



PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 36/06

Nested Context Language 3.0
Part 9 – NCL Live Editing Commands

Luiz Fernando Gomes Soares
Rogério Ferreira Rodrigues
Romualdo Rezende Costa
Marcio Ferreira Moreno

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Nested Context Language 3.0

Part 9 – NCL Live Editing Commands

Luiz Fernando Gomes Soares
Rogério Ferreira Rodrigues
Romualdo Rezende Costa
Marcio Ferreira Moreno

Laboratório TeleMídia DI – PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro, RJ - 22451-900.

lfgs@inf.puc-rio.br, {rogério, Romualdo, Marcio}@telemidia.puc-rio.br

Abstract. *This technical report describes commands for live editing NCL 3.0 documents. NCL (Nested Context Language) is an XML application language based on the NCM (Nested Context Model) conceptual model for hypermedia document specification, with temporal and spatial synchronization among its media objects.*

Keywords: *live editing, digital TV; middleware; declarative environment; NCL.*

Resumo. *Este relatório técnico descreve os comandos para edição ao vivo de documentos NCL. NCL é uma aplicação XML baseada no modelo conceitual NCM (Nested Context Model) para a especificação de documentos hipermídia com sincronismo espacial e temporal entre seus objetos.*

Palavras chave: *edição ao vivo; TV digital; middleware; linguagem declarativa; NCL.*



Nested Context Language 3.0

Part 9 – NCL Live Editing Commands

© Laboratório TeleMídia da PUC-Rio – Todos os direitos reservados

Impresso no Brasil

As informações contidas neste documento são de propriedade do Laboratório TeleMídia (PUC-Rio), sendo proibida a sua divulgação, reprodução ou armazenamento em base de dados ou sistema de recuperação sem permissão prévia e por escrito do Laboratório TeleMídia (PUC-Rio). As informações estão sujeitas a alterações sem notificação prévia.

Os nomes de produtos, serviços ou tecnologias eventualmente mencionadas neste documento são marcas registradas dos respectivos detentores.

Figuras apresentadas, quando obtidas de outros documentos, são sempre referenciadas e são de propriedade dos respectivos autores ou editoras referenciados.

Fazer cópias de qualquer parte deste documento para qualquer finalidade, além do uso pessoal, constitui violação das leis internacionais de direitos autorais.

Laboratório TeleMídia

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente, 225, Prédio ITS - Gávea

22451-900 – Rio de Janeiro – RJ – Brasil

<http://www.telemidia.puc-rio.br>

Table of Contents

1. Introduction.....	5
2. NCL Historical Evolution.....	6
3. Overview of NCL Elements	9
4. NCL Editing Commands	12
5. Command Parameters XML Schemas.....	21
6. Final Remarks	34
References.....	35
Appendix A – DSM-CC Transport of Editing Commands using Stream-event Descriptors and Object Carousels	36
Appendix B –Transport of Editing Commands Using Specific Structures defined by Ginga-NCL	39
B.1 Transporting all data structures in a specific MPEG-2 section type.....	41
B.2 Transporting metadata structures as Editing Command parameter	43
B.3 Transporting metadata structures in MPEG-2 metadata sections	43

Nested Context Language 3.0

Part 9 – NCL Live Editing Commands

Luiz Fernando Gomes Soares
Rogério Ferreira Rodrigues
Romualdo Rezende Costa
Marcio Ferreira Moreno

Laboratório TeleMídia DI – PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro, RJ - 22451-900.

lfgs@inf.puc-rio.br, {rogério, Romualdo, Marcio}@telemidia.puc-rio.br

Abstract. *This technical report describes commands for live editing NCL 3.0 documents. NCL (Nested Context Language) is an XML application language based on the NCM (Nested Context Model) conceptual model for hypermedia document specification, with temporal and spatial synchronization among its media objects.*

1. Introduction

The core of an NCL presentation engine is composed of the NCL Formatter and the Private Base Manager [SoRo05].

The NCL Formatter is in charge of receiving an NCL document and controlling its presentation, trying to guarantee that the specified relationships among media objects are respected. The formatter deals with NCL documents that are collected inside a data structure known as *private base* [SoRo05]. NCL documents in a private base may be started, paused, resumed, stopped and may refer to each other.

The Private Base Manager is in charge of receiving NCL document editing commands and maintaining the active NCL documents (documents being presented).

Editing commands may come from various means. Through editing commands, NCL documents may be authored on-the-fly, that is, they may be authored at the same time they are in exhibition.

This report describes the NCL editing commands aiming at live authoring NCL documents. It is organized as follows. Section 2 gives an historical evolution of the NCL versions. Section 3 presents a brief overview of the NCL 3.0 elements. Section 4 introduces the NCL editing commands whose XML Schema is presented in Section 5. Section 6 presents the final remarks. Appendix A and B discuss the syntax and use of NCL editing commands in MPEG-2 TS elementary streams.

2. NCL Historical Evolution

The first version of NCL [Anto00, AMRS00] was specified through an XML DTD – Document Type Definition [XML98].

The second version of NCL, named NCL 2.0, was specified using XML Schema [SCHE01]. Following recent trends, from version 2.0 on, NCL has been specified in a modular way, allowing the combination of its modules in language profiles.

Besides the modular structure, NCL 2.0 introduced new facilities to the previous version 1.0, among others:

- definition of hypermedia connectors and connector bases;
- use of hypermedia connectors for link authoring;
- definition of ports and maps for composite nodes, satisfying the document compositionality property;
- definition of hypermedia composite-node templates, allowing the specification of constraints on documents;
- definition of composite-node template bases;
- use of composite-node templates for authoring composite nodes;
- refinement of document specifications with content alternatives, through the <switch> element, grouping a set of alternative nodes;
- refinement of document specifications with presentation alternatives, through the <descriptorSwitch> element, grouping a set of alternative descriptors;
- use of a new spatial layout model.

NCL 2.1 brought some refinements to the previous version: a module for defining cost functions associated with media object duration was introduced; a module aiming at describing the selection rules of <switch> and <descriptorSwitch> elements was defined; and refinements in some NCL modules were made, mainly in the XTemplate module.

NCL 2.2 made minor refinements in some NCL 2.1 modules, concerning their element definitions, and introduced a different approach in defining NCL modules and profiles.

NCL 2.3 introduced two new modules for supporting base and entity reuse, and refined the definition of some elements in order to support the new features.

NCL 2.4 reviewed and refined the reuse support introduced in version 2.3, and the specification of the switch and descriptor switch elements. This version also split the Timing module introduced by NCL 2.1, creating a new module to encapsulate issues related with time-scaling operations (elastic time computation using temporal cost functions) in hypermedia documents.

The NCL 3.0 edition revised some functionalities contained in NCL 2.4. NCL 3.0 is more specific regarding some attribute values. This new version introduced two new functionalities, as well: Key Navigation and Animation functionalities. In addition, NCL 3.0 made depth modifications on the Composite-Node Template functionality and introduces some SMIL based modules to NCL profiles for transition effects in media presentation and for metadata definition. NCL 3.0 also reviewed the hypermedia connector specification in order to have a more concise notation. Relationships among imperative and

declarative objects and other objects are also refined in NCL 3.0, as well as the behavior of imperative and declarative object players. Finally, NCL 3.0 also refined the support to multiple exhibition devices and introduced the support to NCL live editing commands.

NCM is the model underlying NCL. However, in its present version 3.0, NCL does not reflect all NCM 3.0 facilities yet. In order to understand NCL facilities in depth, it is necessary to understand the NCM concepts. With the aim of offering a scalable hypermedia model, with characteristics that may be progressively incorporated in hypermedia system implementations, the NCM and NCL family was divided in several parts.

The Nested Context Model is composed of Parts 1, 2, 3, and 4 of the collection:

- Part 1 – NCM Core
concerned with the main model entities, which should be present in all NCM implementations¹.
- Part 2 – NCM Virtual Entities
concerned mainly with the definition of virtual anchors, nodes and links.
- Part 3 – NCM Version Control
concerned with model entities and attributes to support versioning.
- Part 4 – NCM Cooperative Work
concerned with model entities and attributes to support cooperative document handling.

The NCL (Nested Context Language) specification is composed of Parts 5 to 12 of the collection:

- Part 5 – NCL (Nested Context Language) Full Profile
concerned with the definition of an XML application language for authoring and exchanging NCM-based documents, using all NCL modules, including those for the definition and use of templates, and also the definition of constraint connectors, composite-connectors, temporal cost functions, transition effects and metainformation characterization.
- Part 6 – NCL (Nested Context Language) XConnector Profile Family
concerned with the definition of an XML application language for authoring connector bases. One profile is defined for authoring causal connectors, another one for authoring causal and constraint connectors, and a third one for authoring both simple and composite connectors.
- Part 7 – Composite Node Templates
concerned with the definition of the NCL Composite-Node Template functionality, and with the definition of an XML application language (XTemplate) for authoring template bases.
- Part 8 – NCL (Nested Context Language) Digital TV Profiles
concerned with the definition of an XML application language for authoring documents

¹ It is also possible to have NCM implementations that ignore some of the basic entities, but this is not relevant so as to deserve a minimum-core definition.

aiming at the digital TV domain. Two profiles are defined: the Enhanced Digital TV (EDTV) profile and the Basic Digital TV (BDTV) profile.

