (informative): Use of BiM Encoded Metadata in Bi-directional Transport 56

E.1 Applying BiM encoding 56
E.2 Negotiation of BiM Encoding 56

Annex F (informative): Bibliography 57

List of figures 58
List of tables 58
History 59

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

The present document is part 6 sub-part 1, of a multi-part deliverable covering Broadcast and On-line Services: Search, select and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1"), as identified below:

Part 1: "Phase 1 Benchmark Features";

Part 2: "System description";

Part 3: "Metadata";

Part 4: "Content referencing";

Part 5: Not currently applicable in *TV-Anytime* Phase 1;

Part 6: "Delivery of metadata over a bi-directional network";

Sub-part 1: "Service and transport";

Sub-part 2: "Service discovery";

Part 7: "Bi-directional metadata delivery protection".

Introduction

The present document is based on a submission by the *TV-Anytime* forum (http://www.*TV-Anytime*.org).

'*TV-Anytime* Phase 1' (TVA-1) is the first full and synchronized set of specifications established by the *TV-Anytime* Forum. TVA-1 features enable the search, selection, acquisition and rightful use of content on local and/or remote personal storage systems from both broadcast and online services.

The features are supported and enabled by the specifications for Metadata, Content Referencing, and Bi-directional Metadata Delivery Protection, TS 102 822-3 sub-parts 1 [12] and 2 [13], TS 102 822-4 [14], TS 102 822-6-2 [15] and TS 102 822-7 [16]] respectively. All Phase 1 Features listed in TV035r6 are enabled by the normative *TV-Anytime* tools specifications. This list of Phase 1 Features is to be used as guidance to manufacturers, service providers and content providers regarding the implementation of the Phase 1 *TV-Anytime* specifications.

1 Scope

The present document is the sixth in a series of "S-series" specification documents produced by the *TV-Anytime* Forum. These documents establish the fundamental specifications for the services, systems and devices that will conform to the *TV-Anytime* standard, to a level of detail that is implementable for compliant products and services.

As is common practice in such standardization efforts, these specification documents were preceded by requirements documents, which define the requirements for the *TV-Anytime* services, systems, and devices.

Congruent with the structure defined in the initial *TV-Anytime* Call for Contributions (TV014r3), these specifications are parsed into three major areas: Metadata, Content Referencing, and Rights Management and Protection. Within these general areas, four specifications have been developed to date: TS 102 822-3-1 [12], TS 102 822-3-2 [13], TS 102 822-4 [14], TS 102 822-6-1 (the present document), TS 102 822-6-2 [15] and TS 102 822-7 [16]. A specification for TS 102 822-5 is still under development. See the several *TV-Anytime* Calls for Contributions for more detail on the derivation and background of these categories and their respective roles in the *TV-Anytime* standardization process.

The first two documents in the *TV-Anytime* S-series are intended to define the context and system architecture in which the standards in TS 102 822-3-1 [12], TS 102 822-3-2 [13], TS 102 822-4 [14], TS 102 822-6-1 (the present document), TS 102 822-6-2 [15] and TS 102 822-7 [16] are to be implemented in "Phase 1" of the *TV-Anytime* environment. The first document in the series (TS 102 822-1 [25]) provides benchmark business models against which the *TV-Anytime* system architecture is evaluated to ensure that the specification enable key business applications. The next document in the series (TS 102 822-2 [11]) presents the *TV-Anytime* System Architecture. These two documents are placed ahead of the other three for their obvious introductory value. (Note that TS 102 822-1 [25] and TS 102 822-2 [11] are largely informative documents, while the remainder of the S-series is normative. Also note that a "Phase 2" of the *TV-Anytime* process is currently underway, in which additional requirements and specifications that will build on Phase 1 are being developed. Readers are encouraged to check the *TV-Anytime* Forum's website at www.*TV-Anytime*.org for the most recent status of its specifications.)

Although each of the S-series documents is intended to stand alone, a complete and coherent sense of the *TV-Anytime* system standard can be gathered by reading all of the Phase 1 specification documents in numerical order.

The scope of the present document, comprises the delivery of *TV-Anytime* metadata and content referencing information via a bi-directional network using a PDR's return path.

The requirements for this technology are outlined in the *TV-Anytime* Forum's Requirement Series R-1 document [10]. The following paragraphs from those requirements give an overview of the return path's use:

- "The consumer can get more information about the programme from either the content provider or from a programme information service offered by a third party. This could include programme specifications (such as source, duration, format, storage location, etc.), programme schedules, commentary, critiques, liner notes from the provider or third parties, etc."

- "A Return Path is a data connection from a consumer's home digital storage system (e.g. PDR) to one or more service providers. The return path gives the consumer access to interactive content, such as the Internet and interactive television. It also allows service providers to access consumer profile/preference information in order to make business decisions regarding content that is provided to the consumer."

The present document describes a client-initiated means for requesting metadata from, and submitting user-centric data to, IP based web services. In the present document, these web services are termed "metadata services". The specification also defines a means for describing and discovering such metadata services, but does not address the unidirectional delivery of metadata over IP networks, or the delivery of content over IP networks. A more complete definition of the scope of this work, along with the system requirements, may be found in document TV150, "Requirements and Scenarios for the Bi-directional Transport of Metadata" [3].

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

[1] XML, Extensible Markup Language (XML) 1.0, October 2000.

NOTE: Available at: http://www.w3.org/TR/2000/REC-xml-20001006.

[2] Namespaces in XML, W3C Recommendation, 14 January 1999.

NOTE: Available at: http://www.w3.org/TR/REC-xml-names/.

[3] Requirements and Scenarios for the Bi-directional Transport of Metadata, TV150r1. The *TV-Anytime* Forum.

NOTE: Available at: http://www.*TV-Anytime*.org.

[4] IETF RFC 1591: "Domain Name System Structure and Delegation", J. Postel.

[5] IETF RFC 1945: "Hypertext Transfer Protocol, HTTP/1.0", T. Berners-Lee, R. Fielding, H. Frystyk.

[6] IETF RFC 2119: "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner.

[7] IETF RFC 2396: "Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter.

[8] IETF RFC 2616: "Hypertext Transfer Protocol, HTTP/1.1", R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee.

[9] Simple Object Access Protocol (SOAP) 1.1, W3C Note, 8 May 2002. D. Box, et. al.

NOTE: Available at: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[10] *TV-Anytime* Requirements Series: R-1, TV035r6. The *TV-Anytime* Forum.

NOTE: Available at: http://www.*TV-Anytime*.org.

[11] ETSI TS 102 822-2: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1"); Part 2: System description".

[12] ETSI TS 102 822-3-1: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1"); Part 3: Metadata; Sub-part 1: Metadata schemas".

[13] ETSI TS 102 822-3-2: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1"); Part 3: Metadata; Sub-part 2: System aspects in a uni-directional environment".

[14] ETSI TS 102 822-4: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1"); Part 4: Content Referencing".

[15] ETSI TS 102 822-6-2: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1"); Part 6: Delivery of metadata over a bi-directional network; Sub-part 2: Service discovery".

[16] ETSI TS 102 822-7: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1"); Part 7: Bi-directional metadata delivery protection".

[17] Unicode Collation Algorithm, Unicode Technical Report #10. M. Davis, K. Whistler.

NOTE: Available at: http://www.unicode.org/unicode/reports/tr10.

[18] Unicode Normalization Forms, Unicode Standard Annex #15. M. Davis, M. Dürst.

NOTE: Available at: http://www.unicode.org/unicode/reports/tr15.

[19] Universal Description Discovery & Integration, Version 3.0, T. Bellwood, et. al.

NOTE: Available at: http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

[20] Web Services Description Language, Version 1.1, W3C Note 15 March 2001, E. Christensen, F. Curbera, G. Meredith, S. Weerawarana.

NOTE: Available at: http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[21] Web Services Inspection Language, Version 1.0, K. Ballinger, P. Brittenham, A. Malhotra, W. A. Nagy, S. Pharies.

NOTE: Available at: http://www.ibm.com/developerworks/webservices/library/ws-wsilspec.html.

[22] XML Schema, W3C Recommendations (version 20010502).

NOTE: Available at: http://www.w3.org/TR/2001/REC-xmlschema-0-20010502, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502

[23] The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, M. Marchiori et. al.

NOTE: Available at: http://www.w3.org/TR/P3P/.

[24] The WS-Inspection and UDDI Relationship, W. A. Nagy, K. Ballinger.

NOTE: Available at: http://www-106.ibm.com/developerworks/webservices/library/ws-wsiluddi.html.

[25] ETSI TS 102 822-1: "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("*TV-Anytime* Phase 1"); Part 1: Phase 1 Benchmark Features".

[26] ISO/IEC 15938-1: "Information technology - Multimedia content description interface - Part 1: Systems".

3 Definitions, abbreviations and conformance

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

acquisition: retrieval of content

application: a specific set of functions running on the PDR. Some applications use metadata, either automatically or under consumer control

authority: organization that creates CRIDs

bi-directional network: a network that supports two way, point-to-point, one-to-many, and many-to-many data delivery

NOTE: The Internet is an example of such a network. A PDR may access a bi-directional network using its return path.

capture: storing the acquired content (e.g. to local storage)

content: anything the viewer would like to access (movies, games, TV programmes, radio programmes, etc.)

content creator: producers of the content

content provider: entity that acts as the agent for and is the prime exploiter of the content

content reference: pointer to a specific content item

location resolution: process of establishing the address (location and time) of a specific content instance from its CRID

locator: time and place where a content item can be acquired

metadata: generally, data about content, such as the title, genre, and summary of a television programme

NOTE: In the context of *TV-Anytime*, metadata also includes consumer profile and history data.

metadata service: service that provides *TV-Anytime* data using a server on a bi-directional network

NOTE: The formats of the data and the protocols used to deliver that data are defined by the present document.

programme: editorially coherent piece of content

NOTE: Typically, a programme is acquired by the PDR as a whole.

resolving authority: body which provides location resolution

Resolving Authority Record (RAR): information needed for retrieving the location resolution data for the given authority

return path: part of the bi-directional distribution system from the consumer to service provider

segment: continuous portion of a piece of content, for example a single news topic in a news programme

service provider: aggregator and supplier of content which may include gateway and management roles

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ARIB Association of Radio Industries and Businesses

NOTE: A Japanese standards organization.

ATSC Advanced Television Systems Committee

NOTE: American based standards organization for establishing technical standards for advanced television systems, including digital high definition television.

BiM Binary format for multimedia description streams

NOTE: Defined in ISO/IEC 15938-1 [26] (MPEG-7 Systems part).

CE Consumer Electronics
CRID Content Reference Identifier: identifier for content that is independent of its location
DNS Domain Naming System: system used on the Internet to register names that can then be mapped into IP addresses using a DNS server
DVB Digital Video Broadcasting

NOTE: Set of standards used for European digital TV broadcasting.

EPG Electronic Programme Guide

NOTE: Means of presenting content to the consumer, allowing selection of desired content.

HTTP HyperText Transfer Protocol
IP Internet Protocol

NOTE: Generic name for the network protocols used on the Internet.

IPR	Intellectual Property Rights
PDR	Personal Digital Recorder
RAR	Resolving Authority Record
SI	System Information

NOTE: Collection of information tables used in DVB.

SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDDI	Universal Description Discovery & Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VOD	Video On Demand
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WS-Inspection	Web Services Inspection Language
XML	Extensible Markup Language

3.3 Conformance

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [6].

It is important to note that OPTIONAL and RECOMMENDED elements of the specification, if they are implemented, MUST be implemented in the manner documented in the present document.

4 Introduction

The *TV-Anytime* Forum has defined a number of data types that can be exchanged between *TV-Anytime* devices. These include programme metadata, content referencing information, and user-centric metadata. The present document is concerned with data exchange between *TV-Anytime* clients and metadata servers over a bi-directional network using the return path. A *TV-Anytime* client is typically a PDR, although in this document the client can be any Internet connected device. These devices do not necessarily need to have the ability to display or store content, since many types of devices can exploit *TV-Anytime* metadata services (e.g. a mobile phone displaying an EPG).

Programme metadata and content referencing information can be delivered unidirectionally (e.g. via traditional broadcast or IP multicast) or via a bi-directional network. The reasons a *TV-Anytime* provider might choose to deliver data via a bi-directional network are as follows:

- It allows a richer set of metadata to be delivered, since there are much lower bandwidth constraints.
- It allows *TV-Anytime* data providers without access to a broadcast system to deliver metadata to clients.
- It allows *TV-Anytime* data providers to personalize the metadata they offer according to the source of the request.
- It allows a range of client devices, which are not necessarily able to receive broadcast data, to access and exploit *TV-Anytime* data. For example, a mobile phone or personal organizer could use the metadata service to show the user an EPG.

User-centric metadata can only be delivered from a PDR to a *TV-Anytime* metadata service when a return path is available and the user authorizes it. The submission of such user data allows the metadata service to provide a variety of value adding services, which are more completely described in clause 6.5 of the TS 102 822-3-1 [12].

The present document defines the protocols that allow these transactions to take place in an interoperable fashion. Note that, due to the widespread nature and mass penetration of the Internet, the *TV-Anytime* Forum has completely specified the transport and network layer protocols (TCP/IP) necessary for end-to-end interoperability. This is in contrast to unidirectional transport mechanisms, which are not completely specified by the *TV-Anytime* Forum, but instead defined individually by the bodies responsible for the various broadcast standards used around the world (e.g. ARIB, ATSC, DVB, etc.).

To use a *TV-Anytime* metadata service a client takes the steps illustrated in figure 1.

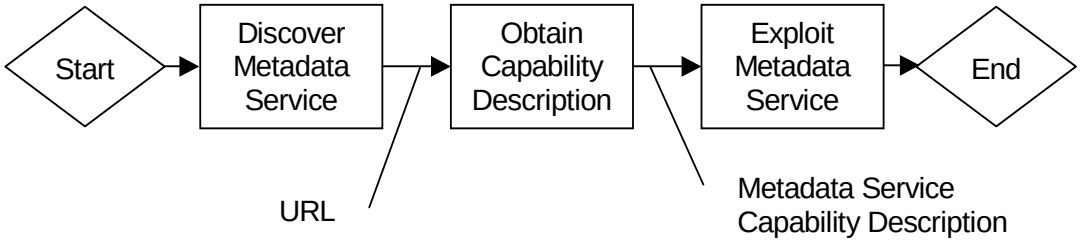


Figure 1: The steps in using a *TV-Anytime* metadata service

These steps are described in more detail in the following three clauses (in reverse order). Note that a metadata service MUST provide a description capability (see clause 7), but the support for metadata service discovery (see TS 102 822-6-2 [15]) is OPTIONAL. The middle step typically will only occur when a metadata service is first discovered or updated, and not each time the metadata service is used.

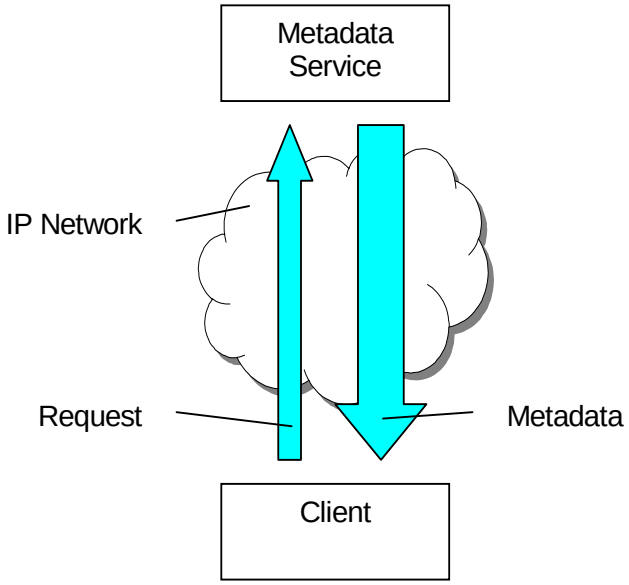
4.1 Types and Functionalities of Metadata Services

TV-Anytime metadata services can be broken into two basic types, which are shown in figures 2 and 3. The metadata services specified are all request-response based. This can be seen in the two figures - the network transaction is always point-to-point (client to server), and the transaction is always initiated by the client.