- Part 9 – NCL Live Editing Commands (this document)
concerned with editing commands used for live authoring applications based on NCL.
- Part 10 – Imperative Objects in NCL: The NCLua Scripting Language
concerned with the definition of objects that contain imperative code and how these objects may be related with other objects in NCL applications.
- Part 11 – Declarative Hypermedia Objects in NCL: Nesting Objects with NCL Code in NCL Documents
concerned with the definition of hypermedia objects that contain declarative code (including nested objects with NCL code) and how these objects may be related with other objects in an NCL application.
- Part 12 – Support to Multiple Exhibition Devices
concerned with the use of multiple devices for simultaneously presenting an NCL document.

In order to understand NCL, the reading of Part 1: NCM Core is recommended.

3. Overview of NCL Elements

NCL is an XML application that follows the modularization approach. The modularization approach has been used in several W3C language recommendations. A *module* is a collection of semantically-related XML elements, attributes, and attribute's values that represents a unit of functionality. Modules are defined in coherent sets. A *language profile* is a combination of modules. Several NCL profiles have been defined, among them those defined by Parts 5, 6, 7, and 8 of the NCL collection presented in Section 2. Of special interest are the profiles defined for Digital TV, the EDTVProfile (*Enhanced Digital TV Profile*) and the BDTVProfile (*Basic Digital TV Profile*). This section briefly describes the elements that compose these profiles. The complete definition of the NCL 3.0 modules for these profiles, using XML Schemas, is presented in [SoRo06]. Any ambiguity found in this text can be clarified by consulting the XML Schemas.

The basic NCL structure module defines the root element, called `<ncl>`, and its children elements, the `<head>` element and the `<body>` element, following the terminology adopted by other W3C standards.

The `<head>` element may have `<importedDocumentBase>`, `<ruleBase>`, `<transitionBase>`, `<regionBase>`, `<descriptorBase>`, `<connectorBase>`, `<meta>`, and `<metadata>` elements as its children.

The `<body>` element may have `<port>`, `<property>`, `<media>`, `<context>`, `<switch>`, and `<link>` elements as its children. The `<body>` element is treated as an NCM context node. In NCM [SoRo05], the conceptual data model of NCL, a node may be a context, a switch or a media object. Context nodes may contain other NCM nodes and links. Switch nodes contain other NCM nodes. NCM nodes are represented by corresponding NCL elements.

The `<media>` element defines a media object specifying its type and its content location. NCL only defines how media objects are structured and related, in time and space. As a glue language, it does not restrict or prescribe the media-object content types. However, some types are defined by the language. For example: the “application/x-ncl-settings” type, specifying an object whose properties are global variables defined by the document author or are reserved environment variables that may be manipulated by the NCL document processing; and the “application/x-ncl-time” type, specifying a special `<media>` element whose content is the Greenwich Mean Time (GMT).

The `<context>` element is responsible for the definition of context nodes. An NCM context node is a particular type of NCM composite node and is defined as containing a set of nodes and a set of links [SoRo05]. Like the `<body>` element, a `<context>` element may have `<port>`, `<property>`, `<media>`, `<context>`, `<switch>`, and `<link>` elements as its children.

The `<switch>` element allows the definition of alternative document nodes (represented by `<media>`, `<context>`, and `<switch>` elements) to be chosen during presentation time. Test rules used in choosing the switch component to be presented are defined by `<rule>` or `<compositeRule>` elements that are grouped by the `<ruleBase>` element, defined as a child element of the `<head>` element.

The NCL Interfaces functionality allows the definition of node interfaces that are used in relationships with other node interfaces. The `<area>` element allows the definition of content anchors representing spatial portions, temporal portions, or temporal and spatial portions of a media object (`<media>` element) content. The `<port>` element specifies a composite node (`<context>`, `<body>` or `<switch>` element) port with its respective mapping to an interface of one of its child components. The `<property>` element is used for defining a node property or a group of node properties as one of the node's interfaces. The `<switchPort>` element allows the creation of `<switch>` element interfaces that are mapped to a set of alternative interfaces of the switch's internal nodes.

The `<descriptor>` element specifies temporal and spatial information needed to present each document component. The element may refer a `<region>` element to define the initial position of the `<media>` element (that is associated with the `<descriptor>` element) presentation in some output device. The definition of `<descriptor>` elements shall be included in the document head, inside the `<descriptorBase>` element, which specifies the set of descriptors of a document. Also inside the document `<head>` element, the `<regionBase>` element defines a set of `<region>` elements, each of which may contain another set of nested `<region>` elements, and so on, recursively; regions define device areas (e.g. screen windows) and are referred by `<descriptor>` elements, as previously mentioned.

A `<causalConnector>` element represents a relation that may be used for creating `<link>` elements in documents. In a causal relation, a condition shall be satisfied in order to trigger an action. A `<link>` element binds (through its `<bind>` elements) a node interface with connector roles, defining a spatio-temporal relationship among objects (represented by `<media>`, `<context>`, `<body>` or `<switch>` elements).

The `<descriptorSwitch>` element contains a set of alternative descriptors to be associated with an object. Analogous to the `<switch>` element, the `<descriptorSwitch>` choice is done during the document presentation, using test rules defined by `<rule>` or `<compositeRule>` elements.

In order to allow an entity base to incorporate another already-defined base, the `<importBase>` element may be used. Additionally, an NCL document may be imported through the `<importNCL>` element. The `<importedDocumentBase>` element specifies a set of imported NCL documents, and shall also be defined as a child element of the `<head>` element.

Some important NCL element's attributes are defined in other NCL modules. The EntityReuse module allows an NCL element to be reused. This module defines the *refer* attribute, which refers to an element URI that will be reused. Only `<media>`, `<context>`, `<body>` and `<switch>` may be reused. The KeyNavigation module provides the extensions necessary to describe focus movement operations using a control device like a remote control. Basically, the module defines attributes that may be incorporated by `<descriptor>` elements. The Animation module provides the extensions necessary to describe what happens when a property value is changed. The change may be instantaneous, but it may also be carried out during an explicitly declared duration, either linearly or step by step. Basically, the Animation module defines attributes that may be incorporated by actions, defined as child elements of `<causalConnector>` elements.

Some SMIL functionalities are also incorporated by NCL. The <transition> element and some transition attributes have the same semantics of homonym element and attributes defined in the SMIL BasicTransitions module and the SMIL TransitionModifiers module. The NCL <transitionBase> element specifies a set of transition effects, defined by <transition> elements, and shall be defined as a child element of the <head> element.

Finally, the MetaInformation module is also incorporated, inheriting the same semantics of the SMIL MetaInformation module. Meta-information does not contain content information that is used or display during a presentation. Instead, it contains information about content that is used or displayed. The Metainformation module contains two elements that allow describing NCL documents. The <meta> element specifies a single property/value pair. The <metadata> element contains information that is also related to meta-information of the document. It acts as the root element of an RDF tree: RDF element and its sub-elements (for more details, refer to W3C metadata recommendations [RDF99]).

4. NCL Editing Commands

The Private Base Manager [SoRo05] is in charge of receiving NCL document editing commands and maintaining the active NCL documents (documents being presented).

Editing commands may come from various means. For example, in a Digital TV environment it is usual to adopt the DSM-CC (Digital Storage Media Command and Control) for carrying editing commands, in MPEG-2 TS elementary streams, coming from datacast providers. It is also possible in a Digital TV environment to receive editing commands via the interactive (return) channel, or even directly from the viewer at a TV receiver. Appendix A and B discuss the syntax and use of NCL editing commands in MPEG-2 TS elementary streams.

Editing commands are wrapped in a structure called *event descriptors*. Event descriptors have an identifier (*eventId*), a time reference (*eventNPT*) and a private data field. The identifier uniquely identifies each editing command event. The time reference indicates the exact moment to trigger the command. A time reference equal to zero informs that the command shall be triggered immediately after being received. The private data field provides support for command parameters, as shown in Figure 4.1.

Syntax	Number of bits
EventDescriptor () {	
eventId	16
eventNPT	33
privateDataLength	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 to 2008
FCS	8
}	

Figure 4.1 - Editing command event descriptor

The *commandTag* uniquely identifies the type of the editing command, as specified in Table 4.1. In order to allow sending a complete command in more than one event descriptor, all event descriptors of the same command shall be numbered and sent in sequence (that is, it cannot be multiplexed with other editing commands with the same *commandTag*), with the *finalFlag* equal to 1, except for the last descriptor that shall have the *finalFlag* field equal to 0. The *privateDataPayload* contains the editing-command parameters. Finally, the *FCS* field contains a checksum of the entire *privateData* field, including the *privateDataLength*.

NCL editing commands are divided in three subsets.

The first subset focuses on the private base activation and deactivation (openBase, activateBase, deactivateBase, saveBase, and closeBase commands).

NCL documents may be added to a private base and then may be started, paused, resumed, stopped, saved and removed, through well-defined commands that compose the second subset.

The third subset defines commands for live editing, allowing NCL elements to be added and removed, and allowing values to be set to NCL <property> elements. *Add* commands always have NCL elements as their arguments. Whether the specified element already exists or not, document consistency shall be maintained by the NCL formatter, in the sense that all element attributes stated as required shall be defined. The elements are defined using an XML-based syntax notation defined in Section 5, with the exception of the *addInterface* command: the *begin* or *first* attribute of an <area> element may receive the “now” value, specifying the current NPT (Normal Play Time) of the node specified in the *nodeId* argument.

If the XML-based *command parameter* is short enough it may be transported directly in the event descriptors’ payload. Otherwise, the *privateDataPayload* carries a set of reference pairs. In the case of pushed files (NCL documents or nodes), each pair is used to associate a set of file paths with their respective location (identification) in the transport system (see examples in Appendix A and B). In the case of pulled files or files sited in the receiver itself, no reference pairs have to be sent, except the {uri, “null”} pair associated with the NCL document or XML node specification that is commanded to be added.

Table 4.1 shows the *command strings* with their arguments surrounded by round brackets. The table also gives the unique identifier of each editing command (*commandTag*) and the command semantics.

Table 4.1 – Editing commands for NCL Private Base Manager

Command String	Command Tag	Description
openBase (baseId, location)	0x00	Opens an existing private base located with the <i>location</i> parameter. If the private base does not exist or the <i>location</i> parameter is not informed, a new base is created with the <i>baseId</i> identifier. The <i>location</i> parameter shall specify the device and the path for opening the base
activateBase (baseId)	0x01	Turns on an opened private base.
deactivateBase (baseId)	0x02	Turns off an opened private base.
saveBase (baseId, location)	0x03	Saves all private base content into a persistent storage device (if available). The <i>location</i> parameter shall specify the device and the path for saving the base.
closeBase (baseId)	0x04	Closes the private base and disposes all private base content.
addDocument (baseId, {uri,	0x05	Adds an NCL document to a private base. The

Command String	Command Tag	Description
id}+)		<p>NCL document's files can be:</p> <p>i) sent in the datacast network as a set of pushed files; for these pushed files, each {uri,id} pair is used to relate a set of file paths in the NCL document specification with their respective locations in a transport system (see examples in Appendix A and B);</p> <p>NOTE: The set of reference pairs shall be sufficient for the middleware to map any file reference present in the NCL document specification to its concrete location in the receiver memory.</p> <p>ii) received from a network as a set of pulled files, or may be files already present in the receiver; for these pulled files, no {uri, id} pairs have to be sent, except the {uri, "null"} pair associated with the NCL document specification that the editing command request to be added in <i>baseId</i>, if this NCL document is not received as a pushed file.</p>
removeDocument (baseId, documentId)	0x06	Removes an NCL document from a private base.
startDocument (baseId, documentId, interfaceId, offset, refDocumentId, refNodeId) NOTE The offset parameter is a time value.	0x07	<p>Starts playing an NCL document in a private base, beginning the presentation from a specific document interface. The time reference provided in the <i>eventNPT</i> field defines the initial time positioning of the document with regards to the NPT time base value of the <i>refNodeId</i> content of the <i>refDocumentId</i> document being received.</p> <p>Three cases may happen:</p> <p>i) If <i>eventNPT</i> is greater than or equal to the current NPT time base value of the <i>refNodeId</i> content being received, the document presentation shall wait until NPT has the value specified in <i>eventNPT</i> to be started from its beginning time+<i>offset</i>.</p> <p>ii) If <i>eventNPT</i> is less than the NPT time base value of the <i>refNodeId</i> content being received, the document shall be started immediately from its beginning time+<i>offset</i>+(NPT – <i>eventNPT</i>)_{seconds}</p> <p>NOTE: Only in this case, the <i>offset</i> parameter value may be a negative time value, but <i>offset</i>+(NPT – <i>eventNPT</i>)_{seconds} shall be a positive time value.</p> <p>iii) If <i>eventNPT</i> is equal to 0, the document shall start its presentation immediately from its beginning time+<i>offset</i>.</p>
stopDocument (baseId, documentId)	0x08	Stops the presentation of an NCL document in a private base. All document events that are

Command String	Command Tag	Description
		occurring shall be stopped.
pauseDocument (baseId, documentId)	0x09	Pauses the presentation of an NCL document in a private base. All document events that are occurring shall be paused.
resumeDocument (baseId, documentId)	0x0A	Resumes the presentation of an NCL document in a private base. All document events that were previously paused by the <i>pauseDocument</i> editing command shall be resumed.
saveDocument (baseId, documentId, location)	0x2E	Saves an NCL document into a persistent storage device (if available). The <i>location</i> parameter shall specify the device and the path for saving the document. If the NCL document to be saved is running in the private base, first stops its presentation (all document events that are occurring shall be stopped).
addRegion (baseId, documentId, regionBaseId, regionId, xmlRegion)	0x0B	Adds a <region> element as a child of another <region> in the <regionBase> or as a child of the <regionBase> (regionId="null") of an NCL document in a private base.
removeRegion (baseId, documentId, regionId)	0x0C	Removes a <region> element from a <regionBase> of an NCL document in a private base.
addRegionBase (baseId, documentId, xmlRegionBase)	0x0D	Adds a <regionBase> element to the <head> element of an NCL document in a private base. If the XML specification of the regionBase is sent in the transport system as a file system, the <i>xmlRegionBase</i> parameter is just a reference to this content in its transport system
removeRegionBase (baseId, documentId, regionBaseId)	0x0E	Removes a <regionBase> element from the <head> element of an NCL document in a private base.
addRule (baseId, documentId, xmlRule)	0x0F	Adds a <rule> element to the <ruleBase> of an NCL document in a private base.
removeRule (baseId, documentId, ruleId)	0x10	Removes a <rule> element from the <ruleBase> of an NCL document in a private base.
addRuleBase (baseId, documentId, xmlRuleBase)	0x11	Adds a <ruleBase> element to the <head> element of an NCL document in a private base. If the XML specification of the ruleBase is sent in the transport system as a file system, the <i>xmlRuleBase</i> parameter is just a reference to this content in its

Command String	Command Tag	Description
		transport system
removeRuleBase (baseId, documentId, ruleBaseId)	0x12	Removes a <ruleBase> element from the <head> element of an NCL document in a private base
addConnector (baseId, documentId, xmlConnector)	0x13	Adds a <connector> element to the <connectorBase> of an NCL document in a private base.
removeConnector (baseId, documentId, connectorId)	0x14	Removes a <connector> element from the <connectorBase> of an NCL document in a private base.
addConnectorBase (baseId, documentId, xmlConnectorBase)	0x15	Adds a <connectorBase> element to the <head> element of an NCL document in a private base. . If the XML specification of the connectorBase is sent in the transport system as a file system, the <i>xmlConnectorBase</i> parameter is just a reference to this content in its transport system
removeConnectorBase (baseId, documentId, connectorBaseId)	0x16	Removes a <connectorBase> element from the <head> element of an NCL document in a private base.
addDescriptor (baseId, documentId, xmlDescriptor)	0x17	Adds a <descriptor> element to the <descriptorBase> of an NCL document in a private base.
removeDescriptor (baseId, documentId, descriptorId)	0x18	Removes a <descriptor> element from the <descriptorBase> of an NCL document in a private base.
addDescriptorSwitch (baseId, documentId, xmlDescriptorSwitch)	0x19	Adds a <descriptorSwitch> element to the <descriptorBase> of an NCL document in a private base. . If the XML specification of the descriptorSwitch is sent in a transport system as a file system, the <i>xmlDescriptorSwitch</i> parameter is just a reference to this content
removeDescriptorSwitch (baseId, documentId, descriptorSwitchId)	0x1A	Removes a <descriptorSwitch> element from the <descriptorBase> of an NCL document in a private base.
addDescriptorBase (baseId, documentId, xmlDescriptorBase)	0x1B	Adds a <descriptorBase> element to the <head> element of an NCL document in a private base. If the XML specification of the descriptorBase is sent in a transport system as a file system, the <i>xmlDescriptorBase</i> parameter is just a reference to this content

Command String	Command Tag	Description
removeDescriptorBase (baseId, documentId, descriptorBaseId)	0x1C	Removes a <descriptorBase> element from the <head> element of an NCL document in a private base.
addTransition (baseId, documentId, xmlTransition)	0x1D	Adds a <transition> element to the <transitionBase> of an NCL document in a private base.
removeTransition (baseId, documentId, transitionId)	0x1E	Removes a <transition> element from the <transitionBase> of an NCL document in a private base.
addTransitionBase (baseId, documentId, xmlTransitionBase)	0x1F	Adds a <transitionBase> element to the <head> element of an NCL document in a private base. If the XML specification of the transitionBase is sent in a transport system as a file system, the <i>xmlTransitionBase</i> parameter is just a reference to this content
removeTransitionBase (baseId, documentId, transitionBaseId)	0x20	Removes a <transitionBase> element from the <head> element of an NCL document in a private base.
addImportBase (baseId, documentId, docBaseId, xmlImportBase)	0x21	Adds an <importBase> element to the base (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase>, or <connectorBase> element identified by <i>docBaseId</i>) of an NCL document in a private base.
removeImportBase (baseId, documentId, docBaseId, documentURI)	0x22	Removes an <importBase> element, whose <i>documentURI</i> attribute is identified by the documentURI parameter, from the base (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase>, or <connectorBase> element identified by <i>docBaseId</i>) of an NCL document in a private base.
addImportedDocumentBase (baseId, documentId, xmlImportedDocumentBase)	0x23	Adds an <importedDocumentBase> element to the <head> element of an NCL document in a private base.
removeImportedDocumentBase (baseId, documentId, importedDocumentBaseId)	0x24	Removes an <importedDocumentBase> element from the <head> element of an NCL document in a private base.
addImportNCL (baseId, documentId, xmlImportNCL)	0x25	Adds a <importNCL> element to the <importedDocumentBase> element of an NCL document in a private base.
removeImportNCL (baseId,	0x26	Removes an <importNCL> element, whose