4.1.1 Metadata Retrieval

Metadata retrieval occurs when a client wishes to obtain certain metadata from a metadata service that it has previously discovered and obtained a capability description for. The following list gives some examples of metadata retrieval.

- A client wishes to obtain programme reviews for a CRID. The client sends a request specifying the CRID and type of metadata required, and the metadata service responds with the appropriate `ProgramReviewTable`.
- A client wishes to obtain the schedule information for a particular channel over the next week. The client sends a request specifying the channel, date range, and type of metadata required. The metadata service returns a `ProgramLocationTable` and `ProgramInformationTable` corresponding to the programmes on that channel.
- A client wishes to search a metadata service that specializes in movie information. The client sends a request specifying the type of movie (e.g. the genre is "Western", and the star is "John Wayne"), and the type of metadata required. The metadata service returns a number of matching movies, using a `ProgramInformationTable` and `ProgramReviewTable`.

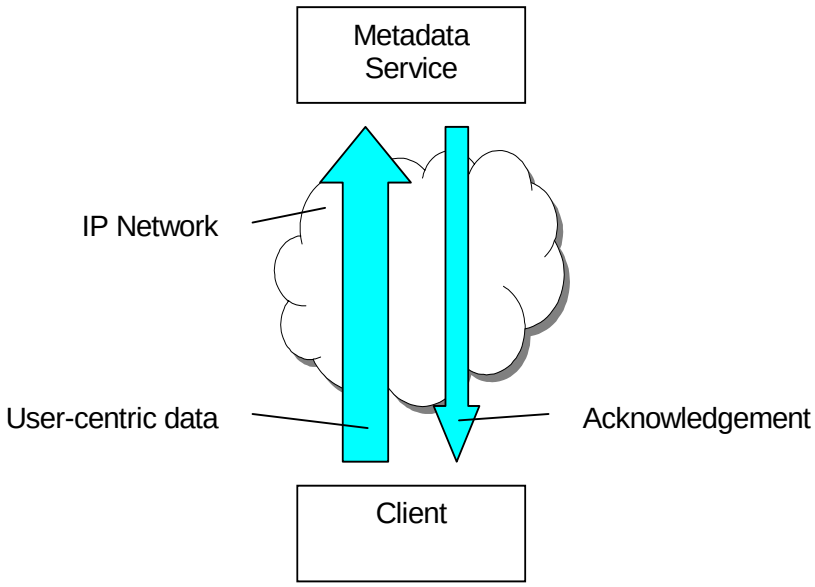


NOTE 1: Any party capable of delivering compliant *TV-Anytime* data could provide a metadata service. Examples include: content creators, content providers, service providers, consumer electronics manufacturers and third parties aggregation services.
NOTE 2: The request contains parameters that specify the type of metadata required by the client.
NOTE 3: The types of metadata returned could be any of the non user-centric data specified in the TS 102 822-3-1 [12] (i.e. the fragments defined in clause 4.3.1.1 of TS 102 322-3-2 [13]), along with content referencing information.

Figure 2: Client requesting metadata from a metadata service

4.1.2 Submission of User-centric Metadata

Submission of user-centric metadata offers a number of possible benefits to both consumers and metadata service providers. These are described in clause 6.5 of the TS 102 822-3-1 [12]. Ensuring the privacy of these transactions and that the metadata service provider is trustworthy is essential to the submission of user-centric metadata. The means by which this is ensured is not defined by the present document.



NOTE 1: In this case, the metadata service is a user data aggregator. Any of the parties listed for figure 2, or any other party capable of usefully exploiting *TV-Anytime* usage information in a trustworthy fashion, could provide the metadata service.

NOTE 2: In principle, the user-centric data may be any of the types defined in the Metadata Specification (TS 102 822-3-1[12] (e.g. `UserPreferences` and `UsageHistory`). For the purposes of this version of the specification, only a submission of carefully constrained, anonymous `UsageHistory` instances is allowed.

Figure 3: Client submitting user-centric data to a service provider

4.2 Metadata Service Capability Descriptions

In order to exploit usefully the metadata services described in the previous clause, the client needs information about the nature of the metadata service being offered. This is because different metadata services will provide different types of metadata and can be queried in different ways. For example, some metadata services may offer just content referencing information, whilst others may provide programme metadata but no segmentation information. Similarly, whereas one server may only be able to accept simple requests for metadata based on a CRID, another server may offer much more sophisticated querying and sorting capabilities. Moreover, different types of queries are only useful if a client is able to establish sensible values with which to query. An example of this is a query for scheduling data (`ProgramLocationTable`). In order to query for scheduling information on a particular content delivery service, the client needs to know the content delivery services for which that metadata service has data.

To address this issue, each metadata service provides, on request from a client, a capability description. This capability description allows a client to flexibly query a metadata service, without making requests that will not be supported by that metadata service. Furthermore, it allows metadata service providers to flexibly implement the server in a way that is appropriate to the data that they have available.

4.3 Metadata Service Discovery

Metadata service discovery is the process by which a client establishes a URL where a *TV-Anytime* metadata service can be found. There are a number of ways this process can occur, but only the third method (see clause 4.3.3) is addressed by the present document.

4.3.1 Non-standardized Discovery

A number of methods exist for discovering the URLs of metadata services that will not be standardized by the *TV-Anytime* Forum. The following list gives some examples.

- The client might be pre-programmed with a set of URLs that refer to one or more metadata services. This will typically be useful in a vertical market, or tightly controlled horizontal market.
- A user might manually enter a URL of a new metadata service he is interested in, using some means of text input.
- The software on a client may be updated using software updates delivered via a unidirectional broadcast, or over the return channel.

4.3.2 Unidirectional Delivery of Discovery Information

TS 102 822-2 [11] and TS 102 822-4 [14] define ways, or requirements on the underlying transport, in which the URL of a bi-directional metadata and/or content referencing service can be discovered from *TV-Anytime* information inserted in a unidirectional stream. The present document defines how a client can usefully exploit the discovered service using the resulting URL.

4.3.3 Client-Initiated Discovery Using the Bi-directional Network

This mode of metadata service discovery involves using the bi-directional network to access a "Yellow Pages" of *TV-Anytime* metadata services. The mechanism is based upon W3C standards for web service discovery (UDDI [19] and WS-Inspection [21], the use of which is standardized by the *TV-Anytime* Forum, according to the rules given in clause 5 in TS 102 822-6-2 [15]. Support for these discovery techniques by clients and servers is OPTIONAL.

5 Metadata Service Types

The *TV-Anytime* Forum defines two types of metadata web service. Each type of web service can be thought of as a remote procedure with a well-defined set of inputs and outputs and a well-defined behaviour. In WSDL [20] terminology, this remote procedure is known as an operation (see annex A). The metadata retrieval operation (see clause 4.1.1 is called the `get_Data` operation, and the user description submission operation (see clause 4.1.2) is called the `submit_Data` operation.

The types used in the requests and responses to *TV-Anytime* metadata services are defined in the target namespace: "urn:tva:transport:2002" . This allows Schema aware tools to validate the various messages. The types defined in TS 102 822-3-1 [12] and TS 102 822-4 [14] schemas are referenced in the transport namespace (using XML Schema's import mechanism).

The Schema fragments in the following clauses are all defined within this namespace. The corresponding namespace qualifier used in these Schema fragments is "`tns`:" The complete XML Schema file (`tva_transport_types_v10.xsd`) may be found attached to the present document as part of a common Zip file.

5.1 get_Data Operation

The `get_Data` operation allows a client to query a server in order to retrieve *TV-Anytime* data for a set of programmes or programme groups. The following list gives a few examples of the types of functionality that a *TV-Anytime* metadata provider can offer using a `get_Data` operation.

- Operation that takes a list of CRIDs and returns content referencing data for those CRIDs.
- Operation that takes a list of CRIDs and returns *TV-Anytime* metadata for those CRIDs.
- Operation that accepts a query for programmes with particular metadata attributes (e.g. with a particular genre, or starring a certain actor, etc.) and returns matching programmes.

- Operation that accepts a query for programmes broadcast at a certain time or on a certain channel and returns matching programmes.

A `get_Data` operation can, in principle, support all these types of queries, as well as more complex queries, involving a wide range of metadata constraints, and logical combinations of those constraints.

5.1.1 Request Format

The request format allows the client to specify three types of parameters. The semantics and structure of these three parameters are explained in more detail in the subsequent clauses.

```
<element name="get_Data" type="tns:get_Data"/>
  <complexType name="get_Data">
    <sequence>
      <element name="QueryConstraints">
        <complexType>
          <choice>
            <element name="PredicateBag" type="tns:PredicateBagType"/>
            <element name="BinaryPredicate" type="tns:BinaryPredicateType"/>
            <element name="UnaryPredicate" type="tns:UnaryPredicateType"/>
          </choice>
        </complexType>
      </element>
      <element name="RequestedTables" type="tns:RequestedTablesType"/>
    </sequence>
    <attribute name="maxPrograms" type="unsignedInt"/>
  </complexType>
```

Name	Definition
`QueryConstraints`	A REQUIRED parameter that defines, by means of one or more logical predicates the set of "result records" that the client is interested in. See clause 5.1.1.4 for a definition of "result record". For example, this set could be specified using a list of CRIDs, the programmes that have the keyword "Thriller" in their metadata, or the programmes that correspond to a BroadcastEvent on Channel 2 on Saturday, etc.
`RequestedTables`	A REQUIRED parameter that specifies the view of the metadata required by the client. This parameter determines the types of metadata that are returned for each result, and also allows the client to specify simple sort criteria to be applied to the result.
`maxPrograms`	An OPTIONAL upper limit on the number of programmes that will be returned. This is to prevent the client being overloaded by very large result sets. If this attribute is not there the server should return all the matching results.

5.1.1.1 Query constraint parameters

This parameter consists of a set of logical predicates, which can be nested together (according to the rules of first order predicate logic) and together define the set of results that a client is interested in. (Using SQL terminology, this parameter plays a similar role to the WHERE clause in an SQL SELECT statement.) Unlike SQL, the interpretation of the logical constraints does not have to be strict - the results returned do not need to precisely match the criteria specified in the query. This provides opportunity for metadata service providers to offer value adding search algorithms.

```
  <complexType name="PredicateBagType">
    <sequence maxOccurs="unbounded">
      <choice>
        <element name="PredicateBag" type="tns:PredicateBagType"/>
        <element name="BinaryPredicate" type="tns:BinaryPredicateType"/>
        <element name="UnaryPredicate" type="tns:UnaryPredicateType"/>
      </choice>
    </sequence>
    <attribute name="contextNode" type="tns:contextNodeIDType"/>
    <attribute name="negate" type="boolean" default="false"/>
```

```
    <attribute name="type" type="tns:PredicateBagTypeType"/>
  </complexType>
  <simpleType name="PredicateBagTypeType">
    <restriction base="string">
      <enumeration value="AND"/>
      <enumeration value="OR"/>
    </restriction>
  </simpleType>
  <complexType name="BinaryPredicateType">
    <attribute name="fieldID" type="tns:fieldIDType" use="required"/>
    <attribute name="fieldValue" type="string" use="required"/>
    <attribute name="test" default="equals"
       type="tns:BinaryPredicateTestType"/>
  </complexType>
  <complexType name="UnaryPredicateType">
    <attribute name="fieldID" type="tns:fieldIDType" use="required"/>
    <attribute name="test" default="exists"
       type="tns:UnaryPredicateTestType"/>
  </complexType>
  <simpleType name="BinaryPredicateTestType">
    <restriction base="string">
      <enumeration value="equals"/>
      <enumeration value="not_equals"/>
      <enumeration value="contains"/>
      <enumeration value="greater_than"/>
      <enumeration value="greater_than_or_equals"/>
      <enumeration value="less_than"/>
      <enumeration value="less_than_or_equals"/>
    </restriction>
  </simpleType>
  <simpleType name="UnaryPredicateTestType">
    <restriction base="string">
      <enumeration value="exists"/>
    </restriction>
  </simpleType>
```

Name	Definition
PredicateBagType	A PredicateBagType contains one or more BinaryPredicate, UnaryPredicate or PredicateBag children, and is used to express logical relationships between these children.
type	This attribute expresses the logical relationship between the children of this PredicateBag. It can take the values "AND" or "OR". The attribute is REQUIRED when there are two or more children. The attribute is meaningless if the PredicateBag contains a single element.
negate	This attribute reverses the Boolean evaluation (true/false) of the PredicateBag. If there are two or more children predicates, it is evaluated after the children predicates have been combined; i.e. the type attribute has tighter precedence.
contextNode	If present, this attribute restricts the way in which fragments are matched with this PredicateBag. Specifically, the predicates contained within this PredicateBag MUST all be satisfied by the fields in a single XML element within the fragment. The root node of this XML element is defined by the contextNodeID value of the contextNode attribute. The contextNodeID values used here MAY have a definition that uses a multiple XPath expression (i.e. the contextNodeID may refer to multiple nodes, but the actual node is given by the RequestedTable element). When used, all the contextNodeID values referenced in the predicates that are descendants of this PredicateBag MUST refer to fields that are descendants of the contextNode element in the XML Schema.
UnaryPredicateType/BinaryPredicateType	A Boolean test that can be evaluated on a field stored by a metadata service. See clause 5.1.1.1.1 for a definition of field.
fieldID	A REQUIRED fieldID value (see clause 5.1.1.1.1) that defines the field being tested (e.g. CRID, Title, etc.).
test	The relationship between the fieldID and the fieldValue. This attribute can take the value "equals", "not_equals", "contains", "greater_than", "greater_than_or_equal", "less_than", "less_than_or_equals", or "exists".
fieldValue	The value being tested. This attribute is not present in predicates of the type UnaryPredicateType.

Annex C provides some examples that illustrate the types of queries a client can issue.

5.1.1.1.1 Identifying contextNodes

A contextNode is an XML node of the following type: element.

- Element contextNode can correspond to nodes with an XML Schema complexType model (an element containing other elements or attributes).

ContextNodes are defined using an XPath expression based on a subset of XPath (for a definition of this XPath subset, see TS 102 822-3-1 [12]). Instead of using this verbose XPath expression directly in every predicate, all fields used in a query MUST be assigned a contextNode. A contextNode is just a syntactic shortcut that provides a consistent alias for the full XPath expression that identifies a node. The *TV-Anytime* Forum defines a normative subset of nodes with predefined contextNode values.