Command String	Command Tag	Description
documentId, documentURI)		<i>documentURI</i> attribute is identified by the documentURI parameter, from the <importedDocumentBase> element of an NCL document in a private base
addNode (baseId, documentId, compositeId, {uri, id}+)	0x27	<p>Adds a node (<media>, <context>, or <switch> element) to a composite node (<body>, <context>, or <switch> element) of an NCL document in a private base. The XML specification of the node and its media content may be:</p> <p>i) sent in the datacast network as a set of pushed files; each {uri,id} pair is used to relate a set of file paths in the XML document specification of the node (see examples in Appendix A and B) with their respective location in a transport system</p> <p>NOTE: The set of reference pairs shall be sufficient for the middleware to map any file reference present in the node specification to its concrete location in the receiver memory.</p> <p>ii) received from a network as a set of pulled files, or may be files already present in the receiver; for these pulled files, no {uri, id} pairs have to be sent, except the {uri, “null”} pair associated with the XML node specification that the editing command request to be added in compositeId, if this XML document is not received as a pushed file.</p>
removeNode(baseId, documentId, compositeId, nodeId)	0x28	Removes a node (<media>, <context>, or <switch> element) from a composite node (<body>, <context>, or <switch> element) of an NCL document in a private base.
addInterface (baseId, documentId, nodeId, xmlInterface)	0x29	Adds an interface (<port>, <area>, <property>, or <switchPort>) to a node (<media>, <body>, <context>, or <switch> element) of an NCL document in a private base.
removeInterface (baseId, documentId, nodeId, interfaceId)	0x2A	Removes an interface (<port>, <area>, <property>, or <switchPort>) from a node (<media>, <body>, <context>, or <switch> element) of an NCL document in a private base. The interfaceID shall identify a <property> element’s <i>name</i> attribute or a <port>, <area>, or <switchPort> element’s <i>id</i> attribute.
addLink (baseId, documentId, compositeId, xmlLink)	0x2B	Adds a <link> element to a composite node (<body>, <context>, or <switch> element) of an NCL document in a private base.

Command String	Command Tag	Description
removeLink (baseId, documentId, compositeId, linkId)	0x2C	Removes a <link> element from a composite node (<body>, <context>, or <switch> element) of an NCL document in a private base.
setPropertyValue(baseId, documentId, nodeId, propertyId, value)	0x2D	Sets the value for a property. The <i>propertyId</i> parameter shall identify a <property> element's <i>name</i> attribute or a <switchPort> element's <i>id</i> attribute. The <property> or <switchPort> shall belong to a node (<body>, <context>, <switch> or <media> element) of an NCL document in a private base identified by the parameters.

The identifiers used in the commands are defined in 4.2

Table 4.2 – Identifiers used in editing commands

Identifiers	Definition
baseId	The <i>id</i> attribute of a private base. Usually, in DTV environment a private base is associated with a TV channel. Thus broadcast channel identifiers are used as the <i>baseId</i> values.
documentId	The <i>id</i> attribute of an <ncl> element of an NCL document.
refDocumentId	The <i>id</i> attribute of an <ncl> element of an NCL document
refNodeId	The <i>id</i> attribute of a <media> element of an NCL document
regionId	The <i>id</i> attribute of a <region> element of an NCL document.
ruleId	The <i>id</i> attribute of a <rule> element of an NCL document.
connectorId	The <i>id</i> attribute of a <connector> element of an NCL document.
descriptorId	The <i>id</i> attribute of a <descriptor> element of an NCL document.
descriptorSwitchId	The <i>id</i> attribute of a <descriptorSwitch> element of an NCL document.
transitionId	The <i>id</i> attribute of a <transition> element of an NCL

Identifiers	Definition
	document.
regionBaseId	The <i>id</i> attribute of a <regionBase> element of an NCL document.
ruleBaseId	The <i>id</i> attribute of a <ruleBase> element of an NCL document.
connectorBaseId	The <i>id</i> attribute of a <connectorBase> element of an NCL document.
descriptorBaseId	The <i>id</i> attribute of a <descriptorBase> element of an NCL document.
transitionBaseId	The <i>id</i> attribute of a <transitionBase> element of an NCL document.
docBaseId	The <i>id</i> attribute of a <regionBase>, <ruleBase>, <connectorBase>, <descriptorBase>, or <transitionBase> element of an NCL document.
documentURI	The <i>documentURI</i> attribute of an <importBase> element or an <importNCL> element of an NCL document.
importedDocumentBaseId	The <i>id</i> attribute of a <importedDocumentBase> element of an NCL document.
compositeID	The <i>id</i> attribute of a <body>, <context> or <switch> element of an NCL document.
nodeId	The <i>id</i> attribute of a <body>, <context>, <switch> or <media> element of an NCL document.
interfaceId	The <i>id</i> attribute of a <port>, <area>, <property> or <switchPort> element of an NCL document.
linkId	The <i>id</i> attribute of a <link> element of an NCL document.
propertyId	The <i>id</i> attribute of a <property> or <switchPort> element of an NCL document.

5. Command Parameters XML Schemas

NCL entities used in editing commands shall be a document in conformance with the NCL 3.0 Command profile defined by the XML Schema that follows.

Note that different from NCL documents, several <ncl> elements may be the root element in the XML command parameters.

NCL30EdCommand.xsd

```
<!--
XML Schema for the NCL Language

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/profiles/NCL30EdCommand.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:animation="http://www.ncl.org.br/NCL3.0/Animation"
  xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/
CompositeNodeInterface"
  xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
  xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/
ConnectorCausalExpression"
  xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
  xmlns:context="http://www.ncl.org.br/NCL3.0/Context"
  xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
  xmlns:entityReuse="http://www.ncl.org.br/NCL3.0/EntityReuse"
  xmlns:extendedEntityReuse="http://www.ncl.org.br/NCL3.0/
ExtendedEntityReuse"
  xmlns:descriptorControl="http://www.ncl.org.br/NCL3.0/
DescriptorControl"
  xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
  xmlns:keyNavigation="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
  xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
  xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
  xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  xmlns:propertyAnchor="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
  xmlns:testRuleUse="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  xmlns:timing="http://www.ncl.org.br/NCL3.0/Timing"
  xmlns:transitionBase="http://www.ncl.org.br/NCL3.0/TransitionBase"
  xmlns:metainformation="http://www.ncl.org.br/NCL3.0/Metainformation"
  xmlns:transition="http://www.ncl.org.br/NCL3.0/Transition"
  xmlns:profile="http://www.ncl.org.br/NCL3.0/EdCommandProfile"
  targetNamespace="http://www.ncl.org.br/NCL3.0/EdCommandProfile"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >
```

```

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/Animation"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Animation.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CompositeNodeInterface.xsd"/>
<import
namespace="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnectorFunctionality.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorBase.xsd"/>
<import
namespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorCausalExpression.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ContentControl"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ContentControl.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Context"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Context.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Descriptor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Descriptor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/DescriptorControl"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30DescriptorControl.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/EntityReuse"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30EntityReuse.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ExtendedEntityReuse.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Import"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Import.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30KeyNavigation.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Layout"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Layout.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Linking"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Linking.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Media"

schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Media.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30MediaContentAnchor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30PropertyAnchor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Structure"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Structure.xsd"/>

```

```

    <import namespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30SwitchInterface.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/TestRule"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TestRule.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TestRuleUse.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/Timing"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Timing.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/TransitionBase"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TransitionBase.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/Metainformation"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Metainformation.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/Transition"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Transition.xsd"/>

    <!-- ===== -->
    <!--EditingCommand -->
    <!-- ===== -->
    <!--defines the command element -->

    <!--This is a pseudo-element, only defined to show the elements that
may be used in the root of the command parameters XML document-->

    <!--
    <complexType name="commandType">
      <choice minOccurs="1" maxOccurs="1">
        <element ref="profile:ncl"/>
        <element ref="profile:region"/>
        <element ref="profile:rule"/>
        <element ref="profile:connector"/>
        <element ref="profile:descriptor"/>
        <element ref="profile:descriptorSwitch"/>
        <element ref="profile:transition"/>
        <element ref="profile:regionBase"/>
        <element ref="profile:ruleBase"/>
        <element ref="profile:connectorBase"/>
        <element ref="profile:descriptorBase"/>
        <element ref="profile:transitionBase"/>
        <element ref="profile:importBase"/>
        <element ref="profile:importedDocumentBase"/>
        <element ref="profile:importNCL"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
        <element ref="profile:switch"/>
        <element ref="profile:port"/>
        <element ref="profile:area"/>
        <element ref="profile:property"/>
        <element ref="profile:switchPort"/>
        <element ref="profile:link"/>
      </choice>
    </complexType>
    <element name="command" type="profile:commandType"/>
    -->

```

```

<!-- ===== -->
<!-- Structure -->
<!-- ===== -->
<!-- extends ncl element -->

<element name="ncl" substitutionGroup="structure:ncl"/>

<!-- extends head element -->

<complexType name="headType">
  <complexContent>
    <extension base="structure:headPrototype">
      <sequence>
        <element ref="profile:importedDocumentBase" minOccurs="0"
maxOccurs="1"/>
        <element ref="profile:ruleBase" minOccurs="0" maxOccurs="1"/>
        <element ref="profile:transitionBase" minOccurs="0"
maxOccurs="1"/>
        <element ref="profile:regionBase" minOccurs="0"
maxOccurs="unbounded"/>
        <element ref="profile:descriptorBase" minOccurs="0"
maxOccurs="1"/>
        <element ref="profile:connectorBase" minOccurs="0"
maxOccurs="1"/>
        <element ref="profile:meta" minOccurs="0"
maxOccurs="unbounded"/>
        <element ref="profile:metadata" minOccurs="0"
maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="head" type="profile:headType"
substitutionGroup="structure:head"/>

<!-- extends body element -->

<complexType name="bodyType">
  <complexContent>
    <extension base="structure:bodyPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:contextInterfaceElementGroup"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
        <element ref="profile:switch"/>
        <element ref="profile:link"/>
        <element ref="profile:meta"/>
        <element ref="profile:metadata"/>
      </choice>
    </extension>
  </complexContent>
</complexType>

<element name="body" type="profile:bodyType"
substitutionGroup="structure:body"/>

<!-- ===== -->
<!-- Layout -->

```



```

<!-- ===== -->
<!-- extends regionBase element -->

<complexType name="regionBaseType">
  <complexContent>
    <extension base="layout:regionBasePrototype">
      <choice minOccurs="1" maxOccurs="unbounded">
        <element ref="profile:region"/>
        <element ref="profile:importBase"/>
      </choice>
    </extension>
  </complexContent>
</complexType>

<complexType name="regionType">
  <complexContent>
    <extension base="layout:regionPrototype">
    </extension>
  </complexContent>
</complexType>

  <element name="regionBase" type="profile:regionBaseType"
substitutionGroup="layout:regionBase"/>
  <element name="region" type="profile:regionType"
substitutionGroup="layout:region"/>

<!-- ===== -->
<!-- Media -->
<!-- ===== -->
<!-- extends Media elements -->

<!-- media interface element groups -->
<group name="mediaInterfaceElementGroup">
  <choice>
    <element ref="profile:area"/>
    <element ref="profile:property"/>
  </choice>
</group>

<complexType name="mediaType">
  <complexContent>
    <extension base="media:mediaPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:mediaInterfaceElementGroup"/>
      </choice>
      <attributeGroup ref="descriptor:descriptorAttrs"/>
      <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      <attributeGroup
ref="extendedEntityReuse:extendedEntityReuseAttrs"/>
    </extension>
  </complexContent>
</complexType>

  <element name="media" type="profile:mediaType"
substitutionGroup="media:media"/>

<!-- ===== -->
<!-- Context -->
<!-- ===== -->
<!-- extends context element -->

```

```

<!-- composite node interface element groups -->
<group name="contextInterfaceElementGroup">
  <choice>
    <element ref="profile:port"/>
    <element ref="profile:property"/>
  </choice>
</group>

<complexType name="contextType">
  <complexContent>
    <extension base="context:contextPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:contextInterfaceElementGroup"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
        <element ref="profile:link"/>
        <element ref="profile:switch"/>
        <element ref="profile:meta"/>
        <element ref="profile:metadata"/>
      </choice>
      <attributeGroup ref="entityReuse:entityReuseAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="context" type="profile:contextType"
substitutionGroup="context:context"/>

<!-- ===== -->
<!-- MediaContentAnchor -->
<!-- ===== -->
<!-- extends area element -->

<complexType name="componentAnchorType">
  <complexContent>
    <extension base="mediaAnchor:componentAnchorPrototype">
      <attribute name="now" type="string" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<element name="area" type="profile:componentAnchorType"
substitutionGroup="mediaAnchor:area"/>

<!-- ===== -->
<!-- CompositeNodeInterface -->
<!-- ===== -->
<!-- extends port element -->

<complexType name="compositeNodePortType">
  <complexContent>
    <extension base="compositeInterface:compositeNodePortPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="port" type="profile:compositeNodePortType"
substitutionGroup="compositeInterface:port"/>

<!-- ===== -->

```

```

<!-- PropertyAnchor -->
<!-- ===== -->
<!-- extends property element -->

<complexType name="propertyAnchorType">
  <complexContent>
    <extension base="propertyAnchor:propertyAnchorPrototype">
    </extension>
  </complexContent>
</complexType>

  <element name="property" type="profile:propertyAnchorType"
substitutionGroup="propertyAnchor:property"/>

<!-- ===== -->
<!-- SwitchInterface -->
<!-- ===== -->
<!-- extends switchPort element -->

<complexType name="switchPortType">
  <complexContent>
    <extension base="switchInterface:switchPortPrototype">
    </extension>
  </complexContent>
</complexType>

  <element name="mapping" substitutionGroup="switchInterface:mapping"/>
  <element name="switchPort" type="profile:switchPortType"
substitutionGroup="switchInterface:switchPort"/>

<!-- ===== -->
<!-- Descriptor -->
<!-- ===== -->

<!-- substitutes descriptorParam element -->

  <element name="descriptorParam"
substitutionGroup="descriptor:descriptorParam"/>

<!-- extends descriptor element -->

<complexType name="descriptorType">
  <complexContent>
    <extension base="descriptor:descriptorPrototype">
      <attributeGroup ref="layout:regionAttrs"/>
      <attributeGroup ref="timing:explicitDurAttrs"/>
      <attributeGroup ref="timing:freezeAttrs"/>
      <attributeGroup ref="keyNavigation:keyNavigationAttrs"/>
      <attributeGroup ref="transition:transAttrs"/>
    </extension>
  </complexContent>
</complexType>

  <element name="descriptor" type="profile:descriptorType"
substitutionGroup="descriptor:descriptor"/>

<!-- extends descriptorBase element -->
<complexType name="descriptorBaseType">
  <complexContent>

```

```

    <extension base="descriptor:descriptorBasePrototype">
      <choice minOccurs="1" maxOccurs="unbounded">
        <element ref="profile:importBase"/>
        <element ref="profile:descriptor"/>
        <element ref="profile:descriptorSwitch"/>
      </choice>
    </extension>
  </complexContent>
</complexType>

<element name="descriptorBase" type="profile:descriptorBaseType"
substitutionGroup="descriptor:descriptorBase"/>

<!-- ===== -->
<!-- Linking -->
<!-- ===== -->

<!-- substitutes linkParam and bindParam elements -->
<element name="linkParam" substitutionGroup="linking:linkParam"/>
<element name="bindParam" substitutionGroup="linking:bindParam"/>

<!-- extends bind element and link element, as a consequence-->

<complexType name="bindType">
  <complexContent>
    <extension base="linking:bindPrototype">
      <attributeGroup ref="descriptor:descriptorAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="bind" type="profile:bindType"
substitutionGroup="linking:bind"/>

<!-- extends link element -->
<complexType name="linkType">
  <complexContent>
    <extension base="linking:linkPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="link" type="profile:linkType"
substitutionGroup="linking:link"/>

<!-- ===== -->
<!-- Connector -->
<!-- ===== -->
<!-- extends connectorBase element -->

<complexType name="connectorBaseType">
  <complexContent>
    <extension base="connectorBase:connectorBasePrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <element ref="profile:importBase"/>
        <element ref="profile:causalConnector" />
      </choice>
    </extension>
  </complexContent>
</complexType>

```

```

    <complexType name="simpleActionType">
      <complexContent>
        <extension
base="connectorCausalExpression:simpleActionPrototype">
          <attributeGroup ref="animation:animationAttrs"/>
        </extension>
      </complexContent>
    </complexType>

    <element name="connectorBase" type="profile:connectorBaseType"
substitutionGroup="connectorBase:connectorBase"/>

    <element name="causalConnector"
substitutionGroup="causalConnectorFunctionality:causalConnector"/>

    <element name="connectorParam"
substitutionGroup="causalConnectorFunctionality:connectorParam"/>

    <element name="simpleCondition"
substitutionGroup="causalConnectorFunctionality:simpleCondition"/>

    <element name="compoundCondition"
substitutionGroup="causalConnectorFunctionality:compoundCondition"/>

    <element name="simpleAction" type="profile:simpleActionType"
substitutionGroup="causalConnectorFunctionality:simpleAction"/>

    <element name="compoundAction"
substitutionGroup="causalConnectorFunctionality:compoundAction"/>

    <element name="assessmentStatement"
substitutionGroup="causalConnectorFunctionality:assessmentStatement"/>

    <element name="attributeAssessment"
substitutionGroup="causalConnectorFunctionality:attributeAssessment"/>

    <element name="valueAssessment"
substitutionGroup="causalConnectorFunctionality:valueAssessment"/>

    <element name="compoundStatement"
substitutionGroup="causalConnectorFunctionality:compoundStatement"/>

    <!-- ===== -->
    <!-- TestRule -->
    <!-- ===== -->
    <!-- extends rule element -->
    <complexType name="ruleType">
      <complexContent>
        <extension base="testRule:rulePrototype">
        </extension>
      </complexContent>
    </complexType>

    <element name="rule" type="profile:ruleType"
substitutionGroup="testRule:rule"/>

    <!-- extends compositeRule element -->
    <complexType name="compositeRuleType">
      <complexContent>

```

```

    <extension base="testRule:compositeRulePrototype">
    </extension>
  </complexType>

  <element name="compositeRule" type="profile:compositeRuleType"
substitutionGroup="testRule:compositeRule"/>

  <!-- extends ruleBase element -->
  <complexType name="ruleBaseType">
    <complexContent>
      <extension base="testRule:ruleBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
          <element ref="profile:rule"/>
          <element ref="profile:compositeRule"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="ruleBase" type="profile:ruleBaseType"
substitutionGroup="testRule:ruleBase"/>

  <!-- ===== -->
  <!-- TestRuleUse -->
  <!-- ===== -->
  <!-- extends bindRule element -->
  <complexType name="bindRuleType">
    <complexContent>
      <extension base="testRuleUse:bindRulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="bindRule" type="profile:bindRuleType"
substitutionGroup="testRuleUse:bindRule"/>

  <!-- ===== -->
  <!-- ContentControl -->
  <!-- ===== -->
  <!-- extends switch element -->

  <!-- switch interface element groups -->
  <group name="switchInterfaceElementGroup">
    <choice>
      <element ref="profile:switchPort"/>
    </choice>
  </group>

  <!-- extends defaultComponent element -->
  <complexType name="defaultComponentType">
    <complexContent>
      <extension base="contentControl:defaultComponentPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="defaultComponent" type="profile:defaultComponentType"
substitutionGroup="contentControl:defaultComponent"/>