In order to define a list of contextNode definitions the following piece of XML Schema is used.

```
  <element name="ContextNodeIDDefinitionList"
           type="tns:ContextNodeIDDefinitionListType">
    <key name="UniqueContextNode">
      <selector xpath="tns:ContextNodeIDDefinition"/>
      <field xpath="@contextNodeID"/>
    </key>
  </element>
 <complexType name="ContextNodeIDDefinitionListType">
    <sequence>
      <element name="ContextNodeIDDefinition" maxOccurs="unbounded">
        <complexType>
          <attribute name="contextNode" type="NCName"/>
```

```
          <attribute name="contextNodeDefinition"
type="tns:contextNodeDefinitionListType"/>
        </complexType>
      </element>
    </sequence>
    <attribute name="targetNamespace" type="anyURI" use="required"/>
  </complexType>
  <simpleType name="contextNodeDefinitionType">
    <restriction base="token">
      <pattern value="(/((\i\c*:)?(\i\c*)))*((/text\(\))|(/@((\i\c*:)?(\i\c*))))?"/>
    </restriction>
  </simpleType>
  <simpleType name="contextNodeDefinitionListType">
    <list itemType="tns:contextNodeDefinitionType"/>
  </simpleType>
  <simpleType name="contextNodeIDType">
    <restriction base="QName"/>
</simpleType>
```

Name	Definition
ContextNodeIDDefinitionListType	Provides a uniform structure for defining a map of contextNode values to XPath expressions.
targetNamespace	The field definition namespace to which this contextNode map belongs. The namespace can be used to unambiguously reference contextNode values from multiple namespaces. The use of namespaces here is similar to, but distinct from, the use of namespaces to identify XML Schema, and is in accordance with Namespaces in XML [2].
ContextNodeIDDefinition	Contains a single contextNode map.
ContextNodeId	A name used to identify this field. All the contextNode values defined in a particular namespace MUST be unique.
ContextNodeDescription	A list of XPath expressions that defines the fields that MAY be searched when this contextNode is used. Each item in the list is a regular expression that corresponds to the XPath subset defined in TS 102 822-3-1 [12], clause 4.8.5.1.2. The XPath namespace context consists of the namespace declarations that are in scope in the XML instance document at the point that this attribute appears.
contextNodeIDType	The simple type that all contextNode values are based on. Note that the use of QName ensures that contextNode values are always namespace qualified.

Element instances based upon the ContextNodeIDDefinitionListType are used to specify the *TV-Anytime* Forum's predefined subset of contextNode values.

The predefined subset of contextNode allocations, which belong in the urn:tva:transport:contextNodeIDs:2002 field definition namespace, can be found in annex B. Within the present document the prefix "tvac" is used to denote this namespace.

Within this same namespace, some contextNode identifiers contain multiple XPath statements. A server HAS to use the specific Xpath statement relevant to the considered RequestedTable when performing a query based on these contextNodes as it does for the CRID field. Table 2 defines some additional special case contextNode identifiers, along with a semantic description of their meaning. These special case contextNodes are those that cannot be identified using single XPath expressions.

Table 1: Description of special case `contextNodeID` values (informative)

ContextNodeID	Comment
`CreditsItem`	The CreditsItem element of the CreditsList element within the ProgramInformation or GroupInformation table.
`AudioAttributes`	The AudioAttributes element of the AVAtrributes element within the ProgramInformation or ProgramLocation tables.
`VideoAttributes`	The VideoAttributes element of the AVAtrributes element within the ProgramInformation or ProgramLocation tables.
`AwardsListItem`	The AwardsListItem element of the AwardsList element within the ProgramInformation or GroupInformation tables.
`ProgramInformation`	The ProgramInformation element of the ProgramInformation table.
`GroupInformation`	The GroupInformation element of the GroupInformation table.
`BroadcastEvent`	The BroadcastEvent element of the ProgramLocation table.
`Schedule`	The Schedule element of the ProgramLocation table.
`OnDemandProgram`	The OnDemandProgram element of the ProgramLocation table.
`ServiceInformation`	The ServiceInformation element of the ServiceInformation table.
`PersonName`	The PersonName element of the CreditsInformation table.
`OrganizationName`	The OrganizationName element of the CreditsInformation table.
`SegmentInformation`	The SegmentInformation element of the SegmentInformation table.
`SegmentGroupInformation`	The SegmentGroupInformation element of the SegmentInformation table.
`Review`	The Review element of the ProgramReview table.

5.1.1.1.2 Identifying fields

A field is an XML node of the following type: attribute, text or element.

- Attribute fields and text fields (the contents of an element's text node) have no structure, so can be represented as a string whose value can be tested against the `fieldValue` given in a predicate.
- Element fields can correspond to nodes with an XML Schema `complexType` model (an element containing other elements or attributes). Fields of this type can only be used in a predicate if the test is "exists". An example of why this might be useful is given in annex C, example 8. Sorts cannot be based upon an element field.

Fields are defined using an XPath expression based on a subset of XPath (for a definition of this XPath subset, see TS 102 822-3-1 [12]). Instead of using this verbose XPath expression directly in every predicate, all fields used in a query MUST be assigned a `fieldID`. A `fieldID` is just a syntactic shortcut that provides a consistent alias for the full XPath expression that identifies a field. The *TV-Anytime* Forum defines a normative subset of fields with predefined `fieldID` values.

In order to define a list of `fieldID` definitions the following piece of XML Schema is used.

```
  <element name="FieldIDDefinitionList"
           type="tns:FieldIDDefinitionListType">
    <key name="UniqueField">
      <selector xpath="tns:FieldIDDefinition"/>
      <field xpath="@fieldID"/>
    </key>
  </element>
  <complexType name="FieldIDDefinitionListType">
    <sequence>
      <element name="FieldIDDefinition" maxOccurs="unbounded">
        <complexType>
          <attribute name="fieldID" type="NCName"/>
          <attribute name="fieldDefinition"
                     type="tns:fieldDefinitionListType"/>
        </complexType>
      </element>
    </sequence>
    <attribute name="targetNamespace" type="anyURI" use="required"/>
  </complexType>
  <simpleType name="fieldDefinitionType">
```

```
    <restriction base="token">
      <pattern value="(/((\i\c*:)?(\i\c*)))*((/text\(\))|(/@((\i\c*:)?(\i\c*))))?"/>
    </restriction>
  </simpleType>
  <simpleType name="fieldDefinitionListType">
    <list itemType="tns:fieldDefinitionType"/>
  </simpleType>
  <simpleType name="fieldIDType">
    <restriction base="QName"/>
  </simpleType>
  <simpleType name="fieldIDListType">
    <list itemType="tns:fieldIDType"/>
  </simpleType>
```

Name	Definition
FieldIDDefinitionListType	Provides a uniform structure for defining a map of fieldID values to XPath expressions.
targetNamespace	The field definition namespace to which this fieldID map belongs. The namespace can be used to unambiguously reference fieldID values from multiple namespaces. The use of namespaces here is similar to, but distinct from, the use of namespaces to identify XML Schema, and is in accordance with "Namespaces in XML" [2].
FieldIDDefinition	Contains a single fieldID map.
fieldID	A name used to identify this field. All the fieldID values defined in a particular namespace MUST be unique.
fieldDefinition	A list of XPath expressions that defines the fields that MAY be searched when this fieldID is used. Each item in the list is a regular expression that corresponds to the XPath subset defined in TS 102 822-3-1 [12], clause 4.8.5.1.2. The XPath namespace context consists of the namespace declarations that are in scope in the XML instance document at the point that this attribute appears.
fieldIDType	The simple type that all fieldID values are based on. Note that the use of QName ensures that fieldID values are always namespace qualified.
fieldIDListType	A whitespace-separated list of fieldID values

Element instances based upon the FieldIDDefinitionListType are used in two places in the present document. Firstly, it is used to specify the *TV-Anytime* Forum's predefined subset of fieldID values. Secondly, it MAY appear in a get_Data operation's capability description, where it is used to allocate fieldID values to fields not in the normative subset. See clause 7.1.3 for an explanation of how this is done. In this, way servers can extend their metadata services to flexibly support querying or sorting on any field in the *TV-Anytime* metadata and content referencing Schema.

The predefined subset of fieldID allocations, which belong in the urn:tva:transport:fieldIDs:2002 field definition namespace, can be found in annex B. Within the present document the prefix "tvaf" is used to denote this namespace.

Within this same namespace, some field identifiers contain multiple XPath statements. A server MAY use any of these XPath statements when performing a query based on these fields. Table 2 defines some additional special case field identifiers, along with a semantic description of their meaning. These special case fields are those that cannot be identified using single XPath expressions.

Table 2: Description of special case fieldID values (informative)

FieldID	Comment
CRID	The CRID when it is used in the context of identifying the CRID with which a fragment is associated. Details of this fieldID can be found in table 3.
Start	The Start attribute of the DecomposedLocator element from the LocationResult element within the content referencing table.

FieldID	Comment
Title	Any title element (Title or ShortTitle) from the ProgramInformation, GroupInformation, ProgramLocation or SegmentInformation tables.
TitleLanguage	The language attribute of the Title element from the relevant ProgramInformation, GroupInformation, ProgramLocation or SegmentInformation table.
Synopsis	Any Synopsis element from the ProgramInformation, GroupInformation, ProgramLocation or SegmentInformation tables.
SynopsisLanguage	The language attribute of the Synopsis element from the ProgramInformation, GroupInformation, ProgramLocation or SegmentInformation tables.
PublishedStart	The PublishedTime element of any items of type ScheduleEventType, and the StartOfAvailability element of a OnDemandProgram item. If the StartOfAvailability element is not present, it is assumed to be the current time (i.e. available now).
PublishedDuration	The PublishedDuration element of any items of type ScheduleEventType.
FragmentID	Any fragmentId attribute. See Annex C for an example of how this fieldID may be used to retrieve a specific fragment.
fragmentVersion	Any fragmentVersion attribute. See Annex C for an example of how this fieldID may be used to update a specific fragment.
AudioCoding	The coding element from the AudioAttributes element from the ProgramInformation or ProgramLocation tables.
AudioChannels	The NumOfChannels element from the AudioAttributes element within the ProgramInformation or ProgramLocation tables.
VideoAspectRatio	The AspectRatio element from the VideoAttributes element within the ProgramInformation or ProgramLocation tables.
Keyword	The Keyword element within the ProgramInformation or GroupInformation tables.
KeywordLanguage	The language attribute from the Keyword element within the ProgramInformation or GroupInformation tables.
Genre	The Genre element within the ProgramInformation or GroupInformation tables.
Language	The Language element from the ProgramInformation or GroupInformation tables.
ParentalGuidance	The ParentalGuidance element from the ProgramInformation or GroupInformation tables.
Role	The Role element from the cast list item that relates to cast member associated with the ProgramInformation or GroupInformation tables.
FamilyName	The FamilyName element from the cast list item that relates to cast member associated with the ProgramInformation or GroupInformation tables.
GivenName	The GivenName element from the cast list item that relates to cast member associated with the ProgramInformation or GroupInformation tables.
AwardTitle	The Title element from an awards list item within the ProgramInformation or GroupInformation tables.
AwardYear	The Year element from an awards list item within the ProgramInformation or GroupInformation tables.
AwardNominee	The Nominee element from an awards list item within the ProgramInformation or GroupInformation tables.
AwardRecipient	The Recipient element from an awards list item within the ProgramInformation or GroupInformation tables.
RatingValue	The RatingValue element from Review element within the ProgramInformation or GroupInformation tables.
RatingScheme	The RatingScheme element from Review element within the ProgramInformation or GroupInformation tables.
ProductionDate	The ProductionDate element from BasicDescription element within the ProgramInformation or GroupInformation tables.

FieldID	Comment
CreditName	Any name element (FamilyName or GivenName) form the ProgramInformation, GroupInformation or ProgramReviews tables.
ProgramURL	The ProgramURL element from the BroadcastEvent Schedule, or OnDemandProgram element within the ProgramLocation table.
FreeToView	The Free element from the BroadcastEvent or Schedule element within the ProgramLocation table.
EpisodeOf	The EpisodeOf element from the ProgramInformation element within the ProgramInformation table.
GroupType	The value attribute of the GroupType element within the GroupInformation table.
ServiceURL	The ServiceURL element of the ServiceInformation element within the ServiceInformation table.
ServiceName	The ServiceName element of the ServiceInformation element within the ServiceInformation table.
CreditsItem	The CreditsItem element of the CreditsList element within the ProgramInformation or GroupInformation table.
AudioAttributes	The AudioAttributes element of the AVAtrributes element within the ProgramInformation or ProgramLocation tables.
VideoAttributes	The VideoAttributes element of the AVAtrributes element within the ProgramInformation or ProgramLocation tables.
AwardsListItem	The AwardsListItem element of the AwardsList element within the ProgramInformation or GroupInformation tables.
ProgramInformation	The ProgramInformation element of the ProgramInformation table.
GroupInformation	The GroupInformation element of the GroupInformation table.
BroadcastEvent	The BroadcastEvent element of the ProgramLocation table.
Schedule	The Schedule element of the ProgramLocation table.
OnDemandProgram	The OnDemandProgram element of the ProgramLocation table.
ServiceInformation	The ServiceInformation element of the ServiceInformation table.
PersonName	The PersonName element of the CreditsInformation table.
OrganizationName	The OrganizationName element of the CreditsInformation table.
SegmentInformation	The SegmentInformation element of the SegmentInformation table.
SegmentGroupInformation	The SegmentGroupInformation element of the SegmentInformation table.
Review	The Review element of the ProgramReview table.
CSUri	The uri attribute of the ClassificationScheme element within the ClassificationSchemeTable.
CSAlias	The mpeg7:alias attribute of the CSAlias element within the ClassificationSchemeTable.
GenreCS	The href attribute of the Genre element within the ProgramInformationTable and GroupInformationTable.

If a metadata service wishes to support querying or sorting on a field within the normative subset it MUST use the predefined `fieldID.` (I.e. servers MUST NOT define their own `fieldID` for a field already in the normative subset.)

5.1.1.1.3 Primary index CRID fields

Fragments containing CRIDs exist in many different locations in the *TV-Anytime* schema. Since CRIDs can occur in multiple places within a fragment (e.g. in a `GroupInformation` fragment they can be used to refer to parent groups), the following table formally defines the CRID field that is used to identify the fragments in each case.

The XPath expressions are namespace qualified. The present document assumes the expression content of the XPath evaluator has the following namespace prefixes:

cr	urn:tva:ContentReferencing:2002
tva	urn:tva:metadata:2002
mpeg7	urn:mpeg:mpeg7:schema:2001

Table 3: The meaning of the CRID field in the different *TV-Anytime* tables

Table type	XPath specification for CRID node set (ignore whitespace)	Number matches
ContentReferencing	/cr:ContentReferencingTable/cr:Result/@CRID	0 or 1
ProgramInformation	/tva:TVAMain/tva:ProgramDescription/tva:ProgramInformationTable/tva:ProgramInformation/@programId	0 or 1
GroupInformation	/tva:TVAMain/tva:ProgramDescription/tva:GroupInformationTable/tva:GroupInformation/@groupId	0 or 1
ProgramReview	/tva:TVAMain/tva:ProgramDescription/tva:ProgramReviewTable/tva:ProgramReviews/tva:Program/@crid	0 to many
ProgramLocation	/tva:TVAMain/tva:ProgramDescription/tva:ProgramLocationTable/tva:Schedule/tva:ScheduleEvent/tva:Program/@crid	0 to many
	/tva:TVAMain/tva:ProgramDescription/tva:ProgramLocationTable/tva:BroadcastEvent/tva:Program/@crid	0 to many
	/tva:TVAMain/tva:ProgramDescription/tva:ProgramLocationTable/tva:OnDemandProgram/tva:Program/@crid	0 to many
	/tva:TVAMain/tva:ProgramDescription/tva:ProgramLocationTable/tva:OnDemandService/tva:OnDemandProgram/tva:Program/@crid	0 to many
SegmentInformation	/tva:TVAMain/tva:ProgramDescription/tva:SegmentInformationTable/tva:SegmentList/tva:SegmentInformation/tva:ProgramRef/@crid The schema does not require this attribute. If it is omitted the implied CRID (from the parent segment group) MUST be used for the server to determine matching SegmentInformation fragments based on the CRID field.	0 to many
	/tva:TVAMain/tva:ProgramDescription/tva:SegmentInformationTable/tva:SegmentGroupList/tva:SegmentGroupInformation/tva:ProgramRef/@crid	0 to many

5.1.1.1.4 Restrictions on the use of QueryConstraints

The `QueryConstraints` element MUST NOT contain predicates with fields from the following tables:

- `ContentReferencingTable`. Content referencing information does not contain metadata attractors, so none of the fields are suitable as query constraints.
- `CreditsInformationTable`. Credits information is considered to always be inlined from the point of view of specifying a search. In other words, fields based upon the credit information fields inside a `BasicDescription` element are used to constrain a search.