```

```

<complexType name="switchType">
  <complexContent>
    <extension base="contentControl:switchPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:switchInterfaceElementGroup"/>
        <element ref="profile:bindRule"/>
        <element ref="profile:switch"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
      </choice>
      <attributeGroup ref="entityReuse:entityReuseAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="switch" type="profile:switchType"
substitutionGroup="contentControl:switch"/>

<!-- ===== -->
<!-- DescriptorControl -->
<!-- ===== -->
<!-- extends defaultDescriptor element -->
<complexType name="defaultDescriptorType">
  <complexContent>
    <extension base="descriptorControl:defaultDescriptorPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="defaultDescriptor" type="profile:defaultDescriptorType"
substitutionGroup="descriptorControl:defaultDescriptor"/>

<!-- extends descriptorSwitch element -->

<complexType name="descriptorSwitchType">
  <complexContent>
    <extension base="descriptorControl:descriptorSwitchPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <element ref="profile:descriptor"/>
        <element ref="profile:bindRule"/>
      </choice>
    </extension>
  </complexContent>
</complexType>

<element name="descriptorSwitch" type="profile:descriptorSwitchType"
substitutionGroup="descriptorControl:descriptorSwitch"/>

<!-- ===== -->
<!-- Timing -->
<!-- ===== -->

<!-- ===== -->
<!-- Import -->
<!-- ===== -->
<complexType name="importBaseType">
  <complexContent>
    <extension base="import:importBasePrototype">

```

```

    </extension>
  </complexContent>
</complexType>

<complexType name="importNCLType">
  <complexContent>
    <extension base="import:importNCLPrototype">
    </extension>
  </complexContent>
</complexType>

<complexType name="importedDocumentBaseType">
  <complexContent>
    <extension base="import:importedDocumentBasePrototype">
    </extension>
  </complexContent>
</complexType>

  <element name="importBase" type="profile:importBaseType"
substitutionGroup="import:importBase"/>

  <element name="importNCL" type="profile:importNCLType"
substitutionGroup="import:importNCL"/>
  <element name="importedDocumentBase"
type="profile:importedDocumentBaseType"
substitutionGroup="import:importedDocumentBase"/>

  <!-- ===== -->
  <!-- EntityReuse -->
  <!-- ===== -->

  <!-- ===== -->
  <!-- ExtendedEntityReuse -->
  <!-- ===== -->

  <!-- ===== -->
  <!-- KeyNavigation -->
  <!-- ===== -->

  <!-- ===== -->
  <!-- TransitionBase -->
  <!-- ===== -->
  <!-- extends transitionBase element -->

  <complexType name="transitionBaseType">
    <complexContent>
      <extension base="transitionBase:transitionBasePrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="profile:transition"/>
          <element ref="profile:importBase"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="transitionBase" type="profile:transitionBaseType"
substitutionGroup="transitionBase:transitionBase"/>

  <!-- ===== -->
  <!-- Transition -->

```



```
<!-- ===== -->

<element name="transition" substitutionGroup="transition:transition"/>

<!-- ===== -->
<!-- Metainformation -->
<!-- ===== -->

<element name="meta" substitutionGroup="metainformation:meta"/>

<element name="metadata" substitutionGroup="metainformation:metadata"/>

</schema>
```

6. Final Remarks

In order to offer a scalable hypermedia model, with characteristics that may be progressively incorporated in hypermedia system implementations, NCM was divided in several parts, and also its declarative XML application language: NCL. This technical report deals with the editing commands used for live authoring NCL documents, which comprises Part 9: NCL Live Editing Commands.

References

- [Anto00] Antonacci M.J. NCL: Uma Linguagem Declarativa para Especificação de Documentos Hipermídia com Sincronização Temporal e Espacial. **Master Dissertation, Departamento de Informática, PUC-Rio**, April 2000.
- [AMRS00] Antonacci M.J., Muchaluat-Saade D.C., Rodrigues R.F., Soares L.F.G. NCL: Uma Linguagem Declarativa para Especificação de Documentos Hipermídia na Web, **VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia - SBMídia2000**, Natal, Rio Grande do Norte, June 2000.
- [ISO98] ISO/IEC 13818-6, Information technology - Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC. 1998/Cor 2:2002.
- [RDF99] Resource Description Framework (RDF) Model and Syntax Specification, Ora Lassila and Ralph R. Swick. W3C Recommendation, 22 February 1999. Available at <http://www.w3.org/TR/REC-rdf-syntax/>
- [SCHE01] XML Schema Part 0: Primer, **W3C Recommendation**, in <http://www.w3.org/TR/xmlschema-0/>, May 2001.
- [SoRo05] Soares L.F.G; Rodrigues R.F. Nested Context Model 3.0: Part 1 – NCM Core, **Technical Report, Departamento de Informática PUC-Rio**, May 2005, ISSN: 0103-9741.
- [SoRo06] Soares L.F.G; Rodrigues R.F. Nested Context Language 3.0: Part 8 – NCL Digital TV Profiles, **Technical Report, Departamento de Informática PUC-Rio**, October 2006, ISSN: 0103-9741.
- [XML98] Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. Extensible Markup Language (XML) 1.0 (Second Edition), **W3C Recommendation**, in <http://www.w3.org/TR/REC-xml>, February 1998.

Appendix A – DSM-CC Transport of Editing Commands using Stream-event Descriptors and Object Carousels

In Digital TV environments, it is usual to adopt DSM-CC to transport editing commands in MPEG-2 TS elementary streams.

Editing commands are transported in DSM-CC stream-event descriptors. As specified on [ISO98], a DSM-CC stream-event descriptor has a very similar structure to that of an event descriptor presented in Figure 4.1 (see Figure A.1).

Syntax	Number of bits
StreamEventDescriptor () {	
descriptorTag	8
descriptorLength	8
eventId	16
reserved	31
eventNPT	33
privateDataLength	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 to 2008
FCS	8
}	

Figure A.1 - Editing command stream event descriptor

The DSM-CC object carousel protocol allows the cyclical transmission of event objects and file systems. Event objects are used to map stream event names into stream event ids defined in event descriptors. Event objects are used to inform about DSM-CC stream events that may be received. Event names allow specifying types of events, offering a higher abstraction level for applications. The Private Base Manager should register itself as a listener of stream events it handles using event names; in the case of editing commands the name “*nclEditingCommand*”.

As aforementioned, besides event objects the DSM-CC object carousel protocol can also be used to transport files organized in directories. A DSM-CC demultiplexer is responsible for mounting the file system at the receiver device. XML-based *command parameters* specified as XML documents (NCL documents or an NCL entities to be added) can thus be organized in file system structures to be transported in these carousels, as an alternative to the direct transportation in the payload of stream event descriptors. A DSM-CC carousel generator is used to join the file systems and stream event objects into data elementary streams.

Thus, when an NCL editing command needs to be sent, a DSM-CC event object shall be created, mapping the string “*nclEditingCommand*” into a selected stream event id, and shall be put in a DSM-CC object carousel sent in a elementary stream of type = “0x0B”. One or more DSM-CC stream event descriptors with a previous selected event id are then created and sent in another MPEG-2 TS elementary stream. These stream events usually have their time reference set to zero, but may be postponed to be executed at a specific time. The Private Base Manager shall register itself as an “*nclEditingCommand*” listener in order to be notified when this kind of stream event arrives.

The *commandTag* of the received stream event descriptor is then used by the Private Base Manager to interpret the complete *command string* semantics. If the XML-based *command parameter* is short enough it is transported directly in the stream event descriptor payload. Otherwise, the *privateDataPayload* field carries a set of reference pairs. In this case, the XML specification shall be placed in the same object carousel that carries the event object. The *uri* parameter of the first reference pair shall have the schema (optional) and the absolute path of the XML specification (the path in the data server). The corresponding *id* parameter in the pair shall refer to the XML specification IOR (carouselId, moduleId, objectKey; see [ISO98]) in the object carousel. If other file systems needs to be transmitted using other object carousels to complete the editing command with media contents (as it is usual in the case of *addDocument* or *addNode* commands), other {uri, id} pairs shall be present in the command. In this case, the *uri* parameter shall have the schema (optional) and the absolute path of file system root (the path in the datacast server), and the corresponding *id* parameter in the pair shall refer to the IOR (carouselId, moduleId, objectKey) of any root child file or child directory in the object carousel (the carousel service gateway).

Figure A.2 describes an example of interactive application transmission using this scheme for the Brazilian Digital TV system. In the example, the content provider wishes to send an interactive application (“trafficConditions.ncl”) stored in one of the broadcaster datacasting servers (in a local file system), but referring to resources in different hard disks or partitions, as illustrated in figure. Two object carousels are generated to carry the interactive content (Service Domain = 0x1 and Service Domain = 0x2). Besides the application and part of their content, the object carousel represented by Service Domain = 1 carries an event object. This object relates the event type “nclEditingCommand” with the event identifier “0x3”.

The stream event descriptor (Figure A.2) is transmitted with the identifier “0x3”, as specified by the event object of carousel “0x1”. In this event, the value “0x0” is assigned to the temporal reference (eventNPT field in Figure A.2), requiring its immediate execution. The private data field carries the “addDocument” command code and a set of <URL, ID> pairs. In the first pair, the URL has the “x-sbtvd” scheme and the local absolute path “E:\newNclRepository\traffic” related with the interactive application. The associated ID locates the object that carries the application: Service Domain “0x1”, module “0x1” and object “0x2”. In the second pair, the URL has the “x-sbtvd” scheme and the “L:\media” path, while the associated ID locates the image “southRegion.jpg” in Service Domain “0x2”, module “0x1” and object “0x2”. Using this information, a Private Base Manager can store the received absolute path of each received URL, relating them with the local memory area URL where the objects, passed on each transmitted IOR, are (or will be) mounted.