There are two exceptions to these rules:

- Searches using the `fragmentId` field and `fragmentVersion` field can be used to retrieve or update a metadata fragment from any metadata table.
- Searches based on the `CRID` field are a special case, which is considered in clause 5.1.1.1.3.

Please note that a request with "QueryConstraints" containing disjunctive condition may lead sometimes to an ambiguous result (see example in clause C.9).

5.1.1.1.5 Evaluating a predicate

Although the `fieldID` values specify precisely the field being matched, a metadata service is free to interpret which metadata fields are used to match the query `fieldValue`. For example, if a query specifies a programme with a certain title, a server is free to match this string with any of the values given by the `Title` or `ShortTitle` elements. Similarly, if a query specifies a keyword search, a server could match the keyword on any of the stored keywords as well as words in the title and synopsis. In general, where the metadata schema offers alternative ways of representing the same information a metadata service SHOULD be lenient in matching either representation (e.g. a UK parental guidance of "18" might be considered to match an American rating of "X"). In particular, it is RECOMMENDED that a metadata service adopt the following behaviour.

- If a controlled term is being matched, the server matches `fieldValue` values specified using both the full URN for the controlled term, or a shortened alias form.
- Any query field inside a `BasicDescription` element can give rise to a match for the corresponding field inside a `ProgramInformation` or `GroupInformation` fragment. E.g. a search for the field `Title` with value "Friends" could give rise to a programme CRID (episode of "Friends"), or a group CRID (series of "Friends"). If the client is only requested in one type of CRID, they can achieve this by requesting the appropriate table type in the `RequestedTables` parameter.
- The fields within a `BroadcastEvent` can be considered to the match corresponding fields within a `ScheduleEvent`. This does not impact how the metadata service chooses to fragment the returned data (i.e. whether to use `BroadcastEvent` or `Schedule` fragments).

The tests evaluated to determine the Boolean value of a predicate are self-explanatory, but the following should be noted.

- *equals*. For numeric types, a server MUST consider values that are numerically equal to the query value to represent a match. For fields that have the `anyURI` type or some derived type (e.g. CRID, ServiceURL, etc.) equality MUST be based upon the rules defined for that particular URI. For fields that have the `TVAIDType type` (e.g. fragmentID, serviceID, etc.), equality MUST be based on the rules for matching `ID` and `IDRef` types, as defined in the XML specification [1].
- For other types derived from string, the server determines the notion of equality. It is RECOMMENDED that the server ignores leading and trailing white space and character case when comparing strings. The server may also employ fuzzy or other type of matching algorithms so that matches may still be returned if a name is misspelt in a query, say.
- For fields that can contain multiple values (e.g. Title, Genre, Keywords, etc.), it is RECOMMENDED that the query `fieldValue` is matched to any of the field values in a fragment. If the server has no data for a particular field, it is RECOMMENDED that the server considers the associated fragment not to match the predicate, as this can lead to very large result sets. However, if the query is sufficiently restrictive (no truncation of the response is necessary) it may be appropriate to relax this rule.
- *less_than*, *greater_than*. The use of XPath and XML Schema mean that the metadata service is always aware of the Schema type of every field. If the type is duration, then the shortest time is considered to be the lesser. If the type is a date and/or time, then the earlier value is considered to be the lesser. If the type is numeric, then *less_than* has its obvious interpretation.
- If the type is derived from a string, then the earliest string (according to a lexicographic sort) is considered to be the lesser. The comparison is based on the Unicode Technical Standard 10 Collation Order [17] on elements normalized according to Unicode Normalization Form C [18]. By default, the collation takes place according to the Default Unicode Collation Element Table in conjunction with Unicode Collation Algorithm. A metadata provider may choose to use another collation table, in which case this is indicated by naming the collation table in the capability description (see clause 6.1).
- For fields that can contain multiple values (e.g. Title, Genre, Keywords, etc.), the metadata service should base the comparison on the field it considers to be primary. For some elements, this is indicated using the `type` attribute with value "main".
- If a fragment has no metadata for a particular field, the fragment SHOULD be tested as if the value for that field is the empty string.

- *exists.* In order for this test to evaluate as "true", the metadata service MUST be able to populate the element or attribute with useful data. I.e. it is not sufficient to instantiate the element or attribute but leave it empty.

- *contains*. This test is only used to test fields that have a type derived from a string. For all other field types the test SHOULD not be used.

5.1.1.2 View on returned data

This REQUIRED `RequestedTables` parameter defines the types of data that a metadata service will try to provide on each result it returns. To achieve this, the client includes a list of table types that it requires. For a given result set, a server SHOULD supply, if the appropriate metadata is available, the corresponding fragments for this result record from the requested tables. It is not an error if the appropriate metadata is not available (e.g. no segmentation information is available for that CRID), and the client MUST NOT assume that requested fragments will always be returned.

```
<complexType name="RequestedTablesType">
  <sequence>
    <element name="Table" maxOccurs="unbounded">
      <complexType>
        <sequence>
          <element name="SortCriteria" type="tns:SortCriteriaType"
                   minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="type" use="required">
          <simpleType>
            <restriction base="string">
              <enumeration value="ContentReferencingTable"/>
              <enumeration value="ClassificationSchemeTable"/>
              <enumeration value="ProgramInformationTable"/>
              <enumeration value="GroupInformationTable"/>
              <enumeration value="CreditsInformationTable"/>
              <enumeration value="ProgramLocationTable"/>
              <enumeration value="ServiceInformationTable"/>
              <enumeration value="ProgramReviewTable"/>
              <enumeration value="SegmentInformationTable"/>
            </restriction>
          </simpleType>
        </attribute>
      </complexType>
    </element>
  </sequence>
</complexType>
```

Name	Definition
`RequestedTables`	A list of table elements containing at least one entry.
`Table`	Specifies the type of table required by the client and the sort criteria to be applied to that table.
`Type`	Mandatory parameter giving the type of *TV-Anytime* table required by the client. Can take the values: `ContentReferencingTable`, `ClassificationSchemeTable`, `ProgramInformationTable`,`GroupInformationTable`, `ProgramLocationTable`, `ServiceInformationTable`, `ProgramReviewTable`, `SegmentInformationTable`, and `CreditsInformationTable`.
`SortCriteria`	The sort criteria to be applied to that table. If not present the returned table will not necessarily be sorted. The order of the `SortCriteria` element is significant See clause 5.1.1.2.1 for a more complete explanation of sorting.

A metadata service can choose whether or not to inline credits information directly, or to reference a fragment in the `CreditsInformationTable`. The request for a `CreditsInformationTable` determines if the server returns credits information at all, but does not impact the above choice on how to include the credits information. A server MUST NOT return credits information if the `CreditsInformationTable` is not listed in the `RequestedTable` parameter. The `SortCriteria` parameter SHALL be ignored if it is used in conjunction with the `CreditsInformationTable`.

A `ServiceInformationTable` will always be included in a response if a `ServiceInformation` entry is being referenced (using a `serviceIDRef`) from a `ProgramLocationTable`. Thus, an explicit request for a `ServiceInformationTable` will have no effect if a `ProgramLocationTable` is being returned. Nevertheless, if the client wishes to specify sort criteria for the `ServiceInformationTable`, it is useful to include it in the requested tables.

5.1.1.2.1 Sort criteria

```
<complexType name="SortCriteriaType">
  <attribute name="fieldID" type="tns:fieldIDType" use="required"/>
  <attribute name="order" type="tns:SortingOrderType"
             default="ascending"/>
</complexType>
<simpleType name="SortingOrderType">
  <restriction base="string">
    <enumeration value="ascending"/>
    <enumeration value="descending"/>
  </restriction>
</simpleType>
```

Name	Definition
`SortCriteriaType`	The definition of one sorting criteria to apply.
`fieldID`	Specifies the field to use
`Order`	Defines the type of sorting to apply (either ascending or descending).

Sorts are applied across the fragments contained in a given table. Sorts can be applied in either ascending order (with the least first) or descending order (with the largest first). The criteria by which fields are compared is identical to that used to make `less_than` (ascending search) and `greater_than` (descending search) comparisons, as specified in clause 5.1.1.1.5. One implication of this is that fragments included in the response containing no data for the field being sorted MUST be sorted as if that field had the value of the empty string. Another implication is that if the sort field can have multiple values in a single fragment (e.g. `Title`), the metadata service SHOULD be sorted on the primary value for that field.

There can be a number of `SortCriteria` elements within the `RequestedTables` parameter. The metadata service uses this to apply a multi-tier sort, with the first sort applied using the first `SortCriteria` element in the list, the next sort using the second `SortCriteria` element (if any), and so on. If a server indicates that it supports one or more sort fields (see clause 7.1) for a table, then it MUST support a single-tier sort for that table. If a server indicates that it supports more than one sort field, then it MAY support a multi-tier sort for that table. The exact sort applied by the server is indicated in the response (see clause 5.1.2.1). If multi-tier sorting is supported, groups of fragments that are equivalent according to the criteria of one sort field are then sorted according to the criteria of the next sort field. For example, this allows a set of `BroadcastEvent` fragments to first be grouped into channels and then sorted according to time and date, and thus make the construction of an EPG on the client much more efficient. Such a sort could be specified as follows.

```
<RequestedTables>
  <Table type="ProgramLocationTable">
    <SortCriteria fieldID="tvaf:ServiceURL" order="descending"/>
    <SortCriteria fieldID="tvaf:PublishedTime"/>
  </Table>
</RequestedTables>
```

If the `SortCriteria` parameter contains a `fieldID` value that refers to a field identifier containing multiple XPath expressions, the metadata service MAY choose any one of the XPath expressions for identifying which field to use for sorting that fragment.

For example, if a result is a `ProgramLocationTable` containing both `OnDemandProgram` fragments and `BroadcastEvent` fragments, and the sort criteria is "PublishedStart", all the `BroadcastEvent` fragments will be sorted according to their published time (see table 2) followed by all the `OnDemandProgram` fragments sorted according to their first available time.

The way in which a client establishes the fields for which sorting is supported (if any) is described in clause 7.1.

5.1.1.3 Size limit parameter

An OPTIONAL parameter (`maxPrograms`) can be inserted in the request. It is a positive integer and is used by the client to specify the number of records returned by a query. If a query results in a greater number of records, the server MUST return `maxPrograms` records, along with their corresponding metadata.

Limiting the number of records being returned does not guarantee the client device's memory not being exceeded since there is no limit on the size of each of data associated with each record. Similarly, knowing the exact byte size of the response is not sufficient, since there is not a linear relationship between this figure and the size of the post-parsing in-memory objects. The client SHOULD ensure that memory overloading is gracefully handled by terminating the parsing step before memory limits are reached. The purpose of the `maxPrograms` parameter is to allow the client to avoid this occurrence for the large majority of cases. (See also the explanation of the `truncated` parameter in clause 5.1.2.3, which may be included in the response).

It is RECOMMENDED that when the server truncates a response, it provide the results that it considers to be the closest match to the specified query.

5.1.1.4 Interpretation of Query Predicates

This clause defines the data model that is used to describe how the server processes the query to build its response. This model is necessary to explain the way in which queries operate across tables. It does not imply an underlying implementation, nor does it imply that a response must strictly satisfy the relational calculus definition given here.

The data model described in this clause is purely a tool to describe how queries across tables work. This model uses a set of relational tables in order to explain how query predicates SHOULD be interpreted. The data model itself is informative.

When all the query predicates contain `fieldID` values belong to the same table (e.g. a query based on `Title` and `Keyword` fields) the result records that satisfy the query constraints will be matched. If the query predicates contain `fieldID` values belonging to multiple tables, then the matching result records should satisfy those predicates in all the relevant tables. The data model is a tool to explain this behaviour, as it can become complex in the general case.

Firstly, this data model defines a set of relational tables containing the following records for each fragment that matches the query.

- Content referencing record. Each record corresponds to a `Result` element and the table is uniquely keyed with a CRID.
- Program information record. Each record corresponds to a `ProgramInformation` element and the table is uniquely keyed with a CRID.
- Group information record. Each record corresponds to a `GroupInformation` element and the table is uniquely keyed with a CRID.
- Broadcast event record. Each record corresponds to a `BroadcastEvent` or a `ScheduleEvent` element, and can be assigned a CRID.
- On demand programme record. Each record corresponds to an `OnDemandProgram` element and can be assigned a CRID.

- Service information record. Each record corresponds to a `ServiceInformation` element and is uniquely keyed with a `serviceId`.
- Review record. Each record corresponds to a `Review` element and is associated with a CRID.
- Segment information record. Each record corresponds to a `SegmentInformation` element and is associated with a CRID.
- Segment group record. Each record corresponds to a `SegmentGroupInformation` element and is associated with a CRID.

Note that, with one exception: the "Content referencing record", these records correspond exactly to the fragments defined in part B of the Metadata Specification. Within each record the data is not flat (i.e., there are nested and repeated data structures) so in practice this would be a poor relational model for *TV-Anytime*.

We now define a combined table formed by performing a sequence of full outer joins as follows:

1) The service information and broadcast event tables are joined based upon the `serviceId` field.
2) The service information and on demand programme tables are joined based upon the `serviceId` field.
3) The two resulting tables, and all the other tables, are joined based upon the CRID field to leave a single combined table. Since the join operation (Cartesian product) is commutative and associative, the order in which this join is performed is not significant.

Each row of the combined table is termed a "result record". A result record has the following properties:

- There can be many result records containing the same value in the CRID field.
- A result record may have no entries for a particular field or set of fields. This is equivalent to saying that not all types of metadata and metadata tables will be available for every programme.
- The same fragment can appear in multiple result records.
- Each field within a result record can be uniquely identified using a `fieldID` value.

5.1.1.5 Definition of server behaviour

This clause defines the server's behaviour to a query that is assumed by the data model. The server SHOULD return a response containing content referencing and/or metadata fragments based upon the predicates in the `QueryConstraints` that satisfy the query.

The server SHALL return a response that contains only the tables specified by the `RequestedTables` parameter, based on the rules defined in clause 5.1.1.2

Each fragment SHALL be encapsulated in either a `TVAMain` or `ContentReferencingTable` element. The complete response SHALL contain either one `TVAMain` element, one `ContentReferencingTable` element or one instance of both `TVAMain` and `ContentReferencingTable`. The response SHALL have duplicated fragments removed.

5.1.2 Response Format

A `get_Data` response contains an XML instance document containing zero or one instance of the following elements:

- TVAMain.
- ContentReferencingTable.
- InvalidFragments.

```
<element name="get_Data_Result" type="tns:get_Data_ResultType"/>
<complexType name="get_Data_ResultType">
  <sequence>
    <element name="TableSortingInformation"
             type="tns:RequestedTablesType" minOccurs="0"/>
    <element ref="tva:TVAMain" minOccurs="0"/>
    <element ref="cr:ContentReferencingTable" minOccurs="0"/>
    <element name="InvalidFragments"
             type="tns:InvalidFragmentsType" minOccurs="0"/>
  </sequence>
  <attribute name="serviceVersion" type="unsignedInt" use="required"/>
  <attribute name="truncated" type="boolean"/>
```

Name	Definition
TVAMain	An element inside which all metadata items defined in TS 102 822-3-1 [12] are instantiated.
ContentReferencingTable	An element inside which all metadata items defined in TS 102 822-4 [14] are instantiated.
InvalidFragments	Represents list of invalid fragments using the InvalidFragmentsType type. See clause 5.1.2.4 for a definition of this type and how it is used.
ServiceVersion	This parameter indicates the version of the get_Data operation's capability description. See clause 5.1.2.2.
truncated	When this attribute is true, it indicates that the result of the query has been truncated because the number of results exceeded the maxPrograms attribute specified in the request or the server was not able to return all the results. See clause 5.1.2.3.