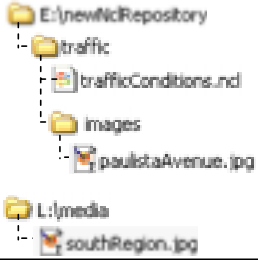
Local File System	Service Domain = 0x1		Service Domain = 0x2
	moduleId = 0x1 ... objectKey = 0x1 objectKind = srg 2 bindings binding #1 objectName = trafficConditions.ncl objectType = fil IOR = 0x1,0x1,0x2 binding #2 objectName = images objectType = dir IOR = 0x1,0x1,0x3 ...	moduleId = 0x2 objectKey = 0x1 objectKind = fil data ... objectKey = 0x2 objectKind = ste eventList eventName = nclEditingCommand eventId = 0x3 ...	moduleId = 0x1 ... objectKey = 0x1 objectKind = srg 1 binding binding #1 objectName = southRegion.jpg objectType = fil IOR = 0x2,0x1,0x2 ... objectKey = 0x2 objectKind = fil data
Stream Event Descriptor descriptorTag = streamEventTag() descriptorLength = descriptorLen() eventId = 0x3 reserved eventNPT = 0 privateDataLength = dataLen() privateData = "0x05", "x-sbtdv://E:/newNclRepository/traffic", "0x1,0x1,0x2", "x-sbtdv://L:/media", "0x2,0x1,0x2"	... objectKey = 0x2 objectKind = fil data ... objectKey = 0x3 objectKind = dir 1 binding binding #1 objectName = paulistaAvenue.jpg objectType = fil IOR = 0x1,0x2,0x1		

Figure A.2 - Example of the proposed mechanism

Note that the resource identification is performed in the receiver exactly as it is performed in the authoring environment, in spite of if the interactive application works with absolute or relative URLs, or if the object carousel being used refers to resources in other carousels. For example, the “trafficConditions.ncl” application could refer to the “brazilianMap.jpg” relatively or absolutely. Furthermore, the application could refer to a content stored in a different provider, which would be loaded through a different object carousel. The resource location mapping is automatically performed by the Private Base Manager.

Appendix B –Transport of Editing Commands Using Specific Structures defined by Ginga-NCL

Event descriptors (defined in Section 4) can be sent in MPEG-2 TS elementary stream, using DSM-CC stream event, as discussed in appendix A, or using any protocol for pushed data transmission.

Three data structure types are defined to support the transmission of NCL editing command parameters: maps, metadata and data files.

For map structures, the *mappingType* field identifies the map type. If the *mappingType* is equal to “0x01” (“events”), an event-map is characterized. In this case, after the *mappingType* field comes a list of event identifiers as defined in Table B.1. Other *mappingType* values may also be defined, but they are not relevant for this discussion.

Table B.1 – List of event identifiers defined by the mapping structure

Syntax	Number of bits
mappingStructure () {	
mappingType	8
for (i=1; i<N; i++){	
eventId	8
eventNameLength	8
eventName	8 to 255
}	
}	

Maps of type “events” (*event maps*) are used to map event names into *eventIds* of event descriptors (see Figure 4.1). Event maps are used to inform which events shall be received. Event names allow specifying types of events, offering a higher abstraction level for middleware applications. The Private Base Manager, as well as execution-objects (e.g. NCLua, NCLet), should register themselves as listeners of events they handle, using event names.

When an NCL editing command needs to be sent, an event map shall be created, mapping the string “*nclEditingCommand*” into a selected event descriptor id (see Figure 4.1). One or more event descriptors with the previous selected *eventId* are then created and sent². These event descriptors may have their time reference set to zero, but may be postponed to be executed at a specific time. The Private Base Manager shall register itself as an “*nclEditingCommand*” listener in order to be notified when this type of event arrives.

² For example, it can be sent in an MPEG-2 TS elementary stream, or using some protocol for pushed data transmission.

Each data file structure is indeed a file content that composes an NCL application or an NCL entity specification: the XML specification file or its media content files (video, audio, text, image, ncl, lua, etc.).

A metadata structure is an XML document, as defined by the following schema. Note that the schema defines, for each pushed file, an association between its location in a transport system (transport system identification (*component_tag* attribute) and the file identification in the transport system (*structureId* attribute)) and its Universal Resource Identifier (*uri* attribute).

```
<!--
XML Schema for NCL Section Metadata File

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCLSectionMetadataFile.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Section Metadata File namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:NCLSectionMetadataFile="http://www.ncl.org.br/
                                     NCLSectionMetadataFile"
  targetNamespace="http:// www.ncl.org.br/NCL3.0/
                                     NCLSectionMetadataFile"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="NCLSectionMetadataType">
    <sequence>
      <sequence>
        <element ref="NCLSectionMetadataFile:baseData" minOccurs="0"
          maxOccurs="unbounded"/>
      </sequence>
      <element ref="NCLSectionMetadataFile:pushedRoot" minOccurs="0"
        maxOccurs="1"/>
      <sequence>
        <element ref="NCLSectionMetadataFile:pushedData" minOccurs="0"
          maxOccurs="unbounded"/>
      </sequence>
    </sequence>
    <attribute name="name" type="string" use="optional"/>
    <attribute name="size" type="positiveInteger" use="optional"/>
  </complexType>

  <complexType name="baseDataType">
    <sequence>
      <element ref="NCLSectionMetadataFile:pushedRoot" minOccurs="0"
        maxOccurs="1"/>
      <sequence>
        <element ref="NCLSectionMetadataFile:pushedData"
          minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </sequence>
    <attribute name="uri" type="anyURI" use="required"/>
  </complexType>
</schema>
```



```

</complexType>

<complexType name="pushedRootType">
  <attribute name="component_tag" type="positiveInteger"
            use="optional"/>
  <attribute name="structureId" type="string" use="required"/>
  <attribute name="uri" type="anyURI" use="required"/>
  <attribute name="size" type="positiveInteger" use="optional"/>
</complexType>

<complexType name="pushedDataType">
  <attribute name="component_tag" type="positiveInteger"
            use="optional"/>
  <attribute name="structureId" type="string" use="required"/>
  <attribute name="uri" type="anyURI" use="required"/>
  <attribute name="size" type="positiveInteger" use="optional"/>
</complexType>

<!-- declare global elements in this module -->
<element name="metadata"
          type="NCLSectionMetadataFile:NCLSectionMetadataType"/>
<element name="baseData"
          type="NCLSectionMetadataFile:baseDataType"/>
<element name="pushedRoot"
          type="NCLSectionMetadataFile:pushedRootType"/>
<element name="pushedData"
          type="NCLSectionMetadataFile:pushedDataType"/>

</schema>

```

For each NCL Document file or other XML Document files used in *addDocument* or *addNode* editing command parameters, at least one metadata structure shall be defined. Only one NCL application file or XML document file representing an NCL node to be inserted may be defined in a metadata structure. More precisely, there can be only one `<pushedRoot>` element in a metadata XML document. However, an NCL application (and its content files) or an XML document (and its content files) may extend for more than one metadata structure. Moreover, there may also be a metadata structure without any NCL application or XML document described in its `<pushedRoot>` and `<pushedData>` elements.

These three data structures can be transmitted using different transport systems, as exemplified in what follows.

B.1 Transporting all data structures in a specific MPEG-2 section type

The use of a specific type of MPEG-2 section (identified by a specific `table_id` value, present in the `table_id` field of an MPEG-2 private section), from now on called NCL Section, may allow the transmission of the three data structure types: maps, metadata and data files.

Every NCL Section contains data of a single structure. However, one structure can extend through several Sections. Every data structure can be transmitted in any order and how many times it is necessary. The beginning of a data structure is delimited by the `payload_unit_start_indicator` field of a TS packet. After the four bytes of the TS header the TS packet payload starts with a `pointer_field` byte indicating the beginning of an NCL

Section (see ISO/IEC 13818-1). The NCL Section header is then defined as MPEG-2 sections (see ISO/IEC 13818-1). The first byte of an NCL Section payload identifies the structure type (0x01 for metadata; 0x02 for data files, and 0x03 for event-map). The second payload byte carries the unique identifier of the structure (*structureId*) in this elementary stream³. After the second byte comes a serialized data structure that can be a mappingStructure (as depicted by Table B.1), or a metadata structure (an XML document), or a data file structure (a serialized file content). The NCL Section demultiplexer is responsible for mounting the application's structure at the receiver device.⁴

In the same elementary stream that carries the XML specification (the NCL Document file or other XML Document file used in NCL editing commands), an event-map file should be transmitted in order to map the name "*nclEditingCommand*" to the *eventId* of the event descriptor, which shall carry an NCL editing command, as described in Section 4. The *privateDataPayload* of the event descriptor shall carry a set of {uri, id} reference pairs. The *uri* parameters are always "null". In the case of *addDocument* and *addNode* commands, the *id* parameter of the first pair shall identify the elementary stream ("component_tag") and its metadata structure ("structureId") that carries the absolute path of the NCL document or the NCL node specification (the path in the data server) and the corresponding related structure ("structureId") transported in NCL Sections of the same elementary stream. If other additional metadata structures are used in order to complete the *addDocument* or *addNode* command, other {uri, id} pairs shall be present in the command. In this case, the *uri* parameter shall also be "null" and the corresponding id parameter in the pair shall refer to the component_tag and the corresponding metadata structureId.