The instance document returned MUST be XML Schema valid with respect to the appropriate metadata and content referencing schemas. Furthermore, each instance document MUST contain the appropriate TVAIDType type to allow complete dereferencing of all TVAIDRefType nodes within the instance document (e.g. if a Schedule is included, the ServiceInformation entry, referenced via the serviceIDRef, must also be present).

5.1.2.1 Indicating the sorting of the response

The TableSortingInformation element SHALL be included in any response where a sort has been performed on the response.

The first SortCriteria element defines the first field by which the response has been sorted. Each subsequent SortCriteria element (if any) defines the next levels of sorting that has been applied.

For example, the following XML snippet shows a response that has been sorted by service URL and then by published time (note that the syntax is the same as that used to specify the sorting in the query).

```
<TableSortingInformation>
  <Table type="ProgramLocationTable">
    <SortCriteria fieldID="tvaf:ServiceURL"/>
    <SortCriteria fieldID="tvaf:PublishedTime"/>
  </Table>
</TableSortingInformation>
```

5.1.2.2 Indicating the service version

The serviceVersion attribute MUST be included in the response. This parameter indicates the version of the get_Data operation's capability description. When the version is updated a client can retrieve a new capability description for the metadata service (see clause 7.1 for more explanation on how this parameter is used). Note that the serviceVersion attribute indicates nothing about the syntax version in the response (which can be inferred from the XML namespace), or the version of the metadata data in the result (metadata version information is indicated using the fragmentVersion attribute).

5.1.2.3 Truncating the result set

If the OPTIONAL `truncated` attribute has the value "true", the records returned do not represent the entire query result set. This is either a result of the inclusion of the `maxPrograms` attribute in the request, or as a result of the server being overloaded. The actual limit used in the latter case is server specific, but in general should be a sufficiently large number so as to not normally be an issue. No behaviours such as paging mechanisms are defined for retrieving more data after a truncated limit, as this requires more complex client and server implementations (since state must be maintained between queries), and would be complex when the server's database is being asynchronously updated. The intent is to support the average query, while at the same time allowing servers the leeway required to be able to manage adequate performance.

A very large number of responses is usually an indication that the query was insufficiently specific. In most cases, it is possible to refine the query by the alteration of the original parameter set. If truncation occurs as a result of flexible interpretation of the query, it is RECOMMENDED that the server falls back to strict interpretation of the query in order to reduce the number of responses.

5.1.2.4 Updating the result set

The `fragmentVersion` and `fragmentId` identifiers MAY be placed at the fragment level in the `get_Data` response (as defined by the TS 102 822-3-1 [12]). If included, this allows the client to request and update individual fragments. A client may cache fragment version information and be able to ignore future instances of the same metadata if the fragment version number has not been incremented. *TV-Anytime* metadata in the bi-directional environment MAY be updated by the unit of fragments based on the `fragementID` and/or `fragmentVersion` defined in TS 102 822-3-1 [12]. To support the metadata updating in the bi-directional environment, the `fragmentVersion` has the following additional validity constraints.

The `fragmentVersion` attribute SHOULD be either one of an 8-digit or 14-digit unsigned integer. When it is an 8-digit integer, it SHOULD follow the format of YYYYMMDD and represents the last updated date of the fragment. When it is an 14-digit integer, it SHOULD follow the format of YYYYMMDDhhmmss and represents the last updated date and time. YYYY represents year in four digits, MM represents month in two digits, DD represents day of the month in two digits, hh represents hour in two digits (using a 24 hour clock), mm represents minute in two digits, and ss represents second in two digits.

If a receiver wants to update a fragment received by the uni-directional link using the bi-directional link it should make a request for the same fragment with a fragmentVersion with a later date than it receives the fragment by the uni-directional link.

The server indicates its support for this updating mechanism using the capability description, as described in clause 7.1.

If the request for a `fragmentID` does not include the `fragmentVersion`, the server shall return its latest version of the fragment.

```
<complexType name="InvalidFragmentsType">
  <sequence>
    <element name="Fragment" minOccurs="0" maxOccurs="unbounded">
      <complexType>
        <attributeGroup ref="tva:fragmentIdentification"/>
      </complexType>
    </element>
  </sequence>
</complexType>
```

Name	Definition
`InvalidFragmentsType`	An `InvalidFragmentsType` is used to express invalid fragment lists. It contains one or more `Fragment` elements.
`Fragment`	Identifies an invalid fragment using the `fragmentIdentification` attributeGroup (as defined in TS 102 822-3-1 [12]). The meaning of `fragmentID` and `fragmentVersion` in this context is described in this clause.

When the `InvalidFragments` element is included in the `get_Data_Result` element, the client SHOULD delete the fragments saved in the local storage identifiable by the `fragmentID` and/or `fragmentVersion` listed in the `InvalidFragments` element.

A `fragmentID` can be reassigned to other fragments by the publisher, when the original fragment became invalid and is deleted. When a `fragmentID` is reused, to ensure the uniqueness of the `fragmentID`, the server is expected to include the reassigned `fragmentID` and the `fragmentVersion` representing the deleted date or the last modified date of the `InvalidFragments` element, whenever the requested `fragmentVersion` is smaller than the `fragmentVersion` of the deleted fragment.

5.2 submit_Data Operation

The `submit_Data` operation is much simpler than the `get_Data` operation. In this version of the specification, its usage is limited according to the constraints outlined in clause 3.1.2.

5.2.1 Usage and user preference data submission policy (informative)

TV-Anytime phase one technical specifications limit the data that can be submitted to a defined (in clause 3.1.2) set of anonymous profile data that has been created via manual input or intelligent agents based on usage of services and content. It is out of scope for the device to send "all" usage data to "all" potential service providers known to the PDR for Phase One because no authenticated return channel rights management process has been specified yet.

The *TV-Anytime* Forum respects and embraces the basic rights of all viewers and providers. These include preserving the basic right of a content consumer to privacy and acknowledging the legitimate rights of all participants such as content creators and providers, service providers, advertisers and network operators.

It is the content consumer's decision as to the amount of privacy invasion and profiling capabilities done by these participants, and will be allocated by the content consumer to a vendor or service provider at his/her discretion.

Providers that accept the content consumer's choice to allocate to them the responsibility (partial or in full) to profile him/her, through a contract with a service/technology/content provider, will adhere to strict privacy regulations. A computer readable representation of this policy (which could be rendered for the content consumer) may be provided as part of the `describe_submit_data` operation, see clause 7.2. The policy regulation will effectively eliminate breaches of security of the collected private information in order to avoid any use of it that was not explicitly permitted by the end-consumer.

Providers of content wish to know how their content is performing as far as the consumer is concerned. Business decisions may be made as a result (cancel the scheduling of a programme because the viewers don't like it; advertisers will want to know viewers numbers, etc).

An anonymised subset of the UsageHistory table may be submitted if this functionality has been enabled by the consumer. Anonymity must be guaranteed so that no detail about the individual will be sent.

This will enable service providers and broadcasters the ability to analyze aggregated viewer preferences and allow them to make business decisions based on their performance in these areas.

5.2.2 Request Format

```
<element name="submit_Data" type="tns:submitDataType"/>
<complexType name="submitDataType">
  <sequence>
    <element name="UsageHistory" type="mpeg7:UsageHistoryType"/>
  </sequence>
</complexType>
```

The input to the `submit_Data` operation simply uses the mpeg7:`UsageHistoryType which is part of the UserDescription` defined in TS 102 822-3-1 [12], and therefore has the same semantics.

5.2.3 Response Format

```
<element name="submit_Data_Result" type="tns:submit_Data_Result"/>
<complexType name="submit_Data_Result">
  <attribute name="serviceVersion" type="unsignedInt" use="required"/>
</complexType>
```

The `submit_Data` response MUST contain some indication of the current version of the capability description. This allows receivers to update the capability description without having to download the capability description each time the `submit_Data` operation is used.

6 Transport Protocol

SOAP [9] and HTTP [5] are used for delivering *TV-Anytime* XML data over the IP networks, since this combination is very well suited to the point-to-point, request-response nature of the *TV-Anytime* operations. The exact usage of SOAP and HTTP is given in the next two clauses. Figure 4 provides a semantic representation of the network stack.

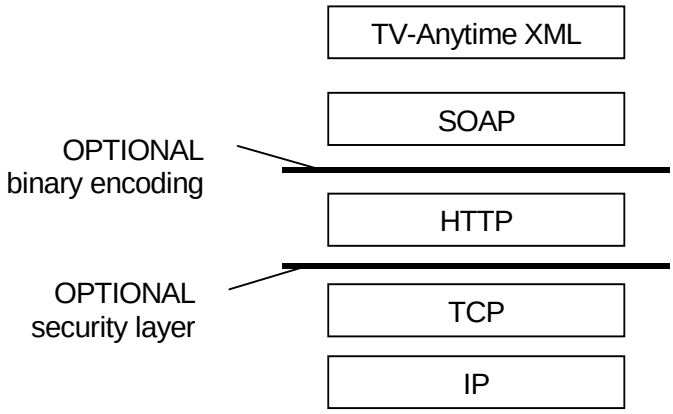


Figure 4: The bi-directional network transport stack

This architecture results in HTTP messages of the following form:

```
POST /tva/md-service HTTP/1.0
Host: www.example.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
Accept-Encoding: deflate
SOAPAction: "get_Data"

<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns=" http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <get_Data xmlns="http://www.TV-Anytime.org/2002/11/transport"
         xmlns:tvaf="http://www.TV-Anytime.org/2002/11/transport/fieldIDs">
      <QueryConstraints type="OR">
          <Predicate fieldID="tvaf:CRID"
                     fieldValue="crid://example.com/foo"/>
          <Predicate fieldID="tvaf:CRID"
                     fieldValue="crid://example.com/bar"/>
      </QueryConstraints>
      <RequestedTables>
        <Table type="ContentReferencingTable"/>
      </RequestedTables>
    </get_Data>
  </Body>
</Envelope>
```

Figure 5: Example HTTP request

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
Content-Encoding: deflate

<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://www.w3.org/2002/06/soap-envelope">
  <Body>
    <get_Data_Result xmlns=" http://schemas.xmlsoap.org/soap/envelope/">
      <ContentReferencingTable version="1"
           xmlns="http://www.TV-Anytime.org/2002/06/ContentReferencing">
        <!-- ... etc. -->
      </ContentReferencingTable>
    </get_Data_Result>
  </Body>
</Envelope>
```

Figure 6: Example HTTP response

6.1 SOAP

The following usage of SOAP is mandated:

- *TV-Anytime* metadata services will provide an HTTP binding, and may support other transport bindings where appropriate.
- SOAP supports different messaging styles, but is most commonly used for Remote Procedure Calls. *TV-Anytime* does not use remote procedure call messaging style, since this implies that every parameter must be included in a procedure call, which is not appropriate for *TV-Anytime* operations that have several optional parameter types. However, the remote procedure call convention of naming the root element in the SOAP body according to the name of the operation is followed.
- SOAP encoding is not used in the request or the response (i.e. the element in the root of the SOAP body will belong to the *TV-Anytime* transport types namespace, urn:tva:transport:2002). Servers will reject any request that arrives with a SOAP encoding attribute with the appropriate SOAP Fault.

- The SOAP Actor feature is not supported and servers will reject any request that arrives with a SOAP Actor attribute in the SOAP Header with the appropriate SOAP Fault.
- The following clause defines application specific fault conditions to be used in the SOAP Fault element.

This usage of SOAP is more formally defined using the WSDL interface definition that can be found in annex A.

6.2 Error Codes

The first line of error reporting is governed by the SOAP specification [9]. SOAP fault reporting and fault codes will be returned for most invalid requests or any request where the intent of the caller cannot be determined.

In a manner consistent with the SOAP processing rules, HTTP status codes will be used for communicating status information in HTTP. As is the case for SOAP, success reporting will use a 200-status code to indicate that the client's request including the SOAP component was successfully processed.

The `ErrorReport` element is defined to allow servers to report application-level errors that are specific to *TV-Anytime* metadata services. In accordance with the SOAP specification, if the content of the SOAP message's `Body` cannot be processed successfully, the SOAP fault MUST contain a `detail` element (which in turn contains an `ErrorReport`). Note that the inability to process a well-formed `Body` element is also termed an "application-level error", since the error cannot be detected by the SOAP processor and instead relies on application-level knowledge. The error report contains error information that includes descriptions and a code that can be used to determine the cause of the error. Errors that arise due to problems with in the HTTP layer or SOAP layer (e.g. the SOAP message does not conform to the SOAP specification) should be reported using the error mechanisms provided by those layers and MUST NOT be reported inside a SOAP fault's `Detail` element.

TV-Anytime application-level errors SHOULD be conveyed using standard HTTP status codes, where a 500-level code indicates a server-induced error. In such cases, the metadata service MUST issue an HTTP 500 "Internal Server Error" response and return an `ErrorReport` inside a SOAP fault report.

Any errors detected in the request will invalidate the entire request, and cause an `ErrorReport` to be generated within a SOAP fault as described below. A server MAY report multiple errors, although there is no requirement for the metadata service to continue processing the request after detecting the first error. In accordance with the SOAP specification, additional application response elements SHOULD NOT be included in the `Body` of the SOAP request. In other words, it is not possible for the server to indicate an error condition and also make a best-effort at providing a response.

The `ErrorReport` element takes the following form:

```
  <element name="ErrorReport" type="tns:ErrorReportType"/>
  <complexType name="ErrorReportType">
    <sequence>
      <element name="Error" type="tns:ErrorType" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <complexType name="ErrorType">
    <sequence>
      <element name="Reason" type="mpeg7:TextualType" minOccurs="0"
               maxOccurs="unbounded"/>
    </sequence>
    <attribute name="errorCode" use="required" type="tns:errorCodeType"/>
    <attribute name="fields" type="tns:fieldIDListType"/>
  </complexType>
  <simpleType name="errorCodeType">
    <restriction base="string">
      <enumeration value="FatalError"/>
      <enumeration value="InvalidRequest"/>
      <enumeration value="Unsupported"/>
      <enumeration value="UnrecognizedVersion"/>
      <enumeration value="UnspecifiedError"/>
      <enumeration value="UnsupportedQueryField"/>
      <enumeration value="UnsupportedSortField"/>
      <enumeration value="InvalidFieldID"/>
      <enumeration value="InvalidFieldValue"/>
```

```
    </restriction>
  </simpleType>
```

Name	Definition
ErrorReport	Lists the application level errors that have occurred as a result of invoking a metadata service.
Error	Describes a single application-level error.
errorCode	A REQUIRED string that precisely defines the nature of the error. The legitimate values for this string are listed in clauses 6.2.1.1 and 6.2.1.2.
fields	An OPTIONAL attribute that lists the fieldID values related to this error.
Reason	An OPTIONAL human meaningful description of the error.

6.2.1 General error conditions

The following error codes MAY be returned by invoking any of the operations defined in the present document. The field attribute is not relevant to these errors, and so SHALL not be present for the following error conditions.

- **FatalError:** Signifies that a serious technical error has occurred whilst processing the request.
- **InvalidRequest:** The query is well-formed but not valid according to the Schema defined by the present document.
- **Unsupported:** Signifies that the metadata service does not support an optional feature that is required in order to correctly process the request. A possible reason for this error is that the client has assumed a functionality that is at odds with the functionality described in the capability description.
- **UnrecognizedVersion:** Signifies that the namespace of the child element inside Body element of the request (e.g. the urn:tva:transport:2002 namespace) is unsupported by this metadata service.
- **UnspecifiedError:** Signifies any other error.

Error conditions are not mutually exclusive and some are special cases of others (e.g. some of the get_Data error conditions below are special cases of the Unsupported error). A metadata service SHOULD provide the most specific error code that is appropriate to the error.

The next clause defines error codes that are specific to the get_Data operation. No specialized error conditions are defined for the submit_Data, describe_get_Data, and describe_submit_Data operations.

6.2.2 get_Data operation error conditions

When any of the following errors are returned, the operation SHALL return a field attribute listing the field identifiers that caused the error to occur.

- **UnsupportedQueryField:** A query contains a fieldID value that is no supported by the metadata service. When this error is returned, the operation SHALL return a field attribute listing the unsupported fields.
- **UnsupportedSortField:** A sort is requested using a field for which sorting is not supported by the metadata service.
- **InvalidFieldID:** A fieldID in any type of predicate or sortCriteria element contained a field identifier that is neither in the *TV-Anytime* defined field identifier list, nor in the field identifier list given by the service capability description of this operation.
- **InvalidFieldValue:** The fieldValue in a BinaryPredicateBagType element contains a string that is not appropriate to the fieldID in that predicate. This might be because the field being queried is an enumerated type or because the field has syntactic restrictions (e.g. the tvaf:CRID field).

If a query requests a `ContentReferencingTable` and contains one or more `CRID` fields in the query, the metadata service should be careful to behave correctly when it is unable to provide location resolution data for the CRID(s). In particular, if the CRID is syntactically correct but no content referencing information is available a `Result` element should be returned for the CRID with the `status` flag set appropriately. No error condition should be raised. In general, it is not an error condition if a metadata service is simply unable to provide appropriate metadata in response to a query.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
Content-Encoding: deflate

<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://www.w3.org/2002/06/soap-envelope">
  <Body>
    <Fault>
      <faultcode>Client</faultcode>
      <detail>
        <ErrorReport xmlns="urn:tva:transport:2002"
          xmlns:tvaf="urn:tva:transport:fieldIDs:2002">
          <Error errorCode="UnsupportedQueryField" fields="tvaf:Title">
            <Reason xml:lang="en">
                   Searching on Title field is unsupported</Reason>
          </Error>
        </ErrorReport>
      </detail>
    </Fault>
  </Body>
</Envelope>
```

Figure 7: Example of response indicating an error

6.3 HTTP

- The HTTP server and client MUST fully support HTTP/1.0 [5].
- The client MUST always send the HTTP/1.1 defined Host header (which is defined in clause 14.23 of HTTP/1.1 [8]).
- The HTTP client and server negotiate a suitable compression using the Accept-Encoding header. Therefore, both the client and server MUST support the Accept-Encoding header (which is defined in clause D.2.3 of HTTP/1.0 [5], but is not required by that specification).
- Clients MUST be prepared to follow HTTP redirects (according to clause 9.3 of HTTP/1.0 [5]) to allow server fall-over and load-balancing. Note that contrary to HTTP/1.0, redirections should be followed automatically, without user intervention, even though the POST method is being used.
- The `SOAPAction` HTTP header field SHOULD be set to the name of the operation being called.

6.4 Encapsulation of Metadata

In all cases, the request and response data form a single, valid XML instance document. In general, these instance documents are self-describing and need no further encapsulation. However, the metadata encapsulation for the `get_Data` operation is defined in more detail.

6.4.1 Encapsulation of get_Data response

The content of the SOAP envelope is a `TVAMain` element and/or `ContentReferencingTable`, as specified in clause 5.1.2. These XML instance documents effectively provide an encapsulation format for the fragments they contain. The context path of each fragment is implicit from the fragment's location in the instance document. The `fragmentVersion` and `fragmentId` identifiers MAY appear at the fragment level (as defined in TS 102 822-3-1 [12]).

Each `get_Data` operation can be considered to provide a single metadata description (the term "metadata description" is defined in TS 102 822-3-2 [27]), from which each query selects a metadata subset. Thus, the content of the `TVAMain` fragment (see clause 4.3.1.1 of TS 102 822-3-2 [13]) is fixed at a given moment in time. The version attribute of the `TVAMain` fragment applies only to the fragment itself and not to its child fragments. Consequently, although this fragment is sent with every response, a client only needs to update its cache of the fragment (if any) when the `TVAMain` element's `version` attribute increases.

6.5 Encoding of Metadata

Servers MUST support UTF-8 textual encoding for all requests and responses, and MAY support other encodings, in which case the encoding MUST be indicated in the XML header according to clause 4.3.3 of the XML specification [1]. If an encoding is so indicated it MUST be consistent with the encoding indicated in the HTTP/1.0 [5] Content-Type header (e.g. see figure 5).

For efficiency reasons, HTTP clients and servers SHOULD support at least one well-known compression format. Clients SHOULD indicate the compression formats they understand using the Accept-Encoding HTTP header. The HTTP server SHOULD use the most efficient encoding that is understood by both the client and server, and indicate this in the HTTP response using the Content-Encoding header. It is RECOMMENDED that both *TV-Anytime* HTTP clients and servers support the use of the `deflate` Content-Encoding type (as defined in clause 3.5 of HTTP/1.1 [8]).

Other compression formats may be supported, including the use of BiM specified in Annex E. In all cases, the use of transport layer compression MUST be transparent to the SOAP processor and higher layers.

6.6 Metadata Service Security

The *TV-Anytime* phase 1 set of specifications offers clients guaranteed integrity of the metadata delivery from an authenticated server. This technology is described in TS 102 822-7 [16]. A metadata server that supports the TS 102 822-7 [16] specification for integrity checking SHALL be capable of communicating with clients that implement TS 102 822-7 [16] and with clients that do not. If TS 102 822-7 [16] is used and the server wishes to identify the source of the request, it SHOULD do so using HTTP Basic Authentication (negotiated according to HTTP/1.0 [5]). Any additional security services are not mandated or prevented by the present document and fall outside its scope.

7 Metadata Service Capability Descriptions

For each *TV-Anytime* `get_Data` or `submit_Data` operation that is provided, there is a corresponding operation, `describe_get_Data` or `describe_submit_Data`. These two operations (an operation, *X*, and its corresponding *describe_X* operation) together form a port (see annex A). Since a port always has a single binding and endpoint, the URL of the operation, *X*, and its corresponding *describe_X* operation must be the same. A describe operation is responsible for returning a capability description for the corresponding operation of the same port. A describe operation has no input parameters.

7.1 describe_get_Data

The `get_Data` capability description provides the following type of information about the operation.

- Human-readable descriptive information about the operation.
- The types of metadata tables available.
- If content referencing information is available, the Resolving Authority Records for that `get_Data` operation.
- If programme metadata is available, the CRID authorities known by the server for that type of metadata.
- If scheduling information is available, the content delivery services know by that server.
- If metadata queries are possible, the query fields that are allowed for each table.

- If sorting is possible, the sort fields that are supported for each table and the type of collation on which the sort is based.
- The ability of the operation to deliver update and invalidation information.

The format of the capability description is as follows.

```
<element name="describe_get_Data_Result"
         type="tns:describe_get_Data_Result"/>
<complexType name="describe_get_Data_Result">
  <sequence>
    <element name="Name" type="string" minOccurs="0"/>
    <element name="Description" type="mpeg7:TextualType" minOccurs="0"
             maxOccurs="unbounded"/>
    <element name="CollationURI" type="anyURI" minOccurs="0"/>
    <element name="ExtendedFieldList" type="tns:FieldIDDefinitionListType"
             minOccurs="0">
      <key name="UniqueExtendedFields">
        <selector xpath="tns:FieldIDDefinition"/>
        <field xpath="@fieldID"/>
      </key>
    </element>
    <element name="AuthorityList" type="tns:AuthorityListType"
             minOccurs="0"/>
    <element name="AvailableTables" type="tns:AvailableTableListType"/>
    <element name="UpdateCapability" type="tns:updateCapabilityType"
             minOccurs="0"/>
  </sequence>
  <attribute name="serviceVersion" type="unsignedInt" use="required"/>
</complexType>
```

Name	Definition
`serviceVersion`	A REQUIRED parameter that MUST equal the version number returned as part of the corresponding operation's result. The intention of this number is to make the client aware when an operation has been upgraded (e.g. can be searched on new channels or for new resolution authorities) and to refresh the cached capability description information (if any).
`Name`	OPTIONAL name of the metadata service, suitable for display to a user.
`Description`	OPTIONAL textual description of the metadata service, suitable for display to a user. This allows an application exploiting the metadata service to indicate to the user information about the metadata service that she is using (e.g. "Specialists in movie information and reviews").
`CollationURI`	OPTIONAL URI that indicates the name of the collation that sorts are based on. If no sorting is supported this field is meaningless and SHOULD be omitted. If sorting is supported and this field is not present then sorting MUST be based on the Default Unicode Collation Element Table. The present document does not define the form of the URI, or any means of specifying or obtaining a collation. Regional bodies may define suitable URIs and their associated collations, as appropriate to their locale.
`ExtendedFieldList`	Provides a list of `fieldID` values and their associated field using the `FieldIDDefinitionListType`. Used for fields that the server wishes to support but are not allocated a `fieldID` by the *TV-Anytime* Forum (see clause 5.1.1.1.1).
`AuthorityList`	An OPTIONAL list of authorities for which this operation can provide metadata for all the table types it is capable of delivering. See also clause 7.1.1.

Name	Definition
AvailableTables	A REQUIRED description of the types of data tables that can be returned by this get_Data response. At least one Table entry MUST be present. For each table, the fields that can be queried on and sorted on are specified. Depending on the type of table, other descriptive information is sometimes included.
UpdateCapability	An OPTIONAL update capability description to indicate the server's capability of update capability as defined in clause 5.1.2.4. When this element does not exist, the server does not support either versioned update nor the invalid fragment notification. See clause 7.1.4 for more details.

For examples of get_Data capability descriptions see annex D.

7.1.1 Use of the AuthorityList element

```
<complexType name="AuthorityListType">
  <sequence>
    <element name="Authority" type="string" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

Name	Definition
AuthorityListType	A definition of a list of one or more Authority elements.
Authority	Specifies a single authority for whose CRIDs the operation is able to provide *TV-Anytime* data.

The AuthorityList element can be instantiated in two locations:

- As one of the elements at the top level of the capability description. In this case, the server is able to provide data on the listed authorities for all of the data table types that are listed in the AvailableTables element. If the capability description indicates that a ContentReferencingTable can be returned, then there SHOULD be a ResolvingAuthorityRecord for each of the authorities listed in the top level AuthorityList. This element allows a server to specify its list of supported authorities, without the need to repeat this information for every table that it supports.

- As one of the children elements of a table listed in the AvailableTables element. In this case, the server is able to provide data of the type indicated by the Table elements xml schema declared type attribute for this authority.

NOTE: The capability description indicating that a certain type of metadata is available for a certain authority does not necessarily mean that the server will be able to provide that metadata for any particular CRID.

7.1.2 AvailableTables information

This element describes which table types the get_Data operation is capable of returning. Depending upon the table type, each entry in the list (a Table element) can contain a different child element. These child elements contain descriptive information that is relevant to that table type.

```
<complexType name="AvailableTableListType">
  <sequence>
    <element name="Table" type="tns:AvailableTableBase"
             maxOccurs="unbounded"/>
  </sequence>
</complexType>
<complexType name="AvailableTableBase" abstract="true">
  <sequence>
```

```
      <element name="AuthorityList" type="tns:AuthorityListType"
               minOccurs="0"/>
    </sequence>
    <attribute name="canSort" type="tns:fieldIDListType"/>
    <attribute name="canQuery" type="tns:fieldIDListType"/>
  </complexType>
  <complexType name="ContentReferencingTable">
    <complexContent>
      <extension base="tns:AvailableTableBase">
        <sequence>
          <element ref="rar:ResolvingAuthorityRecordTable"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
<complexType name="ClassificationSchemeTable">
    <complexContent>
      <extension base="tns:AvailableTableBase"/>
    </complexContent>
</complexType>
  <complexType name="ProgramInformationTable">
    <complexContent>
      <extension base="tns:AvailableTableBase"/>
    </complexContent>
  </complexType>
  <complexType name="GroupInformationTable">
    <complexContent>
      <extension base="tns:AvailableTableBase"/>
    </complexContent>
  </complexType>
  <complexType name="ProgramLocationTable">
    <complexContent>
      <extension base="tns:AvailableTableBase">
        <sequence>
          <element name="AvailableLocations">
            <complexType>
              <sequence>
                <element name="Availability" type="duration" minOccurs="0"/>
                <element name="ServiceURL" type="anyURI"
                         maxOccurs="unbounded"/>
              </sequence>
            </complexType>
          </element>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="ProgramReviewTable">
    <complexContent>
      <extension base="tns:AvailableTableBase"/>
    </complexContent>
  </complexType>
  <complexType name="SegmentInformationTable">
    <complexContent>
      <extension base="tns:AvailableTableBase"/>
    </complexContent>
  </complexType>
  <complexType name="ServiceInformationTable">
    <complexContent>
      <extension base="tns:AvailableTableBase"/>
    </complexContent>
  </complexType>
  <complexType name="CreditInformationTable">
    <complexContent>
      <extension base="tns:AvailableTableBase"/>
    </complexContent>
  </complexType>
```

Name	Definition
Table	Indicates that a particular table is available from this get_Data operation and encapsulates information relevant to querying that table. All instantiations of this element will be derived from the base type, AvailableTableBase, which can have the following three attributes.
xsi:type	This is the xml schema declared type attribute which is used to indicate the type of table where xsi is used as the namespace qualifier for xml schema. Can take the following values: ContentReferencingTable, ProgramInformationTable, GroupInformationTable, ProgramLocationTable, ProgramReviewTable, SegmentInformationTable, ServiceInformationTable, and CreditsInformationTable.
canQuery	List of fieldID values indicating the query fields that can be accepted by this operation. The order of the fields is not significant. All tables that can be queried on (see clause 5.1.1.1.4), except the ServiceInformationTable, MUST support querying using the CRID field. Hence, this attribute is meaningless when combined with a table of type "ContentReferencingTable" and "CreditsInformationTable". If this attribute is not present no other type of querying is supported for this table.
canSort	List of fieldID values indicating the query fields that can be used for sorting (as indicated using the SortCriteria parameter in a query). For every field that a server declares that it supports sorting, it SHALL support both ascending and descending sorting of that field. If this attribute is not present, sorting is not supported for this table. The order of the fields is not significant.
AuthorityList	An OPTIONAL list of authorities for which this operation can provide metadata of the type included in this table. See also clause 7.1.1.
ResolvingAuthorityRecordTable	As defined in TS 102 822-4 [14]. This element is REQUIRED if Table has the type "ContentReferencingTable". Tables of this type SHOULD NOT include an AuthorityList (since the ResolvingAuthorityRecordTable indicates which authorities content referencing information is provided for).
AvailableLocations	This element is REQUIRED if the Table has the type "ProgramLocationTable". The element specifies the types of programme location related queries that can be answered by the server.
Availability	An OPTIONAL duration that indicates to the client the time window over which the schedule information is available (e.g. "it is possible to query these channels for the next 10 days only").
ServiceURL	The content services for which scheduling information is available (corresponds to the ServiceURL element that can be included in a ServiceInformationTable).

7.1.2.1 Operations that can deliver content referencing information

If the server is capable of delivering content referencing information then the description will include a list of Resolving Authority Records describing the resolution services offered by this server. This allows a client to establish whether a particular CRID may be resolved using the metadata service.

7.1.2.2 Operations that can deliver programme metadata

If the server is capable of delivering programme metadata then the description will include a list of CRID authorities for which metadata is available.

If the server is capable of delivering scheduling information (`ProgramLocationTable`), a list of the content service URLs (e.g. broadcast channels, IP multicast services, or on-demand contents servers) known by this metadata service is also provided. Using this data a receiver can determine whether any of the programmes being described by this server are being offered using content delivery services that are actually available to the receiver. Based upon this information a receiver can construct queries for scheduling information, such as the one shown in example 7 in annex C.

7.1.3 Extended Field List

If a metadata service wishes to support querying or sorting on fields that are not allocated `fieldIDs` by the *TV-Anytime* Forum, it has to define these field using its own `ExtendedFieldList`. This allows a `get_Data` operation to provide searching and sorting functionality that is specialized to the metadata available to that metadata service. For example, if a metadata service specializes in providing segmentation information for the highlights of sports programmes, the metadata service can define extended fields for the Title and Synopsis of segments so as to allow receivers to make refined queries for segmentation information.

7.1.4 Description of update capabilities

There are two OPTIONAL features whose support shall be independently indicated using this part of the capability description.