Figure B.1 depicts an example of an NCL document transmission through NCL Sections. In this example, a content provider wants to transmit an interactive program named "weatherConditions.ncl" stored in one of its data servers (Local File System, in Figure B.1). An MPEG-2 elementary stream (component_tag= "0x09") shall then be generated carrying all the interactive program contents (ncl file and all media content files) and also an event-map (structureType="0x03"; structureId="0x0C"), in Figure B.1), mapping the "*nclEditingCommand*" name to the *eventId* value (value "3", in Figure B.1). An event descriptor shall also be transmitted with the appropriated *eventId* value, in the example "3", and the *commandTag* value "0x05", which indicates an *addDocument* command (see Section 4). The *uri* parameter shall have the "null" value and the *id* parameter shall have the (component_tag= "0x09", structureId= "0x0B", in Figure B.1) value.

³ The elementary stream and the structure identifier are those that are associated by the metadata structure to a file locator (URL), through the *component_tag* and *structureId* attributes of the <pushedRoot> and <pushedData> elements.

⁴ It is important to note that NCL Sections can also transport data structures encapsulated in other data structures. For example, MPE (Multi-protocol Encapsulation) can be used and thus, in this case, NCL Sections are MPEG-2 Datagram Sections. Moreover all data structures mentioned in this appendix can be wrapped in other protocol data format, like FLUTE packets.

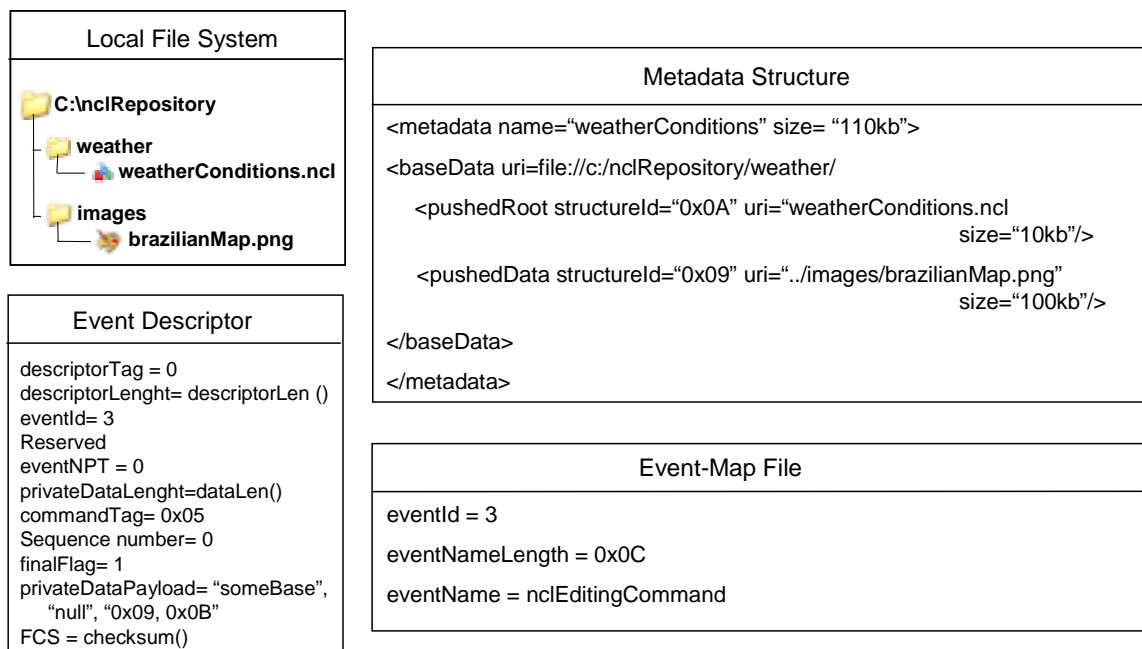


Figure B.1 – Example of an NCL document transmission using MPEG-2 NCL Section

B.2 Transporting metadata structures as Editing Command parameter

Instead of transporting metadata structures directly inside NCL sections, an alternative procedure is treating metadata structures as *addDocument* and *addNode* command parameters, which are transported in the *privateDataPayload* field of an event descriptor.

In this case, the set of {uri, id} parameter pairs of *addDocument* and *addNode* command is substituted by metadata structure parameters that define a set of {"uri", "component_tag, structureId"} pairs for each pushed file.

Taking back the example of Figure B.1, the new scenario would be exactly the same, except by the event descriptor. Instead of having the {uri; id} pair = {"null"; "0x09,0x0B"} value as an event descriptor parameter, it would have the serialized XML metadata structure. In the metadata structure, the *component-tag* attribute of the <pushedRoot> and <pushedData> elements shall in this case be defined, since the metadata structure is not transported anymore in the same elementary stream of the NCL document's files.

B.3 Transporting metadata structures in MPEG-2 metadata sections

Another alternative is transporting metadata structures in MPEG-2 metadata sections, transported in MPEG-2 stream type = "0x16". As usual, every MPEG-2 metadata section contains data of a single metadata structure. However, one metadata structure can extend through several metadata sections.

Table B.2 shows the metadata section syntax for transport of metadata structures, which shall be in agreement with ISO/IEC 13818-1: 2007.

Table B.2 – Section syntax for transport of metadata structures

Syntax	Nº. of bits	Value
Metadata section() {		
table_id	8	0x06
section_syntax_indicator	1	1
private_indicator	1	1
random_access_indicator	1	1
decoder_config_flag	1	0
metadata_section_length	12	integer
metadata_service_id	8	Integer to be standardized
reserved	8	
section_fragment_indication	2	according to Table B.3
version_number	5	integer
current_next_indicator	1	1
section_number	8	integer
last_section_number	8	integer
structureId	8	integer
For (i=1; i< N; i++) {		
serialized_metadata_structure_byte	8	
}		
CRC_32	32	
}		

Table B.3 – Section fragment indication

Value	Description
11	A single metadata section carrying a complete metadata structure.
10	The first metadata section from a series of metadata sections with data from one metadata structure.
01	The last metadata section from a series of metadata sections with data from one metadata structure.
00	A metadata section from a series of metadata sections with data from one metadata structure, but neither the first nor the last one.

As previously, in the same elementary stream that carries the XML specification (the NCL Document file or other XML Document file used in NCL editing commands), an event-map file should be transmitted in order to map the name “*nclEditingCommand*” to the *eventId* of the event descriptor, which shall carry an NCL editing command, as described in Section 4. The *privateDataPayload* of the event descriptor shall carry a set of {uri, id} reference pairs. The *uri* parameters are always “null”. In the case of *addDocument* and *addNode*

commands, the *id* parameter of the first pair shall identify the elementary stream (“component_tag”) of type= “0x16” and the metadata structure (“structureId”) that carries the absolute path of the NCL document or the NCL node specification (the path in the data server). If other metadata structures are used to relate files present in the NCL document or the NCL node specification, in order to complete the *addDocument* or *addNode* command with media content, other {uri, id} pairs shall be present in the command. In this case, the *uri* parameter shall also be “null” and the corresponding *id* parameter in the pair shall refer to the component_tag and the corresponding metadata structureId.

Taking back the example of Figure B.1, the new scenario would be very similar. Only minor changes must be made such that the event descriptor refers to the elementary stream and its section that carries the metadata structure (“component_tag= “0x08” and structureId= “0x0B”), and that the metadata structure also refers to the elementary stream where the document’s file will be transported. Figure B.2 illustrates the new situation.

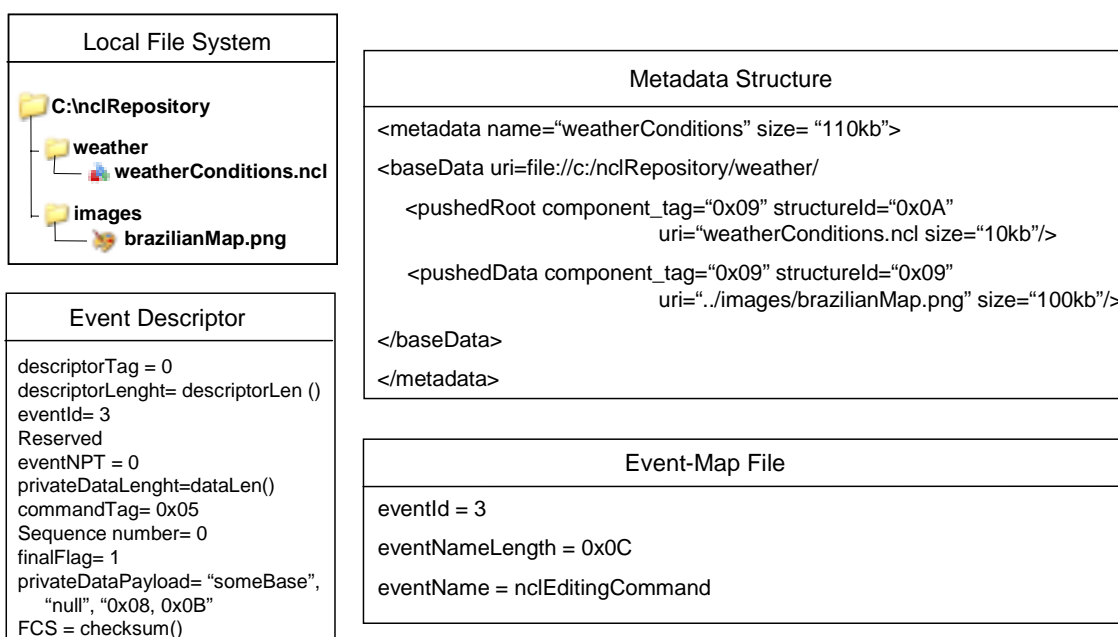


Figure B.2 – Example of an NCL document transmission using MPEG-2 Metadata Section