```
<complexType name="updateCapabilityType">
  <attribute name="versionRequest" type="boolean" default="true"/>
  <attribute name="invalidResponse" type="boolean" default="true"/>
</complexType>
```

Name	Definition
`updateCapabilityType`	Defines a container of `versionRequest` and `invalidResponse` attributes, which can specify whether the server can provide versioned metadata and lists of invalid fragments.
`versionRequest`	An attribute of Boolean value to indicate the server's capability of handling version requests.
`invalidResponse`	An attribute of Boolean value to indicate whether the server is able to provide invalidated fragments when they are no longer valid or removed.

7.2 describe_submit_Data

The `describe_submit_Data` capability description provides a metadata service with the ability to describe the usage and preference information that it wishes to receive. It can optionally specify the privacy policy that will be used when user centric data is submitted to this metadata service.

```
<complexType name="describe_submit_Data_Result">
  <sequence>
    <element name="Name" type="string" minOccurs="0"/>
    <element name="Description" type="mpeg7:TextualType" minOccurs="0"
             maxOccurs="unbounded"/>
    <element name="RequestedTables" type="tns:RequestedSubmitTablesType"/>
    <element ref="p3p:POLICY" minOccurs="0"/>
  </sequence>
</complexType>
<complexType name="RequestedSubmitTablesType">
  <sequence>
    <element name="Table" maxOccurs="unbounded">
      <complexType>
```

```
          <attribute name="type" use="required">
            <simpleType>
              <restriction base="string">
                <enumeration value="UserPreferences"/>
                <enumeration value="UsageHistory"/>
              </restriction>
            </simpleType>
          </attribute>
        </complexType>
      </element>
    </sequence>
  </complexType>
```

Name	Definition
Name	OPTIONAL name of the metadata service, suitable for display to a user.
Description	OPTIONAL textual description of the metadata service, suitable for display to a user. This allows an application exploiting the metadata service to indicate to the user information about the metadata service that she is using.
RequestedTables	The list of tables that this operation wishes to receive.
p3p:POLICY	OPTIONAL element that describes the privacy policy of this operation, using the Platform for Privacy Preferences [23] specification.
RequestedSubmitTablesType	A list of table elements containing at least one entry.
Table	Specifies the type of table requested by the server.
type	REQUIRED parameter giving the type of *TV-Anytime* table requested by the server. Can take the values: "UsageHistory" or "UserPreferences".

Annex A (normative): Formal Definition of Metadata Services

This annex provides a WSDL [20] interface definition for all *TV-Anytime* metadata services. WSDL defines a number of terms used to describe web services. The way in which these relate to *TV-Anytime* metadata services is given below.

- **Operation:** All *TV-Anytime operations* are request-response based, so can be thought of as a type of remote procedure call. An example of a *TV-Anytime operation* is `submit_Data` or `describe_get_Data`.
- **PortType:** A collection of *operations*. When given a *binding* and a concrete endpoint the *portType* is known as a *port*. All *operations* in a given *port* MUST be present (i.e. it is not possible to offer only some of the *operations* in a *port*) and MUST have the same *binding*. The *TV-Anytime* Forum defines two *portTypes* (`get_Data` and `submit_Data`), each of which has two *operations*, the basic functionality (`get_Data` or `submit_Data`) and the corresponding describe *operation* (`describe_get_Data` or `describe_submit_Data`).
- **Binding:** A particular protocol binding (e.g. SOAP or HTTP GET) for a *portType*. There may be more than one *binding* for each *portType*. Each one is a different *port* and offers an alternative means for accessing the same *portType*. A *TV-Anytime* server could choose to provide other *bindings*, such as an HTTP POST implementation. By mandating a SOAP *binding*, a minimum level of interoperability is guaranteed.
- **Service:** A family of WSDL *ports* that are related in some way. A *TV-Anytime* metadata service can contain a get_Data port and/or a submit_Data port. In practice, most *TV-Anytime* servers will have just one WSDL *service*. However, the server provider could choose to group certain *ports* together for some reason. (E.g. a movie service containing a `get_Data` port and a `submit_Data` port, and a children service containing just a `get_Data` port).

The following document is a WSDL interface definition that defines the behaviour of all *TV-Anytime* defined web services. It plays two roles:

- The WSDL interface formally specifies the inputs, outputs, encodings and transport bindings used by all *TV-Anytime* web services. The definitions given correspond to the specification of *TV-Anytime* metadata services earlier in this document.
- Metadata service providers wishing to provide WSDL implementation definitions for their metadata services can import this WSDL interface definition. Annex G gives an example of how such a WSDL implementation can be referenced from a WS-Inspection description.

```
<?xml version="1.0"?>
<definitions
     targetNamespace="urn:tva:transport:wsdl:2002"
     xmlns:tns="urn:tva:transport:wsdl:2002"
     xmlns:tva="urn:tva:transport:2002"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns="http://schemas.xmlsoap.org/wsdl/">
  <documentation> WSDL Service Interface for a TV Anytime web service API. This WSDL
document defines the API calls for the two types of TV Anytime ports</documentation>
  <import namespace="urn:tva:transport:2002"/>
  <!-- Basic input and output messages. -->
  <message name="get_Data">
    <part name="body" element="tva:get_Data"/>
  </message>
  <message name="get_Data_Result">
    <part name="body" element="tva:get_Data_Result"/>
  </message>
  <message name="describe_get_Data">
    <part name="body" element="tva:describe_get_Data"/>
  </message>
  <message name="describe_get_Data_Result">
    <part name="body" element="tva:describe_get_Data_Result"/>
  </message>
  <message name="submit_Data">
```

```
    <part name="body" element="tva:submit_Data"/>
  </message>
  <message name="submit_Data_Result">
    <part name="body" element="tva:submit_Data_Result"/>
  </message>
  <message name="describe_submit_Data">
    <part name="body" element="tva:describe_submit_Data"/>
  </message>
  <message name="describe_submit_Data_Result">
    <part name="body" element="tva:describe_submit_Data_Result"/>
  </message>
  <message name="ErrorReportMessage">
    <part name="body" element="tva:ErrorReport"/>
  </message>
  <!-- The different types of services (ports) with their inputs and outputs. -->
  <portType name="get_Data_Port">
    <operation name="get_Data">
      <input message="tns:get_Data"/>
      <output message="tns:get_Data_Result"/>
      <fault name="error" message="tns:ErroReportMessage"/>
    </operation>
    <operation name="describe_get_Data">
      <input message="tns:describe_get_Data"/>
      <output message="tns:describe_get_Data_Result"/>
      <fault name="error" message="tns:ErroReportMessage"/>
    </operation>
  </portType>
  <portType name="submit_Data_Port">
    <operation name="submit_Data">
      <input message="tns:submit_Data"/>
      <output message="tns:submit_Data_Result"/>
      <fault name="error" message="tns:ErroReportMessage"/>
    </operation>
    <operation name="describe_submit_Data">
      <input message="tns:describe_submit_Data"/>
      <output message="tns:describe_submit_Data_Result"/>
      <fault name="error" message="tns:ErroReportMessage"/>
    </operation>
  </portType>
  <!-- The bindings: defines how SOAP/HTTP is used to carry the service. -->
  <binding name="get_Data_SOAP" type="tns:get_Data_Port">
    <documentation>TV Anytime get_Data binding</documentation>
    <soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="get_Data">
      <soap:operation soapAction="get_Data"/>
      <input>
        <soap:body use="literal" parts="body"/>
      </input>
      <output>
        <soap:body use="literal" parts="body"/>
      </output>
      <fault name="error">
        <soap:fault use="literal"/>
      </fault>
    </operation>
    <operation name="describe_get_Data">
      <soap:operation soapAction="describe_get_Data"/>
      <input>
        <soap:body use="literal" parts="body"/>
      </input>
      <output>
        <soap:body use="literal" parts="body"/>
      </output>
      <fault name="error">
        <soap:fault use="literal"/>
      </fault>
    </operation>
  </binding>
```

```
  <binding name="submit_Data_SOAP" type="tns:submit_Data_Port">
    <documentation>TV Anytime submit_Data binding</documentation>
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="submit_Data">
      <soap:operation soapAction="submit_Data"/>
      <input>
        <soap:body use="literal" parts="body"/>
      </input>
      <output>
        <soap:body use="literal" parts="body"/>
      </output>
      <fault name="error">
        <soap:fault use="literal"/>
      </fault>
    </operation>
    <operation name="describe_submit_Data">
      <soap:operation soapAction="describe_submit_Data"/>
      <input>
        <soap:body use="literal" parts="body"/>
      </input>
      <output>
        <soap:body use="literal" parts="body"/>
      </output>
      <fault name="error">
        <soap:fault use="literal"/>
      </fault>
    </operation>
  </binding>
</definitions>
```

Annex B (normative): TV Anytime defined field and contextNode identifiers

The *TV-Anytime* fieldID definition scheme which is defined in clause 5.1.1.1.2 is included in the tva_fieldID_definitions_v10.xml file.

The *TV-Anytime* contextNodeID definition scheme which is defined in clause 5.1.1.1.1 is included in the tva_contextNodeID_definitions_v10.xml file.

Annex C (informative): Examples of get_Data Requests

This annex gives some simple examples of how a `PredicateBag` element can be used to represent different types of queries. By combining these `PredicateBag` elements more refined queries can be built up.

C.1 Requesting data on specific CRIDs

To retrieve metadata on a particular set of CRIDs, the following query type is used. The type of data returned will depend upon the `RequestedTables` parameter.

```
    <PredicateBag type="OR">
      <BinaryPredicate fieldID="tvaf:CRID" fieldValue="crid://example.com/foo"/>
      <BinaryPredicate fieldID="tvaf:CRID" fieldValue="crid://example.com/bar"/>
    </PredicateBag>
```

C.2 Requesting specific fragments

To obtain a particular set of fragments, the query takes the following form.

```
    <PredicateBag type="OR">
      <BinaryPredicate fieldID="tvaf:fragmentID" fieldValue="34567"/>
      <BinaryPredicate fieldID="tvaf:fragmentID" fieldValue="12344"/>
    </PredicateBag>
```

The `RequestedTable` parameter can still be used to sort fragments retrieved in this fashion, but it has no effect on the fragments returned (i.e. it does not cause fragments from other tables to be returned also).

C.3 Searching for the film "Titanic"

The following search would return any programme with "Titanic" in the title.

```
    <BinaryPredicate fieldID="tvaf:Title" fieldValue="Titanic" test="contains"/>
```

To refine the query to find a specific "Titanic" programme the following query could be used. The following will only return matches that were directed by James Cameron.

```
  <QueryConstraints>
    <PredicateBag type="AND">
      <BinaryPredicate fieldID="tvaf:Title" fieldValue="Titanic"/>
      <PredicateBag type="AND" contextNode="tvac:CreditsItem">
        <BinaryPredicate fieldID="tvaf:Role" fieldValue=":role:V83"/>
        <BinaryPredicate fieldID="tvaf:GivenName" fieldValue="James"/>
        <BinaryPredicate fieldID="tvaf:FamilyName" fieldValue="Cameron"/>
      </PredicateBag>
    </PredicateBag>
  </QueryConstraints>
  <RequestedTables>
    <Table type="ProgramInformationTable">
      <SortCriteria fieldID="tvaf:CRID" order="ascending"/>
    </Table>
    <Table type="GroupInformationTable">
      <SortCriteria fieldID="tvaf:CRID" order="ascending"/>
```

```
    </Table>
  </RequestedTables>
```

C.4 Searching for a comedy drama that does not star Jim Carrey

The following search will find comedy dramas, but only those that do not involve Jim Carrey as a key character.

```
    <PredicateBag type="AND">
      <BinaryPredicate fieldID="tvaf:Genre" fieldValue=":content:3.4.11"/>
      <PredicateBag negate="true" type="AND" contextNode="tvac:CreditsItem">
        <BinaryPredicate fieldID="tvaf:Role" fieldValue=":role:V709"/>
        <BinaryPredicate fieldID="tvaf:GivenName" fieldValue="Jim"/>
        <BinaryPredicate fieldID="tvaf:FamilyName" fieldValue="Carey"/>
      </PredicateBag>
    </PredicateBag>
```

C.5 Searching for a programme with a rating of more than 8

The following predicate will find only programmes that have been assigned review ratings of more than 8 by the user's favorite reviewer, John Green.

```
    <PredicateBag type="AND" contextNode="tvac:Review">
      <BinaryPredicate fieldID="tva:ReviewerGivenName" fieldValue="John"/>
      <BinaryPredicate fieldID="tvaf:ReviewerFamilyName" fieldValue="Green"/>
      <BinaryPredicate fieldID="tvaf:RatingValue" test="greater_than_or_equals"
                  fieldValue="8"/>
    </PredicateBag>
```

C.6 Searching for a ClassificationScheme Table

The following predicate will find the ClassificationScheme Uri equals "urn:tva:metadata:cs:TVARoleCS:2002" or the CSAlias equals "TVARoleCS".

```
<get_Data xmlns=urn:tva:transport:2002 ...>
<QueryConstraints>
    <PredicateBag type="OR">
      <BinaryPredicate fieldID="CSUri"
             fieldValue="urn:tva:metadata:cs:TVARoleCS:2002" />
      <BinaryPredicate fieldID="CSAlias" fieldValue="TVARoleCS" />
    </PredicateBag>
  </QueryConstraints>
  <RequestedTables>
    <Table type="ClassificationScheme"/>
  </RequestedTables>
</get_Data>
```

The response including the ClassificationSchemeTable would be as below:

```
<get_Data_Result ...>
  <TVAMain version="20020928" publisher="TVA-Metadata-Service" ...>
    <ClassificationSchemeTable>
      <ClassificationScheme uri="urn:tva:metadata:cs:TVARoleCS:2002">
```

```
        <Import href="urn:mpeg:mpeg7:cs:RoleCS:2001"/>
        <mpeg7:Term termID="V708">
          <mpeg7:Name xml:lang="en">Dubber</mpeg7:Name>
        </mpeg7:Term>
        ...
      </ClassificationScheme>
    </ClassificationSchemeTable>
  </TVAMain>
</get_Data_Result>
```

C.7 Creating an EPG

The following search retrieves all the broadcast programmes on three specific channels over a two-day period.

```
    <PredicateBag type="AND">
      <BinaryPredicate fieldID="tvaf:PublishedTime"
           fieldValue="2002-09-26T00:00:00Z" test="greater_than_or_equals"/>
      <BinaryPredicate fieldID="tvaf:PublishedTime"
             fieldValue="2002-09-27T23:59:59Z" test="less_than_or_equals"/>
      <PredicateBag type="OR">
        <BinaryPredicate fieldID="tvaf:ServiceURL" fieldValue="dvb://1.2.1"/>
        <BinaryPredicate fieldID="tvaf:ServiceURL" fieldValue="dvb://1.2.2"/>
        <BinaryPredicate fieldID="tvaf:ServiceURL" fieldValue="dvb://1.2.3"/>
      </PredicateBag>
    </PredicateBag>
```

C.8 Searching for programmes with a review

This query searches for programmes being shown tonight that have a review. This is an example of when an "exists" test might be useful.

```
    <PredicateBag type="AND">
      <BinaryPredicate fieldID="tvaf:PublishedTime"
           fieldValue="2002-09-26T00:00:00Z" test="greater_than_or_equals"/>
      <BinaryPredicate fieldID="tvaf:PublishedTime"
           fieldValue="2002-09-27T23:59:59Z" test="less_than_or_equals"/>
      <UnaryPredicate fieldID="tvaf:Review" test="exists"/>
    </PredicateBag>
```

The query below requests programme reviews of programmes with "Titanic" in the title:

```
  <QueryConstraints>
    <BinaryPredicate fieldID="tvaf:Title" fieldValue="Titanic" test="contains"/>
  </QueryConstraints>
  <RequestedTables>
    <Table type="ProgramReviewTable"/>
  </RequestedTables>
```

If we now request for programme reviews of programmes with either "Titanic" or "Star Wars" in the title, it is not clear in the result which review corresponds to which part of the request.

```
  <QueryConstraints>
    <PredicateBag type="OR">
      <BinaryPredicate fieldID="tvaf:Title" fieldValue="Titanic" test="contains"/>
      <BinaryPredicate fieldID="tvaf:Title" fieldValue="Star Wars" test="contains"/>
    </PredicateBag>
  </QueryConstraints>
  <RequestedTables>
    <Table type="ProgramReviewTable"/>
```

```
  </RequestedTables>
```

A solution to remove the ambiguity is to not use the disjunctive condition in this case and to issue separate requests.

Another solution to remove the ambiguity is to add the following tables "ProgramInformationTable" and "GroupInformationTable" to the request or to issue a second request for "ProgramInformationTable" and "GroupInformationTable" using CRIDs returned within the first response.

C.9 Updating a fragment

The following query will check to see if a metadata service has a later version of a fragment than the version cached by the receiver.

```
    <PredicateBag type="AND">
      <Predicate fieldID="tvaf:fragmentID" fieldValue="34567"/>
      <Predicate fieldID="tvaf:fragmentVersion"
                 test="greater_than" fieldValue="20020925"/>
    </PredicateBag>
```

C.10 Requesting update fragments

Following is an example of a query to retrieve fragments which are updated after the given date, i.e. September 25, 2002, using the service described in clause C.5.

```
<BinaryPredicate fieldID="tvaf:fragmentVersion" test="greater_than"
                 fieldValue="20020925"/>
```

As a response to the previous example query, a metadata of following format can be received. In this response, it notifies the client system that the fragment with fragmentID "11" published before September, 30, 2002 and the fragment with fragmentID "15" published before September, 27, 2002 are no longer valid in addition to the updated fragments of programme information table and programme location table.

```
<get_Data_Result ...>
  <Invalid>
    <Fragment fragmentID="11" fragmentVersion="20020930"/>
    <Fragment fragmentID="15" fragmentVersion="20020927"/>
  </Invalid>
  <TVAMain version="20020928" publisher="TVA-Metadata-Service" ...>
    <ProgramDescription>
      <ProgramInformationTable>
        <ProgramInformation fragmentID="040" fragmentVersion="20021001">
           <!--..: etc. -->
        </ProgramInformation>
      </ProgramInformationTable>
      <ProgramLocationTable>
        <Schedule fragmentID="023" fragmentVersion="20020928">
          ...
        </Schedule>
      </ProgramLocationTable>
    </ProgramDescription>
  </TVAMain>
</get_Data_Result>
```

Annex D (informative): Examples of get_Data Operation's Capability Description

This annex gives some examples of the different types of functionality a `get_Data` operation might offer in real-life *TV-Anytime* metadata service deployments.

D.1 Pure location resolution service

A broadcaster provides a location resolution service over IP as an alternative to its unidirectional location resolution service. No other data is provided by the web service, which only offers a `get_Data` operation that is capable of returning a `ContentReferencingTable` and no other table types. Only CRIDs from the autnam.com authority can be resolved using this operation.

```
<describe_get_Data_Result serviceVersion="3"
       xmlns="urn:tva:transport:2002">
  <AvailableTables>
    <Table xsi:type="ContentReferencingTable">
      <ResolvingAuthorityRecordTable
            xmlns="urn:tva:ResolvingAuthority:2002">
        <ResolvingAuthorityRecord>
          <ResolutionProvider>autnam.com</ResolutionProvider>
          <AuthorityName>autnam.com</AuthorityName>
          <Class>primary</Class>
          <VersionNumber>1000</VersionNumber>
          <URL>http://www.autnam.com/lr/</URL>
          <FirstValidDate>2000-09-06T09:30:00Z</FirstValidDate>
          <LastValidDate>2000-09-28T18:00:00Z</LastValidDate>
          <Weighting>1</Weighting>
        </ResolvingAuthorityRecord>
      </ResolvingAuthorityRecordTable>
    </Table>
  </AvailableTables>
</describe_get_Data_Result>
```

D.2 Pure metadata retrieval service for broadcast enhancement

A broadcaster chooses to deliver enhanced metadata (critical reviews and segmentation data) using the bi-directional channel. All other data (including scheduling and content referencing information) is delivered by means of the unidirectional channel. The broadcaster offers a `get_Data` operation that is able to provide a `ProgramInformationTable`, `GroupInformationTable`, `ProgramReviewTable` and `SegmentInformationTable` in its responses. All of these tables can be queried using the `fragmentId` and `fragmentVersion` to allow integration with, and updating of, unidirectionally delivered metadata. This is indicated using the `UpdateCapability` element appropriately. In addition, the `ProgramInformationTable` and `GroupInformationTable` can be sorted according to Title and Genre fields. Queries based upon metadata constraints are not supported by this metadata service.

```
<describe_get_Data_Result serviceVersion="3"
            xmlns="urn:tva:transport:2002">
  <AuthorityList>
    <Authority>broadcaster.com</Authority>
  </AuthorityList>
  <AvailableTables
      xmlns:tvaf="urn:tva:transport:fieldIDs:2002">
    <Table xsi:type="ProgramInformationTable" canQuery="tvaf:fragmentID
           tvaf:fragmentVersion" canSort="tvaf:Title tvaf:Genre"/>
```

```
    <Table xsi:type="GroupInformationTable" canQuery="tvaf:fragmentID
           tvaf:fragmentVersion" canSort="tvaf:Title tvaf:Genre"/>
    <Table xsi:type="ProgramReviewTable" canQuery="tvaf:fragmentID
           tvaf:fragmentVersion"/>
    <Table xsi:type="SegmentInformationTable" canQuery="tvaf:fragmentID
           tvaf:fragmentVersion"/>
  </AvailableTables>
  <UpdateCapability versionRequest="true" invalidResponse="false"/>
</describe_get_Data_Result>
```

D.3 A rich metadata service that allows users to search for movies

The operation could be provided by a third party (e.g. IMDB.com) who creates its own CRIDs that can be resolved into creator CRIDs. This would require the web server to provide a `get_Data` operation that supports searches for movies based on Title, Keywords, Genre and Actors / Directors. The operation also allows searches on an extended field: colour. The operation is able to return a `ContentReferencingTable`, `ProgramInformationTable` and `ProgramReviewTable`.

```
<describe_get_Data_Result serviceVersion="3"
            xmlns="urn:tva:transport:2002">
  <Name>Barry Norman's Movie Recommendations</Name>
  <Description>For dedicated movie fans. Find the latest reviews and
ratings</Description>
  <ExtendedFieldList
         targetNamespace=http://www.barry-norman.com/tva-fields
         xmlns:tva="urn:tva:metadata:2002">
    <FieldIDDefinition fieldID="Color"
fieldDefinition="/tva:TVAMain/tva:ProgramDescription/tva:ProgramInformationTable/tva:Prog
ramInformation/tva:AVAttributes/tva:VideoAttributes/tva:Color/@type"/>
  </ExtendedFieldList>
  <AuthorityList>
    <Authority>barry-norman.com</Authority>
  </AuthorityList>
  <AvailableTables>
    <Table xsi:type="ContentReferencingTable">
      <ResolvingAuthorityRecordTable
         xmlns="urn:tva:ResolvingAuthority:2002">
        <ResolvingAuthorityRecord>
          <ResolutionProvider>barry-norman.com</ResolutionProvider>
          <AuthorityName>barry-norman.com</AuthorityName>
          <Class>primary</Class>
          <VersionNumber>1000</VersionNumber>
          <URL>http://www.barry-norman.com/tva</URL>
          <FirstValidDate>2000-09-06T09:30:00Z</FirstValidDate>
          <LastValidDate>2000-09-28T18:00:00Z</LastValidDate>
          <Weighting>1</Weighting>
        </ResolvingAuthorityRecord>
      </ResolvingAuthorityRecordTable>
    </Table>
    <Table xsi:type="ProgramInformationTable"
        xmlns:tvaf=urn:tva:transport:fieldIDs:2002
        xmlns:ef=http://www.barry-norman.com/tva-fields
        canQuery="tvaf:Title tvaf:Keyword tvaf:Genre tvaf:Role
                  tvaf:GivenName tvaf:FamilyName ef:Color"/>
    <Table xsi:type="ProgramReviewTable"/>
  </AvailableTables>
</describe_get_Data_Result>
```

D.4 Broadcaster provided metadata service used for constructing traditional EPGs

The metadata service allows querying for scheduling information, based upon the start time and content service of the programmes. The resulting `ProgramLocationTable` can be sorted according to the start time and channel of the programmes. Using the associated `ProgramInformationTable` a receiver can efficiently construct an EPG.

```
<describe_get_Data_Result serviceVersion="3"
            xmlns="urn:tva:transport:2002">
  <AuthorityList>
    <Authority>broadcaster.com</Authority>
  </AuthorityList>
  <AvailableTables
       xmlns:tvaf="urn:tva:transport:fieldIDs:2002">
    <Table xsi:type="ProgramInformationTable"/>
    <Table xsi:type="ProgramLocationTable" canQuery="tvaf:ServiceURL
          tvaf:PublishedTime" canSort="tvaf:ServiceURL tvaf:PublishedTime">
      <AvailableLocations>
        <Availability>P7D</Availability>
        <ServiceURL>dvb://2.7d1.13</ServiceURL>
        <ServiceURL>dvb://2.7d1.14</ServiceURL>
      </AvailableLocations>
    </Table>
  </AvailableTables>
</describe_get_Data_Result>
```

D.5 Pure metadata retrieval service for bi-directional channel

A third party or a broadcaster could deliver metadata using the bi-directional channel. The third party or the broadcaster offers a `get_Data` operation that is able to provide a `ProgramInformationTable`, `GroupInformationTable`, `ProgramReviewTable ProgramLocationTable,` and `SegmentInformationTable` in its responses. The metadata service allows querying for scheduling information, based upon the start time, end time. The resulting `ProgramLocationTable` can be sorted according to the start time and channel of the programmes. All of these tables can be queried using the `fragmentId` and `fragmentVersion` to allow updating of the previously delivered metadata. This is indicated using the `UpdateCapability` element appropriately.

```
<describe_get_Data_Result serviceVersion="3"
       xmlns="urn:tva:transport:2002">
  <AuthorityList>
    <Authority>broadcaster.com</Authority>
  </AuthorityList>
  <AvailableTables
     xmlns:tvaf="urn:tva:transport:fieldIDs:2002">
    <Table xsi:type="ProgramInformationTable"
        canQuery="tvaf:fragmentID tvaf:fragmentVersion"
        canSort="tvaf:Title tvaf:Genre"/>
    <Table xsi:type="GroupInformationTable"
        canQuery="tvaf:fragmentID tvaf:fragmentVersion"
        canSort="tvaf:Title tvaf:Genre"/>
    <Table xsi:type="ProgramReviewTable"
        canQuery="tvaf:fragmentID tvaf:fragmentVersion"/>
    <Table xsi:type="SegmentInformationTable"
        canQuery="tvaf:fragmentID tvaf:fragmentVersion"/>
    <Table xsi:type="ProgramLocationTable"
        canQuery=" tvaf:fragmentID tvaf:fragmentVersion tvaf:PublishedTime"
        canSort="tvaf:ServiceURL tvaf:PublishedTime">
      <AvailableLocations>
        <Availability>P7D</Availability>
```

```
        <ServiceURL>dvb://2.7d1.13</ServiceURL>
        <ServiceURL>dvb://2.7d1.14</ServiceURL>
      </AvailableLocations>
    </Table>
  </AvailableTables>
  <UpdateCapability versionRequest="true" invalidResponse="true"/>
</describe_get_Data_Result>
```

Annex E (informative): Use of BiM Encoded Metadata in Bi-directional Transport

This clause provides a framework for the OPTIONAL usage of BiM encoding of SOAP, within the context of *TV-Anytime* delivery of metadata over a bi-directional network. It ensures encodability of SOAP messages and interoperability between applications within that context.

E.1 Applying BiM encoding

In order to apply BiM encoding to a SOAP message, one needs to have an XML Schema describing it. Given the constraints expressed in the present document and the schema it contains, a schema describing a complete SOAP message is produced to provide the SOAP specific parts that are missing. Once such a schema is obtained, BiM encoding functionality can be added in a non-intrusive manner as it is solely concerned with encoding and Content Negotiation (described in clause 6.3).

Similarly, additional SOAP headers MAY be defined using locally added schema definitions, but will be discarded otherwise, taking into account limitations imposed on SOAP listed in clause 6.1. Future versions of the present document may define SOAP headers either directly or by reference, in which case they will be added to the schema describing the complete SOAP messages.

E.2 Negotiation of BiM Encoding

The HTTP 1.0 specification defines a mechanism allowing a user agent to reach an agreement with the server with which it is communicating based on the Accept-Encoding and Content-Encoding headers. This process is known as Content Negotiation. Clause 6.3 of the present document requires that clients and servers support the Accept-Encoding header, and that Content Negotiation take place between them so as to determine the best encoding.

In addition to this, clients and servers that choose to transfer data in a BiM encoded form SHALL flag BiM encoded content with a proper Content-Encoding header upon transmission, and SHALL NOT change the Content-Type corresponding to their content.

The content coding token corresponding to the BiM encoding is `x-bim`.

Annex F (informative): Bibliography

Documents are available from the *TV-Anytime* web site http://www.*TV-Anytime*.org.

"R-1: Call For Contributions" (TV014r3)

List of figures

Figure 1: The steps in using a *TV-Anytime* metadata service 11

Figure 2: Client requesting metadata from a metadata service 12

Figure 3: Client submitting user-centric data to a service provider 13

Figure 4: The bi-directional network transport stack 32

Figure 5: Example HTTP request 33

Figure 6: Example HTTP response 33

Figure 7: Example of response indicating an error 36

List of tables

Table 1: Description of special case `contextNodeID` values (informative) 19

Table 2: Description of special case `fieldID` values (informative) 20

Table 3: The meaning of the CRID field in the different *TV-Anytime* tables 23

History

Document history		
V1.1.1	October 2003	Publication